

Unit1

1.Distributed Database System in DBMS

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Types:

1.Homogeneous

Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

2.

Heterogeneous

Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Distributed

Data

Storage :

There are 2 ways in which data can be stored on different sites. These are:

1.

Replication –

In this approach, the entire relationship is stored redundantly at 2 or more sites.

If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

This is advantageous as it increases the availability of data at different sites. Also, now query requests can be processed in parallel. However, it has certain disadvantages as well. Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

2. **Fragmentation –**

In this approach, the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. 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). Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

Fragmentation of relations can be done in two ways:

- **Horizontal fragmentation – Splitting by rows –**
The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.
- **Vertical fragmentation – Splitting by columns –**
The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure a lossless join.

In certain cases, an approach that is hybrid of fragmentation and replication is used.

Applications of Distributed Database:

- It is used in Corporate Management Information System.
- It is used in multimedia applications.
- Used in Military's control system, Hotel chains etc.
- It is also used in manufacturing control system.

References

:

[Database System Concepts by Silberschatz, Korth and Sudarshan](#)

A distributed database system is a type of database management system that stores data across multiple computers or sites that are connected by a network. In a distributed database system, each site has its own database, and the databases are connected to each other to form a single, integrated system.

The main advantage of a distributed database system is that it can provide higher availability and reliability than a centralized database system. Because the data is stored across multiple sites, the system can continue to function even if one or more sites fail. In addition, a distributed database system can provide better performance by distributing the data and processing load across multiple sites.

There are several different architectures for distributed database systems, including:

Client-server architecture: In this architecture, clients connect to a central server, which manages the distributed database system. The server is responsible for coordinating transactions, managing data storage, and providing access control.

Peer-to-peer architecture: In this architecture, each site in the distributed database system is connected to all other sites. Each site is responsible for managing its own data and coordinating transactions with other sites.

Federated architecture: In this architecture, each site in the distributed database system maintains its own independent database, but the databases are integrated through a middleware layer that provides a common interface for accessing and querying the data.

Distributed database systems can be used in a variety of applications, including e-commerce, financial services, and telecommunications. However, designing and managing a distributed database system can be complex and requires careful consideration of factors such as data distribution, replication, and consistency.

Advantages of Distributed Database System :

- 1) There is fast data processing as several sites participate in request processing.
- 2) Reliability and availability of this system is high.
- 3) It possess reduced operating cost.
- 4) It is easier to expand the system by adding more sites.
- 5) It has improved sharing ability and local autonomy.

Disadvantages of Distributed Database System :

- 1) The system becomes complex to manage and control.
- 2) The security issues must be carefully managed.
- 3) The system require deadlock handling during the transaction processing otherwise the entire system may be in inconsistent state.
- 4) There is need of some standardization for processing of distributed database system.

3.Distributed Database Architecture in DBMS

Distributed Database System:

A Distributed Database System is a kind of database that is present or divided in more than one location, which means it is not limited to any single computer system. It is divided over the network of various systems. The Distributed Database System is physically present on the different systems in different locations. This can be necessary when different users from all over the world need to access a specific database. For a user, it should be handled in such a way that it seems like a single database.

Parameters of Distributed Database Systems:

- **Distribution:**

It describes how data is physically distributed among the several sites.

- **Autonomy:**

It reveals the division of power inside the Database System and the degree of autonomy enjoyed by each individual DBMS.

- **Heterogeneity:**

It speaks of the similarity or differences between the databases, system parts, and data models.

Common Architecture Models of Distributed Database Systems:

- **Client-Server Architecture of DDBMS:**

This architecture is two level architecture where clients and servers are the points or levels where the main functionality is divided. There is various functionality

provided by the server, like managing the transaction, managing the data, processing the queries, and optimization.

- **Peer-to-peer Architecture of DDBMS:**

In this architecture, each node or peer is considered as a server as well as a client, and it performs its database services as both (server and client). The peers coordinate their efforts and share their resources with one another.

- **Multi DBMS Architecture of DDBMS:**

This is an amalgam of two or more independent Database Systems that functions as a single integrated Database System.

Types of Distributed Database Systems:

- **Homogeneous Database System:**

Each site stores the same database in a Homogenous Database. Since each site has the same database stored, so all the data management schemes, operating system, and data structures will be the same across all sites. They are, therefore, simple to handle.

- **Heterogeneous Database System:**

In this type of Database System, different sites are used to store the data and relational tables, which makes it difficult for database administrators to do the transactions and run the queries into the database. Additionally, one site might not even be aware of the existence of the other sites. Different operating systems and database applications may be used by various computers. Since each system has its own database model to store the data, therefore it is required there should be

translation schemes to establish the connections between different sites to transfer the data.

Distributed Data Storage:

There are two methods by which we can store the data on different sites:

- **Replication:**

This method involves redundantly storing the full relationship at two or more locations. Since a complete database can be accessed from each site, it becomes a redundant database. Systems preserve copies of the data as a result of replication.

This has advantages because it makes more data accessible at many locations. Additionally, query requests can now be handled in parallel.

However, there are some drawbacks as well. Data must be updated frequently. Any changes performed at one site must be documented at every site where that relation is stored in order to avoid inconsistent results. There is a tonne of overhead here. Additionally, since concurrent access must now be monitored across several sites, concurrency control becomes far more complicated.

- **Fragmentation:**

According to this method, the relationships are divided (i.e., broken up into smaller pieces), and each fragment is stored at the many locations where it is needed. To ensure there is no data loss, the pieces must be created in a way that allows for the reconstruction of the original relation.

Since Fragmentation doesn't result in duplicate data, consistency is not a concern.

Ways of fragmentation:

- **Horizontal Fragmentation:**

In Horizontal Fragmentation, the relational table or schema is broken down into a group of one and more rows, and each row gets one fragment of the schema. It is also called **splitting by rows**.

- **Vertical Fragmentation:**

In this fragmentation, a relational table or schema is divided into some more schemas of smaller sizes. A common candidate key must be present in each fragment in order to guarantee a lossless join. This is also called **splitting by columns**.

Note: Most of the time, a hybrid approach of replication and fragmentation is used.

Application of Distributed Database Systems:

- Multimedia apps use it.
- The manufacturing control system also makes use of it.
- Another application is by corporate management for the information system.
- It is used in hotel chains, military command systems, etc.

4. Concepts of Distributed databases

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)

5. Why is Distributed Storage Important?

- **Software-defined** - distributed storage replaces the traditional centralized SAN and NAS with a software-defined storage platform that empowers customers to deploy, manage, and scale a single, unified storage platform

across datacenters, branch offices, or the cloud. An integrated distributed storage platform enables seamless access to storage by delivering files, objects and volumes across multiple protocols to all workloads and users.

- **Access to all protocols** - customers have been procuring Files, Objects and Volumes as point solutions and are managed by independent teams. A distributed storage system offers simplicity, consolidating all three access types on a single platform, helping customers to deploy the storage services at core/edge or to extend to the cloud. In addition, all three storage services are managed and monitored centrally.
- **Scale-out architecture** - unlike traditional storage arrays, distributed storage is by design a scale-out architecture. You can add as many nodes as you want, increasing the storage capacity ad infinitum.
- **Faster provisioning** - since the distributed storage system creates a shared pool of storage resources from a number physical nodes, storage policies can be created and attached to virtual machines that can instantaneously leverage resources from the dynamic storage pools. This makes faster storage provisioning unlike traditional storage where an admin has to create a volume/file share and attach to the virtual machine manually.
- **Simplified management and monitoring** - distributed storage system offers simple management and monitoring with dashboards, data analytics tools, etc.

Distributed Storage Features

While features can vary across cloud storage providers, most distributed cloud storage systems include:

- **Partitioning** – allows users to spread data across cluster nodes and easily access data from those nodes

- **Replication** – data is copied across a variety of nodes and are updated consistently whenever that data is modified
- **Resiliency** – data remains available, even if one or multiple nodes malfunction
- **Easy scaling** – system operators can scale storage capacity up or down as needed, simply by adding or removing nodes to the cluster

Pros and Cons of Distributed Cloud Storage

Distributed cloud storage has a number of advantages and benefits:

- **Aids regulatory compliance** – many regulations limit organizations from moving sensitive data across borders; now they can more easily keep country data in-country, for instance
- **More ambiguous attack surface** – because there are no “central” servers, there’s no obvious target for attack by bad actors
- **Reduced risk of network failure** – because data is stored in local or regional clusters, they can sometimes run separately—which increases fault tolerance
- **Enhanced privacy** – data files are split apart, encrypted, and stored across a network of servers
- **Reduced energy costs** – there’s no need to build and cool a massive centralized data center

Challenges arise primarily from the distributed nature of this storage model:

- **Bandwidth** – made up of a variety of cloud storage types and systems, distributed cloud storage might have a range of different connectivity models, which can put strain on edge-located internet connections

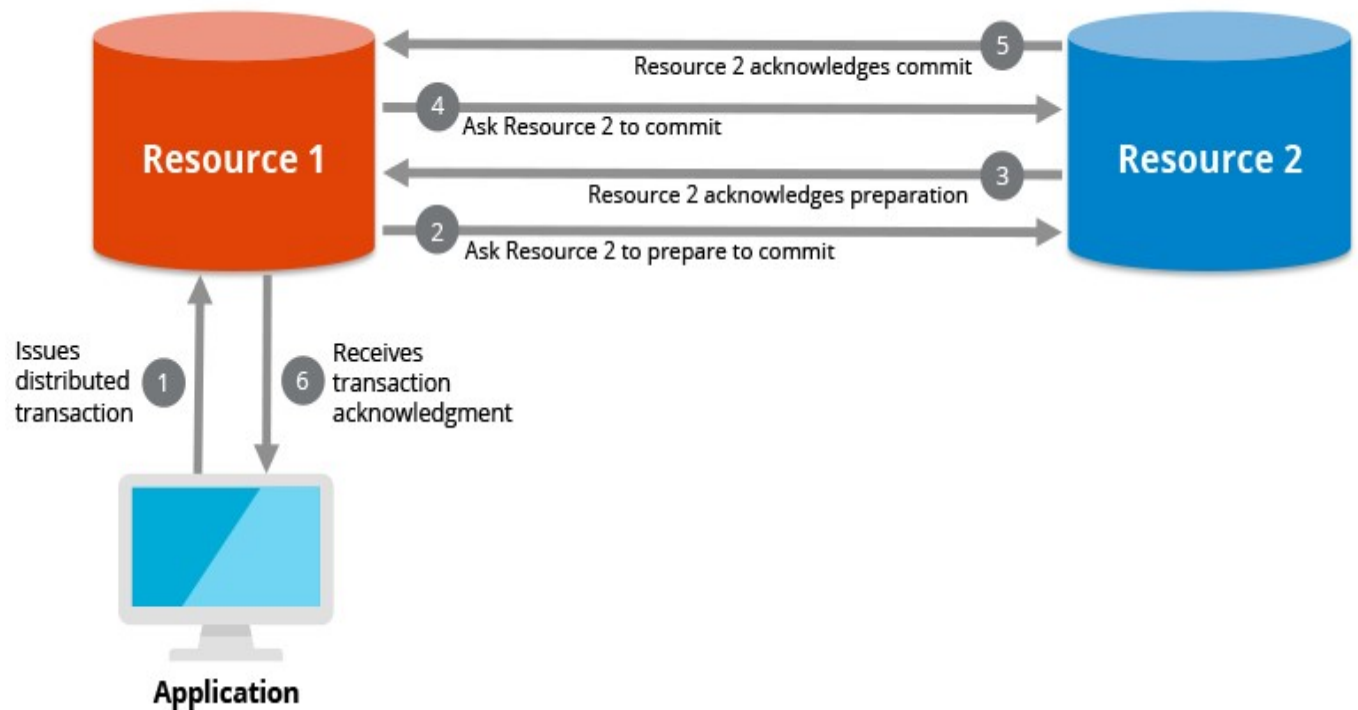
- **Security** – ensuring data is secure across varying cloud storage types spread across the world can be difficult
- **Data protection** – backup and business continuity can get tricky, especially when it comes to making sure geography-limited data stays where it should

6.What is a Distributed Transaction?

A **distributed transaction** is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network, but may also span multiple databases on a single server.

There are two possible outcomes: 1) all operations successfully complete, or 2) none of the operations are performed at all due to a failure somewhere in the system. In the latter case, if some work was completed prior to the failure, that work will be reversed to ensure no net work was done. This type of operation is in compliance with the “ACID” (atomicity-consistency-isolation-durability) principles of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions extend that guarantee across multiple databases.

The operation known as a “two-phase commit” (2PC) is a form of a distributed transaction. “XA transactions” are transactions using the XA protocol, which is one implementation of a two-phase commit operation.



A distributed transaction spans multiple databases and guarantees data integrity.

How Do Distributed Transactions Work?

Distributed transactions have the same processing completion requirements as regular database transactions, but they must be managed across multiple resources, making them more challenging to implement for database developers. The multiple resources add more points of failure, such as the separate software systems that run the resources (e.g., the database software), the extra hardware servers, and network failures. This makes distributed transactions susceptible to failures, which is why safeguards must be put in place to retain data integrity.

For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager can be one of the data repositories that will be updated

as part of the transaction, or it can be a completely independent separate resource that is only responsible for coordination. The transaction manager decides whether to commit a successful transaction or rollback an unsuccessful transaction, the latter of which leaves the database unchanged.

First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own “resource manager” to help it participate in distributed transactions. Distributed transactions are often done in two phases to safeguard against partial updates that might occur when a failure is encountered. The first phase involves acknowledging an intent to commit, or a “prepare-to-commit” phase. After all resources acknowledge, they are then asked to run a final commit, and then the transaction is completed.

We can examine a basic example of what happens when a failure occurs during a distributed transaction. Let’s say one or more of the resources become unavailable during the prepare-to-commit phase. When the request times out, the transaction manager tells each resource to delete the prepare-to-commit status, and all data will be reset to its original state. If instead, any of the resources become unavailable during the commit phase, then the transaction manager will tell the other resources that successfully committed their portion of the transaction to undo or “rollback” that transaction, and once again, the data is back to its original state. It is then up to the application to retry the transaction to make sure it gets completed.

Why Do You Need Distributed Transactions?

Distributed transactions are necessary when you need to quickly update related data that is spread across multiple databases. For example, if you have multiple

systems that track customer information and you need to make a universal update (like updating the mailing address) across all records, a distributed transaction will ensure that all records get updated. And if a failure occurs, the data is reset to its original state, and it is up to the originating application to resubmit the transaction.

7.Two-phase commit protocol

In [transaction processing](#), [databases](#), and [computer networking](#), the **two-phase commit protocol (2PC)** is a type of [atomic commitment protocol](#) (ACP). It is a [distributed algorithm](#) that coordinates all the processes that participate in a [distributed atomic transaction](#) on whether to [commit](#) or abort (roll back) the transaction. This protocol (a specialised type of [consensus](#) protocol) achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely used.[\[1\]\[2\]\[3\]](#) However, it is not resilient to all possible failure configurations, and in rare cases, manual intervention is needed to remedy an outcome. To accommodate recovery from failure (automatic in most cases) the protocol's participants use [logging](#) of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's [recovery procedures](#). Many protocol variants exist that primarily differ in logging strategies and recovery mechanisms. Though usually intended to be used infrequently, recovery procedures compose a substantial portion of the protocol, due to many possible failure scenarios to be considered and supported by the protocol.

In a "normal execution" of any single [distributed transaction](#) (i.e., when no failure occurs, which is typically the most frequent situation), the protocol consists of two phases:

1. The commit-request phase (or voting phase), in which a coordinator process attempts to prepare all the transaction's participating processes (named participants, cohorts, or workers) to take the necessary steps for either committing or aborting the transaction and to vote, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and
2. The commit phase, in which, based on voting of the participants, the coordinator decides whether to commit (only if all have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the participants. The participants then follow with the needed actions (commit or abort) with their local transactional resources (also called recoverable resources; e.g., database data) and their respective portions in the transaction's other output (if applicable).

The two-phase commit (2PC) protocol should not be confused with the [two-phase locking](#) (2PL) protocol, a [concurrency control](#) protocol.

Commit request (or voting) phase[Edit](#)

1. The coordinator sends a query to commit message to all participants and waits until it has received a reply from all participants.
2. The participants execute the transaction up to the point where they will be asked to commit. They each write an entry to their undo log and an entry to their [redo log](#).
3. Each participant replies with an agreement message (participant votes Yes to commit), if the participant's actions succeeded, or an abort message (participant votes No to commit), if the participant experiences a failure that will make it impossible to commit.

Commit (or completion) phase[Edit](#)

Success[Edit](#)

If the coordinator received an agreement message from all participants during the commit-request phase:

1. The coordinator sends a commit message to all the participants.
2. Each participant completes the operation, and releases all the locks and resources held during the transaction.
3. Each participant sends an acknowledgement to the coordinator.
4. The coordinator completes the transaction when all acknowledgements have been received.

Failure[Edit](#)

If any participant votes No during the commit-request phase (or the coordinator's timeout expires):

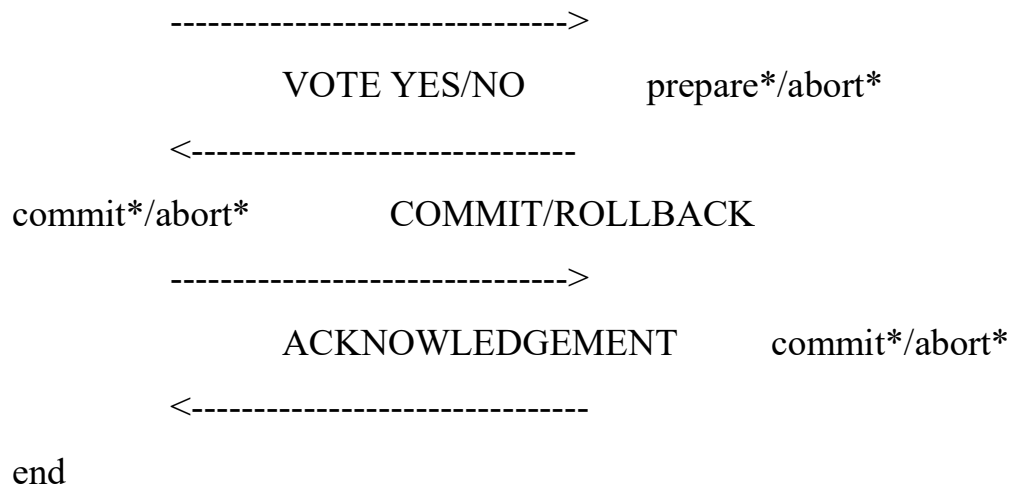
1. The coordinator sends a rollback message to all the participants.
2. Each participant undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
3. Each participant sends an acknowledgement to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received.

Message flow[Edit](#)

Coordinator

Participant

QUERY TO COMMIT



An * next to the record type means that the record is forced to stable storage.[\[4\]](#)

Disadvantages[Edit](#)

The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails permanently, some participants will never resolve their transactions: After a participant has sent an agreement message to the coordinator, it will block until a commit or rollback is received.

Protocol optimizations[Edit](#)

[Database](#) research has been done on ways to get most of the benefits of the two-phase commit protocol while reducing costs by protocol optimizations[\[1\]\[2\]\[3\]](#) and protocol operations saving under certain system's behavior assumptions.

Presumed abort and presumed commit[Edit](#)

Presumed abort or Presumed commit are common such optimizations.[\[2\]\[3\]\[5\]](#) An assumption about the outcome of transactions, either commit, or abort, can save both messages and logging operations by the participants during the 2PC protocol's

execution. For example, when presumed abort, if during system recovery from failure no logged evidence for commit of some transaction is found by the recovery procedure, then it assumes that the transaction has been aborted, and acts accordingly. This means that it does not matter if aborts are logged at all, and such logging can be saved under this assumption. Typically a penalty of additional operations is paid during recovery from failure, depending on optimization type. Thus the best variant of optimization, if any, is chosen according to failure and transaction outcome statistics.

Tree two-phase commit protocol [Edit](#)

The [Tree](#) 2PC protocol[2] (also called Nested 2PC, or Recursive 2PC) is a common variant of 2PC in a [computer network](#), which better utilizes the underlying communication infrastructure. The participants in a distributed transaction are typically invoked in an order which defines a tree structure, the invocation tree, where the participants are the nodes and the edges are the invocations (communication links). The same tree is commonly utilized to complete the transaction by a 2PC protocol, but also another communication tree can be utilized for this, in principle. In a tree 2PC the coordinator is considered the root ("top") of a communication tree (inverted tree), while the participants are the other nodes. The coordinator can be the node that originated the transaction (invoked recursively (transitively) the other participants), but also another node in the same tree can take the coordinator role instead. 2PC messages from the coordinator are propagated "down" the tree, while messages to the coordinator are "collected" by a participant from all the participants below it, before it sends the appropriate message "up" the tree (except an abort message, which is propagated "up" immediately upon receiving it or if the current participant initiates the abort).

The Dynamic two-phase commit (Dynamic two-phase commitment, D2PC) protocol[2][6] is a variant of Tree 2PC with no predetermined coordinator. It subsumes several optimizations that have been proposed earlier. Agreement messages (Yes votes) start to propagate from all the leaves, each leaf when completing its tasks on behalf of the transaction (becoming ready). An intermediate (non leaf) node sends ready when an agreement message to the last (single) neighboring node from which agreement message has not yet been received. The coordinator is determined dynamically by racing agreement messages over the transaction tree, at the place where they collide. They collide either at a transaction tree node, to be the coordinator, or on a tree edge. In the latter case one of the two edge's nodes is elected as a coordinator (any node). D2PC is time optimal (among all the instances of a specific transaction tree, and any specific Tree 2PC protocol implementation; all instances have the same tree; each instance has a different node as coordinator): By choosing an optimal coordinator D2PC commits both the coordinator and each participant in minimum possible time, allowing the earliest possible release of locked resources in each transaction participant (tree node).

8. Concurrency control

In [information technology](#) and [computer science](#), especially in the fields of [computer programming](#), [operating systems](#), [multiprocessors](#), and [databases](#), **concurrency control** ensures that correct results for [concurrent](#) operations are generated, while getting those results as quickly as possible.

Computer systems, both [software](#) and [hardware](#), consist of modules, or components. Each component is designed to operate correctly, i.e., to obey or to meet certain consistency rules. When components that operate concurrently interact by messaging or by sharing accessed data (in [memory](#) or [storage](#)), a certain component's consistency may be violated by another component. The general area of concurrency control provides rules, methods, design methodologies, and [theories](#) to maintain the consistency of components operating concurrently while interacting, and thus the consistency and correctness of the whole system. Introducing concurrency control into a system means applying operation constraints which typically result in some performance reduction. Operation consistency and correctness should be achieved with as good as possible efficiency, without reducing performance below reasonable levels. Concurrency control can require significant additional complexity and overhead in a [concurrent algorithm](#) compared to the simpler [sequential algorithm](#).

Database transaction and the ACID rules[Edit](#)

Main articles: [Database transaction](#) and [ACID](#)

The concept of a *database transaction* (or *atomic transaction*) has evolved in order to enable both a well understood database system behavior in a faulty environment where crashes can happen any time, and *recovery* from a crash to a well understood database state. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands). Every database

transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

- **Atomicity** - Either the effects of all or none of its operations remain ("all or nothing" semantics) when a transaction is completed (*committed* or *aborted* respectively). In other words, to the outside world a committed transaction appears (by its effects on the database) to be indivisible (atomic), and an aborted transaction does not affect the database at all. Either all the operations are done or none of them are.
- **Consistency** - Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects). A transaction must transform a database from one consistent state to another consistent state (however, it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct, i.e., performs correctly what it intends to perform (from the application's point of view) while the predefined integrity rules are enforced by the DBMS). Thus since a database can be normally changed only by transactions, all the database's states are consistent.
- **Isolation** - Transactions cannot interfere with each other (as an end result of their executions). Moreover, usually (depending on concurrency control method) the effects of an incomplete transaction are not even visible to another transaction. Providing isolation is the main goal of concurrency control.
- **Durability** - Effects of successful (committed) transactions must persist through crashes (typically by recording the transaction's effects and its commit event in a non-volatile memory).

The concept of atomic transaction has been extended during the years to what has become Business transactions which actually implement types of Workflow and

are not atomic. However also such enhanced transactions typically utilize atomic transactions as components.

Why is concurrency control needed? [Edit](#)

If transactions are executed *serially*, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable results may occur, such as:

1. The lost update problem: A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.
2. The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.
3. The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements. Thus, without concurrency control such

systems can neither provide correct results nor maintain their databases consistently.

Concurrency control mechanisms [Edit](#)

Categories [Edit](#)

The main categories of concurrency control mechanisms are:

- **[Optimistic](#)** - Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., [serializability](#) and [recoverability](#)) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy.
- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.
- **Semi-optimistic** - Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

Different categories provide different performance, i.e., different average transaction completion rates (*throughput*), depending on transaction types mix, computing level of parallelism, and other factors. If selection and knowledge about trade-offs are available, then category and method should be chosen to provide the highest performance.

The mutual blocking between two transactions (where each one blocks the other) or more results in a [deadlock](#), where the transactions involved are stalled and cannot reach completion. Most non-optimistic mechanisms (with blocking) are prone to deadlocks which are resolved by an intentional abort of a stalled transaction (which releases the other transactions in that deadlock), and its immediate restart and re-execution. The likelihood of a deadlock is typically low.

Blocking, deadlocks, and aborts all result in performance reduction, and hence the trade-offs between the categories.

Methods[Edit](#)

Many methods for concurrency control exist. Most of them can be implemented within either main category above. The major methods,[\[1\]](#) which have each many variants, and in some cases may overlap or be combined, are:

1. Locking (e.g., [Two-phase locking](#) - 2PL) - Controlling access to data by [locks](#) assigned to the data. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.
2. **[Serialization graph checking](#)** (also called Serializability, or Conflict, or Precedence graph checking) - Checking for [cycles](#) in the schedule's [graph](#) and breaking them by aborts.
3. **[Timestamp ordering](#)** (TO) - Assigning timestamps to transactions, and controlling or checking access to data by timestamp order.
4. **[Commitment ordering](#)** (or Commit ordering; CO) - Controlling or checking transactions' chronological order of commit events to be compatible with their respective [precedence order](#).

Other major concurrency control types that are utilized in conjunction with the methods above include:

- **Multiversion concurrency control** (MVCC) - Increasing concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object) depending on scheduling method.
- **Index concurrency control** - Synchronizing access operations to indexes, rather than to user data. Specialized methods provide substantial performance gains.
- **Private workspace model (Deferred update)** - Each transaction maintains a private workspace for its accessed data, and its changed data become visible outside the transaction only upon its commit (e.g., Weikum and Vossen 2001). This model provides a different concurrency control behavior with benefits in many cases.

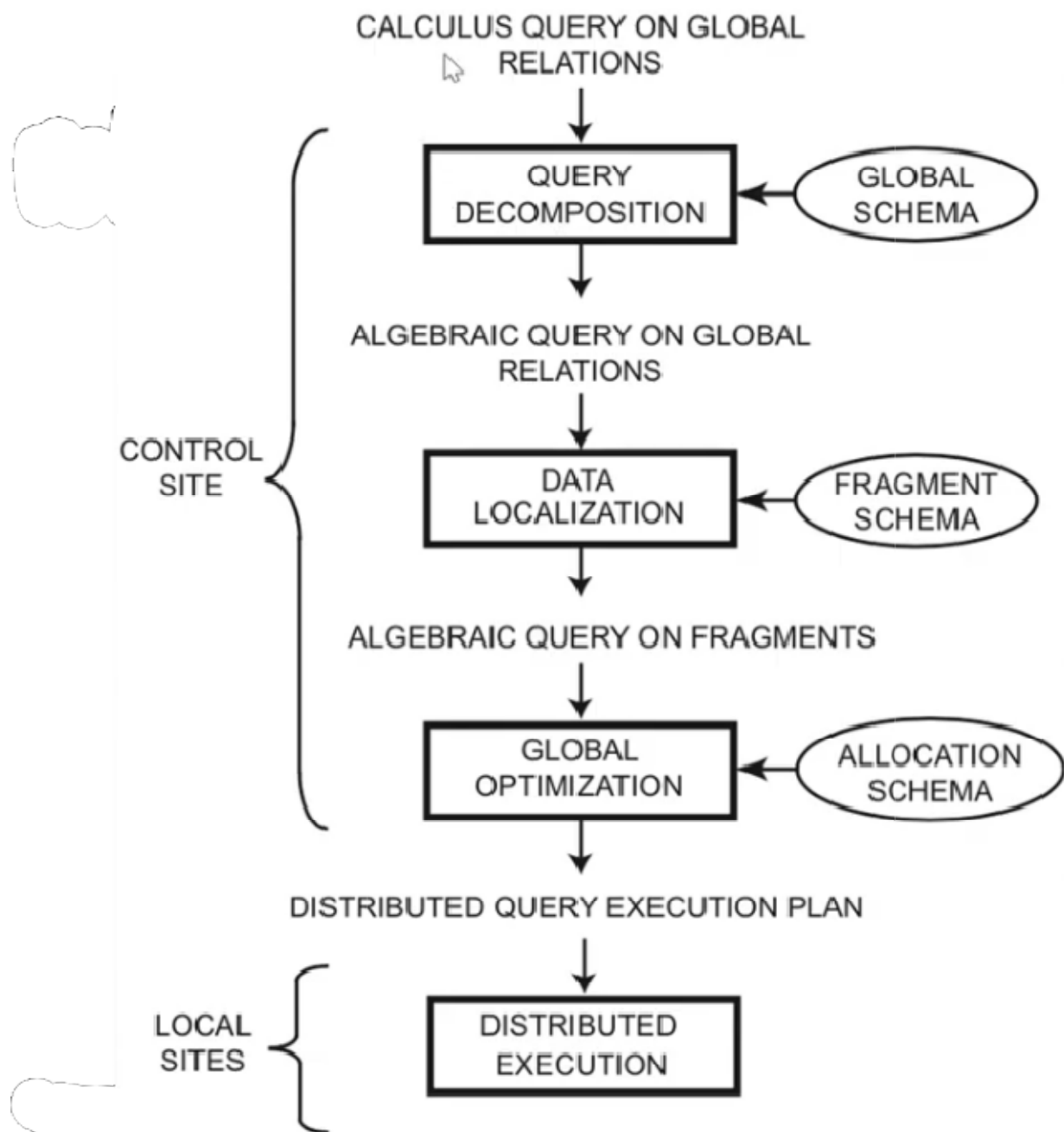
The most common mechanism type in database systems since their early days in the 1970s has been Strong strict Two-phase locking (SS2PL; also called *Rigorous scheduling* or *Rigorous 2PL*) which is a special case (variant) of both Two-phase locking (2PL) and Commitment ordering (CO). It is pessimistic. In spite of its long name (for historical reasons) the idea of the **SS2PL** mechanism is simple: "Release all locks applied by a transaction only after the transaction has ended." SS2PL (or Rigorousness) is also the name of the set of all schedules that can be generated by this mechanism, i.e., these SS2PL (or Rigorous) schedules have the SS2PL (or Rigorousness) property.

9.Explain the phases of query processing in distributed database.

Layers of Query Processing

Query processing has 4 layers:

- Query Decomposition
- Data Localization
- Global Query Optimization
- Distribution Query Execution



Query Decomposition

The first layer decomposes the calculus query into an algebraic query on global relations. The information needed for this transformation is found in the global conceptual schema describing the global relations.

- Query decomposition can be viewed as four successive steps.

- Normalization
 - Analysis
 - Simplification
 - Restructure
- First, the calculus query is rewritten in a normalized form that is suitable for subsequent manipulation. Normalization of a query generally involves the manipulation of the query quantifiers and of the query qualification by applying logical operator priority.
 - Second, the normalized query is analyzed semantically so that incorrect queries are detected and rejected as early as possible. Techniques to detect incorrect queries exist only for a subset of relational calculus. Typically, they use some sort of graph that captures the semantics of the query.
 - Third, the correct query (still expressed in relational calculus) is simplified. One way to simplify a query is to eliminate redundant predicates. Note that redundant queries are likely to arise when a query is the result of system transformations applied to the user query. such transformations are used for performing semantic data control (views, protection, and semantic integrity control).
 - Fourth, the calculus query is restructured as an algebraic query. The traditional way to do this transformation toward a "better" algebraic specification is to start with an initial algebraic query and transform it in order to find a "go
 - The algebraic query generated by this layer is good in the sense that the PR152 yorse executions are typically avoided.

Query Processing Example

Query:

```
select salary  
from instructor  
where salary < 75000;
```

Data Localization

- The input to the second layer is an algebraic query on global relations. The main role of the second layer is to localize the query's data using data distribution information in the fragment schema.
- This layer determines which fragments are involved in the query and transforms the distributed query into a query on fragments.
- A global relation can be reconstructed by applying the fragmentation rules, and then deriving a program, called a localization program, of relational algebra operators, which then act on fragments.

Generating a query on fragments is done in two steps

- First, the query is mapped into a fragment query by substituting each relation by its reconstruction program (also called materialization program).
- Second, the fragment query is simplified and restructured to produce another "good" query.

Global Query Optimization

- The input to the third layer is an algebraic query on fragments. The goal of query optimization is to find an execution strategy for the query which is close to optimal.
- The previous layers have already optimized the query, for example, by eliminating redundant expressions. However, this optimization is independent of fragment characteristics such as fragment allocation and cardinalities.
- Query optimization consists of finding the "best" ordering of operators in the query, including communication operators that minimize a cost function.
- The output of the query optimization layer is a optimized algebraic query with communication operators included on fragments. It is typically represented and saved (for future executions) as a distributed query execution plan.

Distribution Query Execution

- The last layer is performed by all the sites having fragments involved in the query.
- Each sub queryexecuting atone site, called a local query, is then optimized using the local schema of the site and executed.

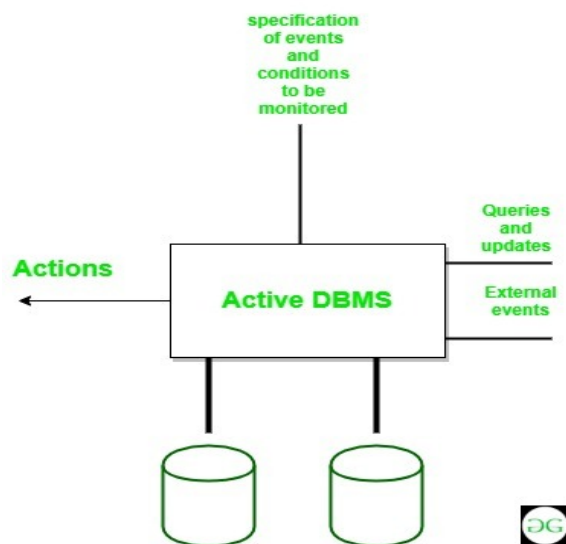
Unit2

UNIT II

SPATIAL AND TEMPORAL DATABASES

1.Active Databases

An active Database is a database consisting of a set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not, prior to executing the statement. If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement. In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'. The DBMS executes such chains of trigger in some pre-defined manner but it effects the concept of understanding.



Features of Active Database:

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.

2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution

Advantages :

1. Enhances traditional database functionalities with powerful rule processing capabilities.
2. Enable a uniform and centralized description of the business rules relevant to the information system.
3. Avoids redundancy of checking and repair operations.
4. Suitable platform for building large and efficient knowledge base and expert systems

2.Design and Implementation Issues for Active Databases

The first issue concerns activation, deactivation, and grouping of rules. In addition to creating rules, an active database system should allow users to activate, deactivate, and drop rules by referring to their rule names. A deactivated rule will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed. The activate command will make the rule active again. The drop command deletes the rule from the system. Another option is to group rules into named rule sets , so the whole set of rules can be activated, deactivated, or dropped. It is also useful to

have a command that can trigger a rule or rule set via an explicit PROCESS RULES command issued by the user.

The second issue concerns whether the triggered action should be executed before, after, instead of, or concurrently with the triggering event. A before trigger executes the trigger before executing the event that caused the trigger. It can be used in applications such as checking for constraint violations. An after trigger executes the trigger after executing the event, and it can be used in applications such as maintaining derived data and monitoring for specific events and conditions. An instead of trigger executes the trigger instead of executing the event, and it can be used in applications such as executing corresponding updates on base relations in response to an event that is an update of a view.

A related issue is whether the action being executed should be considered as a separate transaction or whether it should be part of the same transaction that triggered the rule. We will try to categorize the various options. It is important to note that not all options may be available for a particular active database system. In fact, most commercial systems are limited to one or two of the options that we will now discuss.

Let us assume that the triggering event occurs as part of a transaction execution. We should first consider the various options for how the triggering event is related to the evaluation of the rule's condition. The rule condition evaluation is also known as rule consideration, since the action is to be executed only after considering whether the condition evaluates to true or false. There are three main possibilities for rule consideration:

1. Immediate consideration. The condition is evaluated as part of the same transaction as the triggering event, and is evaluated immediately. This case

can be further categorized into three options:

Evaluate the condition before executing the triggering event.

Evaluate the condition after executing the triggering event.

Evaluate the condition instead of executing the triggering event

2. Deferred consideration. The condition is evaluated at the end of the transaction that included the triggering event. In this case, there could be many

3.Temporal Database

A temporal database is a database that needs some aspect of time for the organization of information. In the temporal database, each tuple in relation is associated with time. It stores information about the states of the real world and time. The temporal database does store information about past states it only stores information about current states. Whenever the state of the database changes, the information in the database gets updated. In many fields, it is very necessary to store information about past states. For example, a stock database must store information about past stock prizes for analysis. Historical information can be stored manually in the schema.

There are various terminologies in the temporal database:

- **Valid Time:** The valid time is a time in which the facts are true with respect to the real world.

- **Transaction Time:** The transaction time of the database is the time at which the fact is currently present in the database.
- **Decision Time:** Decision time in the temporal database is the time at which the decision is made about the fact.

Temporal databases use a relational database for support. But relational databases have some problems in temporal database, i.e. it does not provide support for complex operations. Query operations also provide poor support for performing temporal queries.

Applications of Temporal Databases

Finance: It is used to maintain the **stock price** histories.

1. It can be used in **Factory Monitoring System** for storing information about current and past readings of sensors in the factory.
2. **Healthcare:** The histories of the patient need to be maintained for giving the right treatment.
3. **Banking:** For maintaining the credit histories of the user.

Examples of Temporal Databases

1. An **EMPLOYEE table** consists of a **Department table** that the employee is assigned to. If an employee is **transferred to another department** at some point in time, this can be tracked if the EMPLOYEE table is an application time-period table that assigns the appropriate time periods to each department he/she works for.

Temporal Relation

A temporal relation is defined as a relation in which each tuple in a table of the database is associated with time, the time can be either transaction time or valid time.

Types of Temporal Relation

There are mainly three types of temporal relations:

1. **Uni-Temporal Relation:** The relation which is associated with valid or transaction time is called Uni-Temporal relation. It is related to only one time.
2. **Bi-Temporal Relation:** The relation which is associated with both valid time and transaction time is called a Bi-Temporal relation. Valid time has two parts namely start time and end time, similar in the case of transaction time.
3. **Tri-Temporal Relation:** The relation which is associated with three aspects of time namely Valid time, Transaction time, and Decision time called as Tri-Temporal relation.

Features of Temporal Databases

- The temporal database provides built-in support for the time dimension.
- Temporal database stores data related to the time aspects.
- A temporal database contains Historical data instead of current data.
- It provides a uniform way to deal with historical data.

Challenges of Temporal Databases

1. **Data Storage:** In temporal databases, each version of the data needs to be stored **separately**. As a result, storing the data in temporal databases requires more storage as compared to storing data in non-temporal databases.

2. **Schema Design:** The temporal database schema must accommodate the **time dimension**. Creating such a schema is more difficult than creating a schema for non-temporal databases.
3. **Query Processing:** Processing the query in temporal databases is **slower** than processing the query in non-temporal databases due to the additional complexity of managing temporal data

4. Temporal Query Languages

Definition

A temporal query language is a database query language that offers some form of built-in support for the querying and modification of time-referenced data, as well as enabling the specification of assertions and constraints on such data. A temporal query language is usually quite closely associated with a temporal data model that defines the underlying data structures to which the language applies.

Temporal Table Queries

Queries involving a temporal table with valid time can be current, sequenced, or nonsequenced. On a table with transaction time, temporal queries can be current or nonsequenced.

A current query is a SELECT statement that extracts and operates on the current rows of a table:

- For a transaction-time table, current queries operate only on open rows.
- For a valid-time table, current queries operate only on rows with valid-time periods that overlap the current time.
- For bitemporal tables, current queries operate only on rows that are both open in the transaction-time dimension and current in the valid-time dimension.

A current query produces a nontemporal table as a result set.

A sequenced query is a SELECT statement that extracts and operates on rows in a valid-time or bitemporal table with valid-time periods that overlap a time period specified in the query (the PA of the query). If no time period is explicitly specified in the query, the default PA is all time, and the query applies to all open rows in the table. Such queries can return rows that are history rows, current rows, future rows, or combinations of the three.

A sequenced query produces a temporal table as a result set. The valid time of the result rows is the overlap of the query PA with the original row PV.

An as of query is a SELECT statement that extracts and operates on rows in temporal tables with valid-time and transaction-time periods that overlap an AS OF date or time specified in the query. As of queries operate on the data as a snapshot at any point in time. Typically, as of queries are used for querying historical data.

The AS OF clause can be applied to the valid-time and transaction-time dimensions together or independently. When applied to the valid-time dimension, it retrieves rows where the PV overlaps the specified AS OF time. When applied to the transaction-time dimension, it retrieves rows with transaction-time periods that overlap the specified AS OF time.

An as of query is similar in semantics to a current query; an AS OF extracts the information based on the specified time and a current query extracts the information as of the current time or date of the query being executed. However, a current query operates on only open rows in the transaction-time dimension. An AS OF query can read rows as of a particular point in time in the transaction-time dimension regardless of whether the rows are closed or open.

An as of query produces a nontemporal table as a result set.

5.Spatial Data

The data set that is used to analyze the past as well as to work on analytics is known as Spatial Data. Spatial Data is limited to simple spreadsheet level information, but it also comprises imagery from Satellites and Drones, addresses data points, and longitudinal and latitudinal data.

Primarily Spatial Data is classified as Vector Data and Raster Data. Vector Data consists of Coordinates information, while Raster Data is all about layers of imageries extracted from camera sensors. Spatial data that belongs to geographical and geological information is known as geospatial data. Relational Database Management Systems handle these geospatial data, and they are called as GIS Databases

Types of Spatial Data

Spatial Data is mainly classified into two types, i.e. Vector data and Raster data.

1. Vector Data

Vector Data is the data portrayed in points, lines and It can be represented in two dimensional and two-dimensional models depending on the coordinates used. Vector Data in GIS is used to feed in information with the help of coordinates and to visualize the address points & places of interest, lines for the rivers, roads, railways, ferry routes and even major pipeline flows, polygons, on the other hand, are used to showcase inland water bodies like lakes, buildings, etc.

- **Points:** A single dot on the layer depicts them. It can be either x or y or z coordinate.
- **Lines:** This form of vector data is depicted using two coordinates, i.e. either x coordinate – y coordinate or inverse of this. This feature has a defined length, and also, it doesn't have the width because of the two-dimensional model.
- **Polygon:** The feature is depicted using 3 or more than 3 coordinates. This form of vector data is generally used when any area is defined

2. Raster Data

Raster Data is all about multilayered map images from satellites, drones and various other camera sensors. The data stored is in cell-based and colour pixel format. These are pixels that are arranged in columns and rows format. The data is in .jpg, .png, bit map, .tif and .bmp. The Raster Data in GIS is very much efficient for visualization and analysis that is barely possible in Vector Based data. Unlike Vector Data, the Raster form of GIS data is large and complex to manage due to richer qualities.

Use of Spatial Data in GIS

- The Spatial Data is collected from various camera sources, drones, satellite, sensors and geological field workers. Vector Data is mostly about address points, lines and polygons. Attributional values and georeferenced coding is done on all the features. All the attributes are as per organizational Standardized Operating Procedures, also known as SOPs. There are many geological concepts and logic involved while adding the attributional data in the features.
- GIS Technicians, GIS Analysts and GIS Developers work together in the process known as Geocoding. For example, features like address points, roads, rivers and even polygonal features like lakes are fed with all the attributional information like name, length, width and even some extra parameters if needed.
- The data is integrated into a conjunction with the longitudinal and latitudinal information depending on the placement. The vector form of data is always added after being referred to and validated with the specific Raster data.
- Any of the Spatial data is processed through GIS software where it can be analyzed, manipulated and visualized perfectly as per an individual's or organization's requirement. Based on Raster data that is available in the GIS system, GIS Technicians start the leftover or half-cooked map and complete all the geocoding by referring to the metrics available. While geocoding and referencing, Technicians also make a note of the parameters like longitudinal, latitudinal values and even the extra information like name, length and width.
- The integrated data is then saved in the RDBMS, and so that same can be used to understand the problem statement related to earth. 3D models nowadays are used to coordinate systems to portray business problems in a more granular way. For dealing with 3D Models, Organizations are also

involving VFX and graphics experts with higher-end knowledge in various animation tools.

Advantages of Using Spatial Data

Now let's look at some of the advantages:

- With timely updates on the data sets, the organisation can easily perform analysis and analytics.
- Updated information can be rolled out to the consumers promptly.
- The spatial databases store both vector and raster data, hence it can be used to tackle the maximum amount of business problems.
- The blend of both vector and raster data produces a powerful product that can tackle various economic and earth-related problems.
- Spatial data, when combined with non-spatial data like information on soil, the population of the city, can become a rich source of knowledge.
- With the help of available information, Decision making and strategic planning can be done thoroughly.
- The data is corrected and updated regularly, and hence the chance of analyzing erroneous data from the system is very low.
- Spatial data can be integrated with various other technologies like **LIDAR** to create various models.

Spatial Data Types and Models

Spatial data is the data collected through with physical real life locations like towns, cities, islands etc. Spatial data are basically of three different types and are wisely used in commercial sectors :

1. **Map data** : Map data includes different types of spatial features of objects in map, e.g – an object's shape and location of object within map. The three basic types of features are points, lines, and polygons (or areas).
 - **Points** – Points are used to represent spatial characteristics of objects whose locations correspond to single 2-D coordinates (x, y, or longitude/latitude) in the scale of particular application. For examples : Buildings, cellular towers, or stationary vehicles. Moving vehicles and other moving objects can be represented by sequence of point locations that change over time.
 - **Lines** – Lines represent objects having length, such as roads or rivers, whose spatial characteristics can be approximated by sequence of connected lines.
 - **Polygons** – Polygons are used to represent characteristics of objects that have boundary, like states, lakes, or countries.
2. **Attribute data** : It is the descriptive data that [Geographic Information Systems](#) associate with features in the map. For example, in map representing countries within an Indian state (ex – Odisha or Mumbai). Attributes- Population, largest city/town, area in square miles, and so on.
3. **Image data** : It includes camera created data like satellite images and aerial photographs. Objects of interest, such as buildings and roads, can be identified and overlaid on these images. Aerial and satellite images are typical examples of *raster data*.

Models of Spatial Information : It is divided into two categories :

- **Field** : These models are used to model spatial data that is continuous in nature, e.g. terrain elevation, air quality index, temperature data, and soil variation characteristics.

- **Object** : These models have been used for applications such as transportation networks, land parcels, buildings, and other objects that possess both spatial and non-spatial attributes. A spatial application is modeled using either **field** or an **object** based model, which depends on the requirements and the traditional choice of model for the application. Example – High traffic analysing system, etc.

6.Spatial Operators and Queries

1. Spatial operators :
Spatial operators these operators are applied in **geometric properties** of objects.

It is then used in the physical space to capture them and the relation among them. It is also used to perform spatial analysis.

Spatial operators are grouped into three categories :

- 1. Topological operators :**
 Topological properties do not vary when topological operations are applied, like translation or rotation.
 Topological operators are hierarchically structured in many levels. The base level offers operators, ability to check for detailed topological relations between regions with a broad boundary. The higher levels offer more abstract operators that allow users to query uncertain spatial data independent of the geometric data model.
- 1. Examples –**
 open (region), close (region), and inside (point, loop).

2. Projective operators :

Projective operators, like convex hull are used to establish predicates regarding the concavity convexity of objects.

Example —

Having inside the object's concavity.

3. Metric operators :

Metric operators's task is to provide a more accurate description of the geometry of the object. They are often used to measure the global properties of singular objects, and to measure the relative position of different objects, in terms of distance and direction.

Example —

length (of an arc) and distance (of a point to point).

2. Dynamic Spatial Operators :

Dynamic operations changes the objects upon which the operators are applied. Create, destroy, and update are the fundamental dynamic operations.

Example —

Updation of a spatial object via **translate, rotate, scale up or scale down, reflect, and shear.**

3. Spatial Queries :

Any type of spatial data that is data related to location and which represents objects defined in a geometric space, is stored and maintained by **Spatial Databases**. These are used to handle these Spatial Databases. Spatial database mainly contain representation of simple geometric objects such as 3D objects, topological coverage, linear networks and TINs(Triangulated irregular networks).

The requests for the spatial data which requires the use of spatial operations are called Spatial Queries.

It can be divided into –

1. **Range queries :**

It finds all objects of a particular type that are within a given spatial area.

Example –

Finds all hospitals within the Siliguri area. A variation of this query is for a given location, find all objects within a particular distance, for example, find all banks within 5 km range.

2. **Nearest neighbor queries :**

It Finds object of a particular type which is nearest to a given location.

Example –

Finds the nearest police station from the location of accident.

3. **Spatial joins or overlays :**

It joins the objects of two types based on spatial condition, such as the objects which are intersecting or overlapping spatially.

Example –

Finds all Dhabas on a National Highway between two cities. It spatially joins township objects and highway object.

Finds all hotels that are within 5 kilometres of a railway station. It spatially joins railway station objects and hotels objects.

A spatial database saves a huge amount of space-related data, including maps, preprocessed remote sensing or medical imaging records, and VLSI chip design data. Spatial databases have several features that distinguish them from relational databases. They carry topological and/or distance information, usually organized by sophisticated, multidimensional spatial indexing structures that are accessed by spatial data access methods and often require spatial reasoning, geometric computation, and spatial knowledge representation techniques.

Spatial data mining refers to the extraction of knowledge, spatial relationships, or other interesting patterns not explicitly stored in spatial databases. Such mining demands the unification of data mining with spatial database technologies. It can be used for learning spatial records, discovering spatial relationships and relationships among spatial and nonspatial records, constructing spatial knowledge bases, reorganizing spatial databases, and optimizing spatial queries.

It is expected to have broad applications in geographic data systems, marketing, remote sensing, image database exploration, medical imaging, navigation, traffic control, environmental studies, and many other areas where spatial data are used.

A central challenge to spatial data mining is the exploration of efficient spatial data mining techniques because of the large amount of spatial data and the difficulty of spatial data types and spatial access methods. Statistical spatial data analysis has been a popular approach to analyzing spatial data and exploring geographic information.

The term geostatistics is often associated with continuous geographic space, whereas the term spatial statistics is often associated with discrete space. In a

statistical model that manages non-spatial records, one generally considers statistical independence among different areas of data.

There is no such separation among spatially distributed records because, actually spatial objects are interrelated, or more exactly spatially co-located, in the sense that the closer the two objects are placed, the more likely they send the same properties. For example, natural resources, climate, temperature, and economic situations are likely to be similar in geographically closely located regions.

Such a property of close interdependency across nearby space leads to the notion of spatial autocorrelation. Based on this notion, spatial statistical modeling methods have been developed with success. Spatial data mining will create spatial statistical analysis methods and extend them for large amounts of spatial data, with more emphasis on effectiveness, scalability, cooperation with database and data warehouse systems, enhanced user interaction, and the discovery of new kinds of knowledge.

8.Spatial Indexing

A spatial index is a data structure that allows for accessing a spatial object efficiently. It is a common technique used by spatial databases. Without indexing, any search for a feature would require a "sequential scan" of every record in the database, resulting in much longer processing time. In a spatial index construction process, the minimum bounding rectangle serves as an object approximation. Various types of spatial indices across commercial and open-source databases yield measurable performance differences. Spatial

indexing techniques are playing a central role in time-critical applications and the manipulation of spatial big data.

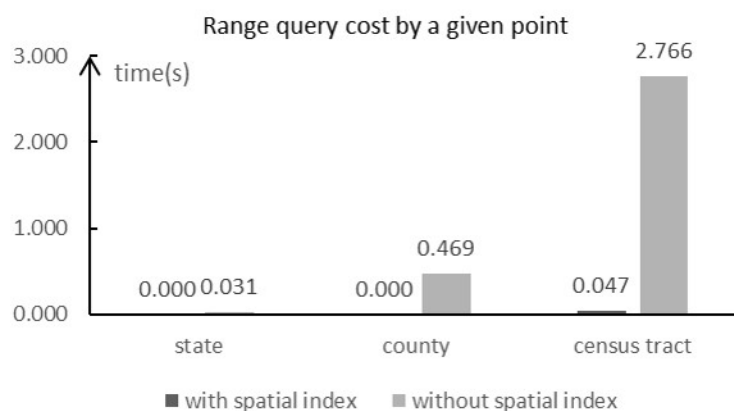
Spatial Index is a data structure that allows for accessing a spatial object efficiently. It is a common technique used by spatial databases. A variety of spatial operations needs the support from spatial index for efficient processing:

Range query: Finding objects containing a given point (point query) or overlapping with an area of interest (window query).

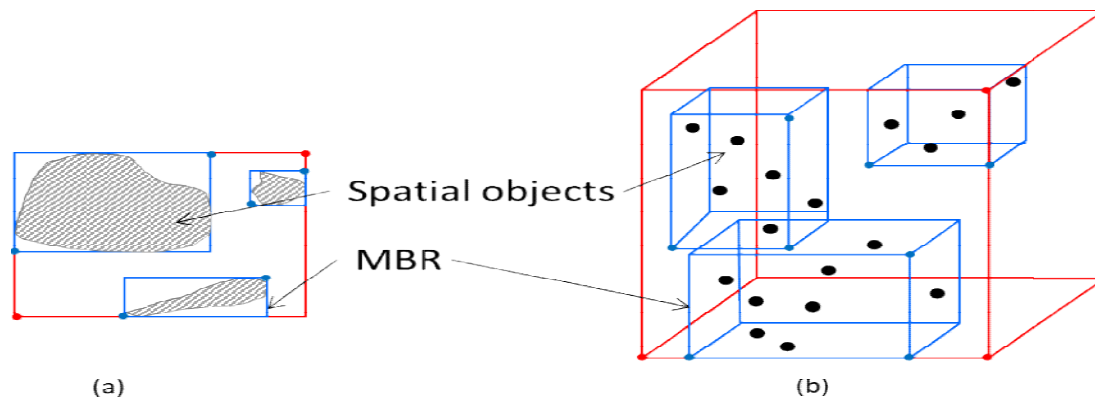
Spatial join: Finding pairs of objects that interact spatially with each other. Intersection, adjacency, and containment are common examples of spatial predicates to perform spatial joins.

K-Nearest Neighbor (KNN): Finding the nearest K spatial objects in a defined neighborhood of a target object.

Here is a simple test executing the SELECT statement with and without a spatial index, finding corresponding features from three different datasets by a given point. Both SELECT operations yield the same results, however, the execution time increases when the data volume gets larger. Without indexing, any search for a feature would require a "sequential scan" of every record in the database, resulting in a much longer processing time.



In order to reduce the cost of calculating the complex shape of spatial object during the search traversal, we use approximations of the complex object geometries. The most commonly used approximation is the minimum bounding rectangle, or MBR, which is also called minimum bounding box, or MBB. An MBR is a single rectangle that minimally encloses the geometry in 2D plane, as shown in Figure 2(a). It can also be extended to higher dimensions, such as a 3D MBR as in Figure 2(b).



Spatial Index in Different Databases

Different data sources use different data structures and access methods. Here we list two well-known spatial indices as well as databases which use them. The categorization system proposed by Riguax et al. (2002) is employed here for illustration:

Space-driven structures. These data structures are based on partitioning of the embedding 2D space into cells (or grids), mapping MBRs to the cells according to some spatial relationship (overlap or intersect). Commercial databases like IBM DB2 and Microsoft SQL Server use these methods.

Besides, Esri also implement its spatial data access method on geodatabases using space-driven structures.

Data-driven structures. These data structures are directly organized by partition of the collection of spatial objects. Data Objects are grouped using MBRs adapting to their distribution in the embedding space. Commercial databases like Oracle and Informix as well as some open-source databases, PostGIS and MySQL, use these data structures.

8.Mobile Database

A Mobile database is a database that can be connected to a mobile computing device over a mobile network (or wireless network). Here the client and the server have wireless connections. In today's world, mobile computing is growing very rapidly, and it is huge potential in the field of the database. It will be applicable on different-different devices like android based mobile databases, iOS based mobile databases, etc. Common examples of databases are Couch base Lite, Object Box, etc.

Features of Mobile database :

Here, we will discuss the features of the mobile database as follows.

- A cache is maintained to hold frequent and transactions so that they are not lost due to connection failure.

- As the use of laptops, mobile and PDAs is increasing to reside in the mobile system.
- Mobile databases are physically separate from the central database server.
- Mobile databases resided on mobile devices.
- Mobile databases are capable of communicating with a central database server or other mobile clients from remote sites.
- With the help of a mobile database, mobile users must be able to work without a wireless connection due to poor or even non-existent connections (disconnected).
- A mobile database is used to analyze and manipulate data on mobile devices.

Mobile Database typically involves three parties :

1. Fixed Hosts –

It performs the transactions and data management functions with the help of database servers.

2. Mobiles Units –

These are portable computers that move around a geographical region that includes the cellular network that these units use to communicate to base stations.

3. Base Stations –

These are two-way radios installation in fixed locations, that pass communication with the mobile units to and from the fixed hosts.

Limitations :

Here, we will discuss the limitation of mobile databases as follows.

- It has Limited wireless bandwidth.
- In the mobile database, Wireless communication speed.

- It required Unlimited battery power to access.
- It is Less secured.
- It is Hard to make theft-proof.

9.Mobile Databases: Location and Handoff Management

Location and Handoff Management The current point of attachment or location of a subscriber (mobile unit) is expressed in terms of the cell or the base station to which it is presently connected. The mobile units (called and calling subscribers) can continue to talk and move around in their respective cells; but as soon as both or any one of the units moves to a different cell, the location management procedure is invoked to identify the new location

Location and Handoff Management

The location management performs three fundamental tasks: –

- (a) location update,
- (b) location lookup
- (c) paging.

In location update, which is initiated by the mobile unit, the current location of the unit is recorded in HLR and VLR databases. Location lookup is basically a database search to obtain the current location of the mobile unit and through paging the system informs the caller the location of the called unit in terms of its current base station. These two tasks are initiated by the MSC..

Location and Handoff Management The cost of update and paging increases as cell size decreases, which becomes quite significant for finer granularity

cells such as micro- or picocell clusters. The presence of frequent cell crossing, which is a common scenario in highly commuting zones, further adds to the cost. The system creates location areas and paging areas to minimize the cost. A number of neighboring cells are grouped together to form a location area, and the paging area is constructed in a similar way.

Mobility Management Location Management –Search: find a mobile user's current location –Update (Register): update a mobile user's location –Location info: maintained at various granularities (cell vs. a group of cells called a registration area) –Research Issue: organization of location databases
Global Systems for Mobile (GSM) vs. Mobile IP vs. Wireless Mesh Networks (WMN) **Handoff Management** –Ensuring that a mobile user remains connected while moving from one location (e.g., cell) to another –Packets or connection are routed to the new location

Handoff Management Decide when to handoff to a new access point (AP) Select a new AP from among several APs Acquire resources such as bandwidth channels (GSM), or a new IP address (Mobile IP) –Channel allocation is a research issue: goal may be to maximize channel usage, satisfy QoS, or maximize revenue generated Inform the old AP to reroute packets and also to transfer state information to the new AP Packets are routed to the new AP

Tradeoff in Location Management Network may only know approximate location By location update (or location registration): –Network is informed of the location of a mobile user By location search or terminal paging: –Network is finding the location of a mobile user A tradeoff exists between location update and search –When the user is not called often (or if the service arrival

rate) is low, resources are wasted with frequent updates –If not done and a call comes, bandwidth or time is wasted in searching

Registration Area (RA) and the Basic HLR-VLR Scheme Current Personal Communication Service (PCS) networks (i.e., cellular networks such as GSM) use RA-based basic HLR-VLR schemes: –The service coverage area is divided into registration areas (RAs), each with a visitor location register (VLR) –Each RA covers a group of base stations (cells). –A user has a permanent home location register (HLR) –Base stations within the same RA broadcast their IDs –If ID is sensed different by the mobile terminal then a terminal, location update is sent to the VLR of the current RA. –When crossing a RA boundary, an update is sent to the HLR. –A search goes by HLR->VLR->cell->paging (by the base

Registration Areas in a PCN The figure on the right shows a PCS network – RA topology –RA graph model

PSTN: Public Switched Telephone Network
HLR: Home Location Register
STP: Service Transfer Point
VLR: Visitor Location Register

forwarding Pointers (Ref [1]) Update (upon crossing a RA boundary) –When the length of forwarding pointers $< K$: Set up a pointer between the two involving VLRs –When the length of forwarding pointers $= K$: Update information to the HLR Search – HLR \rightarrow VLR₀ \rightarrow \rightarrow VLR_i \rightarrow cell \rightarrow paging

10.Deductive Databases

A **Deductive Database** is a type of database that can make conclusions or we can say deductions using a sets of well defined rules and fact that are stored in the database. In today's world as we deal with a large amount of data, this deductive database provides a lot of advantages. It helps to combine the *RDBMS* with logic programming. To design a deductive database a purely declarative programming language called Datalog is used.

The implementations of deductive databases can be seen in LDL (Logic Data Language), NAIL (Not Another Implementation of Logic), CORAL, and VALIDITY.

The use of LDL and VALIDITY in a variety of business/industrial applications are as follows.

1. **LDL** **Applications:**

This system has been applied to the following application domains:

- **Enterprise modeling:**

Data related to an enterprise may result in an extended ER model containing hundreds of entities and relationship and thousands of attributes. This domain involves modeling the structure, processes, and constraints within an enterprise.

- **Hypothesis testing or data dredging:**

This domain involves formulating a hypothesis, translating in into an LDL rule set and a query, and then executing the query against given data to test the hypothesis. This has been applied to genome data analysis in the field of microbiology, where data dredging consists of identifying the DNA sequences from low-level digitized auto radio graphs from experiments performed on E.Coli Bacteria.

- **Software** **reuse:**

A small fraction of the software for an application is rule-based and encoded in LDL (bulk is developed in standard procedural code). The rules give rise to a knowledge base that contains, *A definition of each C module used in system and A set of rules that defines ways in which modules can export/import functions, constraints and so on.* The “Knowledge base” can be used to make decisions that pertain to the reuse of software subsets. This is being experimented within banking software.

2. **VALIDITY** **Applications:**

Validity combines deductive capabilities with the ability to manipulate complex objects (OIDs, inheritance, methods, etc). It provides a DOOD data model and language called DEL (Datalog Extended Language), an engine working along a client-server model and a set of tools for schema and rule editing, validation, and querying.

The following are some application areas of the VALIDITY system:

- **Electronic** **commerce:**

In electronic commerce, complex customers profiles have to be matched against target descriptions. The matching process is also described by rules, and computed predicates deal with numeric computations. The declarative nature of DEL makes the formulation of the matching algorithm easy.

- **Rules-governed** **processes:**

In a rules-governed process, well defined rules define the actions to be performed. In those process some classes are modeled as DEL classes. The main advantage of VALIDITY is the ease with which new regulations are taken into account.

- **Knowledge** **discovery:**

The goal of knowledge discovery is to find new data relationships by

analyzing existing data. An application prototype developed by University of Illinois utilizes already existing minority student data that has been enhanced with rules in DEL.

- **Concurrent**

Engineering:

A concurrent engineering applications deals with large amounts of centralized data, shared by several participants. An application prototype has been developed in the area of civil engineering. The design data is modeled using the object-oriented power of the DEL language. DEL is able to handle transformation of rules into constraints, and it can also handle any closed formula as an integrity constraint.

11.Multimedia Database

Multimedia database is the collection of interrelated multimedia data that includes text, graphics (sketches, drawings), images, animations, video, audio etc and have vast amounts of multisource multimedia data. The framework that manages different types of multimedia data which can be stored, delivered and utilized in different ways is known as multimedia database management system. There are three classes of the multimedia database which includes static media, dynamic media and dimensional media.

Content of Multimedia Database management system :

1. **Media data** – The actual data representing an object.
2. **Media format data** – Information such as sampling rate, resolution, encoding scheme etc. about the format of the media data after it goes through the acquisition, processing and encoding phase.

3. **Media keyword data** – Keywords description relating to the generation of data. It is also known as content descriptive data. Example: date, time and place of recording.
4. **Media feature data** – Content dependent data such as the distribution of colors, kinds of texture and different shapes present in data.

Types of multimedia applications based on data management characteristic are :

1. **Repository applications** – A Large amount of multimedia data as well as meta-data(Media format date, Media keyword data, Media feature data) that is stored for retrieval purpose, e.g., Repository of satellite images, engineering drawings, radiology scanned pictures.
2. **Presentation applications** – They involve delivery of multimedia data subject to temporal constraint. Optimal viewing or listening requires DBMS to deliver data at certain rate offering the quality of service above a certain threshold. Here data is processed as it is delivered. Example: Annotating of video and audio data, real-time editing analysis.
3. **Collaborative work using multimedia information** – It involves executing a complex task by merging drawings, changing notifications. Example: Intelligent healthcare network.

There are still many challenges to multimedia databases, some of which are :

1. **Modelling** – Working in this area can improve database versus information retrieval techniques thus, documents constitute a specialized area and deserve special consideration.
2. **Design** – The conceptual, logical and physical design of multimedia databases has not yet been addressed fully as performance and tuning issues at each level are far more complex as they consist of a variety of formats like

JPEG, GIF, PNG, MPEG which is not easy to convert from one form to another.

3. **Storage** – Storage of multimedia database on any standard disk presents the problem of representation, compression, mapping to device hierarchies, archiving and buffering during input-output operation. In DBMS, a "BLOB"(Binary Large Object) facility allows untyped bitmaps to be stored and retrieved.
4. **Performance** – For an application involving video playback or audio-video synchronization, physical limitations dominate. The use of parallel processing may alleviate some problems but such techniques are not yet fully developed. Apart from this multimedia database consume a lot of processing time as well as bandwidth.
5. **Queries and retrieval** –For multimedia data like images, video, audio accessing data through query opens up many issues like efficient query formulation, query execution and optimization which need to be worked upon.

Areas where multimedia database is applied are :

- **Documents and record management** : Industries and businesses that keep detailed records and variety of documents. Example: Insurance claim record.
- **Knowledge dissemination** : Multimedia database is a very effective tool for knowledge dissemination in terms of providing several resources. Example: Electronic books.
- **Education and training** : Computer-aided learning materials can be designed using multimedia sources which are nowadays very popular sources of learning. Example: Digital libraries.
- Marketing, advertising, retailing, entertainment and travel. Example: a virtual tour of cities.

- **Real-time control and monitoring** : Coupled with active database technology, multimedia presentation of information can be very effective means for monitoring and controlling complex tasks Example: Manufacturing operation control

UNIT IV XML DATABASES

XML - Databases

XML Database is used to store huge amount of information in the XML format. As the use of XML is increasing in every field, it is required to have a secured place to store the XML documents. The data stored in the database can be queried using **XQuery**, serialized, and exported into a desired format.

XML Database Types

There are two major types of XML databases –

- XML- enabled
- Native XML (NXD)

XML - Enabled Database

XML enabled database is nothing but the extension provided for the conversion of XML document. This is a relational database, where data is stored in tables consisting of rows and columns. The tables contain set of records, which in turn consist of fields.

Native XML Database

Native XML database is based on the container rather than table format. It can store large amount of XML document and data. Native XML database is queried by the **XPath**-expressions.

Native XML database has an advantage over the XML-enabled database. It is highly capable to store, query and maintain the XML document than XML-enabled database.

Example

Following example demonstrates XML database –

```
<?xml version = "1.0"?>
<contact-info>
  <contact1>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
  </contact1>

  <contact2>
    <name>Manisha Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 789-4567</phone>
  </contact2>
</contact-info>
```

Here, a table of contacts is created that holds the records of contacts (contact1 and contact2), which in turn consists of three entities – *name*, *company* and *phone*.

1.Structured data

Big Data includes huge volume, high velocity, and extensible variety of data. There are 3 types: Structured data, Semi-structured data, and Unstructured data.

Structured data refers to data that is organized and formatted in a specific way to make it easily readable and understandable by both humans and machines. This is typically achieved through the use of a well-defined schema or data model, which provides a structure for the data.

Structured data is typically found in databases and spreadsheets, and is characterized by its organized nature. Each data element is typically assigned a specific field or column in the schema, and each record or row represents a specific instance of that data. For example, in a customer database, each record might contain fields for the customer's name, address, phone number, and email address.

Structured data is highly valuable because it can be easily searched, queried, and analyzed using various tools and techniques. This makes it an ideal format for data-driven applications such as business intelligence and analytics, as well as for machine learning and artificial intelligence applications.

Examples of structured data formats include relational databases, XML, and JSON. In contrast, unstructured data, such as text documents or images, do not have a predefined schema or structure, and can be more difficult to analyze and interpret.

Structured data is the data which conforms to a data model, has a well define structure, follows a consistent order and can be easily accessed and used by a

person or a computer program. Structured data is usually stored in well-defined schemas such as Databases. It is generally tabular with column and rows that clearly define its attributes. SQL (Structured Query language) is often used to manage structured data stored in databases.

Characteristics of Structured Data:

- Data conforms to a data model and has easily identifiable structure
- Data is stored in the form of rows and columns

Example : Database

- Data is well organised so, Definition, Format and Meaning of data is explicitly known
- Data resides in fixed fields within a record or file
- Similar entities are grouped together to form relations or classes
- Entities in the same group have same attributes
- Easy to access and query, So data can be easily used by other programs
- Data elements are addressable, so efficient to analyse and process

Sources of Structured Data:

- SQL Databases
- Spreadsheets such as Excel
- OLTP Systems
- Online forms
- Sensors such as GPS or RFID tags
- Network and Web server logs
- Medical devices

Advantages of Structured Data:

- Structured data have a well defined structure that helps in easy storage and access of data

- Data can be indexed based on text string as well as attributes. This makes search operation hassle-free
- Data mining is easy i.e knowledge can be easily extracted from data
- Operations such as Updating and deleting is easy due to well structured form of data
- Business Intelligence operations such as Data warehousing can be easily undertaken
- Easily scalable in case there is an increment of data
- Ensuring security to data is easy

Advantages:

- Easy to understand and use: Structured data has a well-defined schema or data model, making it easy to understand and use. This allows for easy data retrieval, analysis, and reporting.
- Consistency: The well-defined structure of structured data ensures consistency and accuracy in the data, making it easier to compare and analyze data across different sources.
- Efficient storage and retrieval: Structured data is typically stored in relational databases, which are designed to efficiently store and retrieve large amounts of data. This makes it easy to access and process data quickly.
- Enhanced data security: Structured data can be more easily secured than unstructured or semi-structured data, as access to the data can be controlled through database security protocols.
- Clear data lineage: Structured data typically has a clear lineage or history, making it easy to track changes and ensure data quality.

Disadvantages:

1. Inflexibility: Structured data can be inflexible in terms of accommodating new types of data, as any changes to the schema or data model require significant changes to the database.
2. Limited complexity: Structured data is often limited in terms of the complexity of relationships between data entities. This can make it difficult to model complex real-world scenarios.
3. Limited context: Structured data often lacks the additional context and information that unstructured or semi-structured data can provide, making it more difficult to understand the meaning and significance of the data.
4. Expensive: Structured data requires the use of relational databases and related technologies, which can be expensive to implement and maintain.
5. Data quality: The structured nature of the data can sometimes lead to missing or incomplete data, or data that does not fit cleanly into the defined schema, leading to data quality issues.

Overall, structured data offers many advantages in terms of ease of use, consistency, and security, but also presents some limitations in terms of flexibility and complexity that need to be carefully considered when designing and implementing data management systems.

2.Semi-structured data

Semi-structured data is a type of data that is not purely structured, but also not completely unstructured. It contains some level of organization or structure, but does not conform to a rigid schema or data model, and may contain elements that are not easily categorized or classified.

1. Semi-structured data is typically characterized by the use of metadata or tags that provide additional information about the data elements. For example, an XML document might contain tags that indicate the structure of the

document, but may also contain additional tags that provide metadata about the content, such as author, date, or keywords.

2. Other examples of semi-structured data include JSON, which is commonly used for exchanging data between web applications, and log files, which often contain a mix of structured and unstructured data.

Semi-structured data is becoming increasingly common as organizations collect and process more data from a variety of sources, including social media, IoT devices, and other unstructured sources. While semi-structured data can be more challenging to work with than strictly structured data, it offers greater flexibility and adaptability, making it a valuable tool for data analysis and management.

Semi-structured data is data that does not conform to a data model but has some structure. It lacks a fixed or rigid schema. It is the data that does not reside in a relational database but that have some organizational properties that make it easier to analyze. With some processes, we can store them in the relational database.

Characteristics of semi-structured Data:

- Data does not conform to a data model but has some structure.
- Data can not be stored in the form of rows and columns as in Databases
- Semi-structured data contains tags and elements (Metadata) which is used to group data and describe how the data is stored
- Similar entities are grouped together and organized in a hierarchy
- Entities in the same group may or may not have the same attributes or properties
- Does not contain sufficient metadata which makes automation and management of data difficult
- Size and type of the same attributes in a group may differ

- Due to lack of a well-defined structure, it can not used by computer programs easily

Sources of semi-structured Data:

- E-mails
- XML and other markup languages
- Binary executables
- TCP/IP packets
- Zipped files
- Integration of data from different sources
- Web pages

Advantages of Semi-structured Data:

- The data is not constrained by a fixed schema
- Flexible i.e Schema can be easily changed.
- Data is portable
- It is possible to view structured data as semi-structured data
- Its supports users who can not express their need in SQL
- It can deal easily with the heterogeneity of sources.
- Flexibility: Semi-structured data provides more flexibility in terms of data storage and management, as it can accommodate data that does not fit into a strict, predefined schema. This makes it easier to incorporate new types of data into an existing database or data processing pipeline.
- Scalability: Semi-structured data is particularly well-suited for managing large volumes of data, as it can be stored and processed using distributed computing systems, such as Hadoop or Spark, which can scale to handle massive amounts of data.

- **Faster data processing:** Semi-structured data can be processed more quickly than traditional structured data, as it can be indexed and queried in a more flexible way. This makes it easier to retrieve specific subsets of data for analysis and reporting.
- **Improved data integration:** Semi-structured data can be more easily integrated with other types of data, such as unstructured data, making it easier to combine and analyze data from multiple sources.
- **Richer data analysis:** Semi-structured data often contains more contextual information than traditional structured data, such as metadata or tags. This can provide additional insights and context that can improve the accuracy and relevance of data analysis.

Overall, semi-structured data provides a number of advantages over traditional structured data, particularly when it comes to managing and analyzing large volumes of data that do not fit neatly into predefined data models.

Disadvantages of Semi-structured data

- Lack of fixed, rigid schema make it difficult in storage of the data
- Interpreting the relationship between data is difficult as there is no separation of the schema and the data.
- Queries are less efficient as compared to [structured data](#).
- **Complexity:** Semi-structured data can be more complex to manage and process than structured data, as it may contain a wide variety of formats, tags, and metadata. This can make it more difficult to develop and maintain data models and processing pipelines.
- **Lack of standardization:** Semi-structured data often lacks the standardization and consistency of structured data, which can make it more difficult to ensure

data quality and accuracy. This can also make it harder to compare and analyze data across different sources.

- **Reduced performance:** Processing semi-structured data can be more resource-intensive than processing structured data, as it often requires more complex parsing and indexing operations. This can lead to reduced performance and longer processing times.
- **Limited tooling:** While there are many tools and technologies available for working with structured data, there are fewer options for working with semi-structured data. This can make it more challenging to find the right tools and technologies for a particular use case.
- **Data security:** Semi-structured data can be more difficult to secure than structured data, as it may contain sensitive information in unstructured or less-visible parts of the data. This can make it more challenging to identify and protect sensitive information from unauthorized access.
- Overall, while semi-structured data offers many advantages in terms of flexibility and scalability, it also presents some challenges and limitations that need to be carefully considered when designing and implementing data processing and analysis pipelines.

Extracting information from semi-structured Data

Semi-structured data have different structure because of heterogeneity of the sources. Sometimes they do not contain any structure at all. This makes it difficult to tag and index. So while extract information from them is tough job. Here are possible solutions –

- Graph based models (e.g OEM) can be used to index semi-structured data
- Data modelling technique in OEM allows the data to be stored in graph based model. The data in graph based model is easier to search and index.

- XML allows data to be arranged in hierarchical order which enables the data to be indexed and searched
- Use of various data mining tools

Problems faced in storing semi-structured data

- Data usually has an irregular and partial structure. Some sources have implicit structure of data, which makes it difficult to interpret the relationship between data.
- Schema and data are usually tightly coupled i.e they are not only linked together but are also dependent of each other. Same query may update both schema and data with the schema being updated frequently.

Possible solution for storing semi-structured data

- Data can be stored in DBMS specially designed to store semi-structured data
- XML is widely used to store and exchange semi-structured data. It allows its user to define tags and attributes to store the data in hierarchical form. Schema and Data are not tightly coupled in XML.
- Object Exchange Model (OEM) can be used to store and exchange semi-structured data. OEM structures data in form of graph.
- RDBMS can be used to store the data by mapping the data to relational schema and then mapping it to a table

3.Unstructured Data

Unstructured data is the data which does not conform to a data model and has no easily identifiable structure such that it can not be used by a computer program easily. Unstructured data is not organised in a pre-defined manner or does not have a pre-defined data model, thus it is not a good fit for a mainstream relational database.

Characteristics of Unstructured Data:

- Data neither conforms to a data model nor has any structure.
- Data can not be stored in the form of rows and columns as in Databases
- Data does not follows any semantic or rules
- Data lacks any particular format or sequence
- Data has no easily identifiable structure
- Due to lack of identifiable structure, it can not used by computer programs easily

Sources of Unstructured Data:

- Web pages
- Images (JPEG, GIF, PNG, etc.)
- Videos
- Memos
- Reports
- Word documents and PowerPoint presentations
- Surveys

Advantages of Unstructured Data:

- Its supports the data which lacks a proper format or sequence
- The data is not constrained by a fixed schema
- Very Flexible due to absence of schema.
- Data is portable
- It is very scalable
- It can deal easily with the heterogeneity of sources.
- These type of data have a variety of business intelligence and analytics applications.

Disadvantages Of Unstructured data:

- It is difficult to store and manage unstructured data due to lack of schema and structure

- Indexing the data is difficult and error prone due to unclear structure and not having pre-defined attributes. Due to which search results are not very accurate.
- Ensuring security to data is difficult task.

Problems faced in storing unstructured data:

- It requires a lot of storage space to store unstructured data.
- It is difficult to store videos, images, audios, etc.
- Due to unclear structure, operations like update, delete and search is very difficult.
- Storage cost is high as compared to [structured data](#)
- Indexing the unstructured data is difficult

Possible solution for storing Unstructured data:

- Unstructured data can be converted to easily manageable formats
- using Content addressable storage system (CAS) to store unstructured data. It stores data based on their metadata and a unique name is assigned to every object stored in it. The object is retrieved based on content not its location.
- Unstructured data can be stored in XML format.
- Unstructured data can be stored in RDBMS which supports BLOBs

Extracting information from unstructured Data:

unstructured data do not have any structure. So it can not easily interpreted by conventional algorithms. It is also difficult to tag and index unstructured data. So extracting information from them is tough job. Here are possible solutions:

- Taxonomies or classification of data helps in organising data in hierarchical structure. Which will make search process easy.
- Data can be stored in virtual repository and be automatically tagged. For example Documentum.

- Use of application platforms like XOLAP. XOLAP helps in extracting information from e-mails and XML based documents
- Use of various data mining tools

Differences between Structured, Semi-structured and Unstructured data:

Properties	Structured data	Semi-structured data	Unstructured data
Technology	It is based on Relational database table	It is based on XML/RDF(Resource Description Framework).	It is based on character and binary data
Transaction management	Matured transaction and various concurrency techniques	Transaction is adapted from DBMS not matured	No transaction management and no concurrency
Version management	Versioning over tuples,row,tables	Versioning over tuples or graph is possible	Versioned as a whole
Flexibility	It is schema dependent and less flexible	It is more flexible than structured data but less flexible than	It is more flexible and there is

Properties	Structured data	Semi-structured data	Unstructured data
		unstructured data	absence of schema
Scalability	It is very difficult to scale DB schema	It's scaling is simpler than structured data	It is more scalable.
Robustness	Very robust	New technology, not very spread	—
Query performance	Structured query allow complex joining	Queries over anonymous nodes are possible	Only textual queries are

4.XML Hierarchical (Tree) Data Model

We now introduce the data model used in XML. The basic object in XML is the XML document. Two main structuring concepts are used to construct an XML document: **elements** and **attributes**. It is important to note that the term *attribute* in XML is not used in the same manner as is customary in database terminology, but rather as it is used in document description languages such as HTML and SGML. Attributes in XML provide additional information that describes elements, as we will see. There are additional concepts in XML, such as

entities, identifiers, and references, but first we concentrate on describing elements and attributes to show the essence of the XML model.

Figure 12.3 shows an example of an XML element called <Projects>. As in HTML, elements are identified in a document by their start tag and end tag. The tag names are enclosed between angled brackets < ... >, and end tags are further identified by a slash, </ ... >.

```
<?xml version= "1.0" standalone="yes"?> <Projects>
```

```
<Project>
```

```
<Name>ProductX</Name>
```

```
<Number>1</Number>
```

```
<Location>Bellaire</Location> <Dept_no>5</Dept_no> <Worker>
```

```
<Ssn>123456789</Ssn>
```

```
<Last_name>Smith</Last_name>
```

```
<Hours>32.5</Hours>
```

```
</Worker>
```

```
<Worker>
```

```
<Ssn>453453453</Ssn>
```

```
<First_name>Joyce</First_name>
```

```
<Hours>20.0</Hours>
```

```
</Worker>
```

```
</Project>
```

```
<Project>
```

```
<Name>ProductY</Name>
```

```
<Number>2</Number>
```

```
<Location>Sugarland</Location> <Dept_no>5</Dept_no> <Worker>
<Ssn>123456789</Ssn>
<Hours>7.5</Hours>
</Worker>
<Worker>
<Ssn>453453453</Ssn>
<Hours>20.0</Hours>
</Worker>
<Worker>
<Ssn>333445555</Ssn>
<Hours>10.0</Hours>
</Worker>
</Project>
...
</Projects>
```

Figure 12.3 A complex XML element called <Projects>

Complex elements are constructed from other elements hierarchically, whereas **simple elements** contain data values. A major difference between XML and HTML is that XML tag names are defined to describe the meaning of the data elements in the document, rather than to describe how the text is to be displayed. This makes it possible to process the data elements in the XML document

automatically by computer programs. Also, the XML tag (element) names can be defined in another document, known as the *schema document*, to give a semantic meaning to the tag names that can be exchanged among multiple users. In HTML, all tag names are predefined and fixed; that is why they are not extendible.

It is straightforward to see the correspondence between the XML textual representation shown in Figure 12.3 and the tree structure shown in Figure 12.1. In the tree representation, internal nodes represent complex elements, whereas leaf nodes represent simple elements. That is why the XML model is called a **tree model** or a **hierarchical model**. In Figure 12.3, the simple elements are the ones with the tag names <Name>, <Number>, <Location>, <Dept_no>, <Ssn>, <Last_name>, <First_name>, and <Hours>. The complex elements are the ones with the tag names <Projects>, <Project>, and <Worker>. In general, there is no limit on the levels of nesting of elements.

It is possible to characterize three main types of XML documents:

Data-centric XML documents. These documents have many small data items that follow a specific structure and hence may be extracted from a structured database. They are formatted as XML documents in order to exchange them over or display them on the Web. These usually follow a *predefined schema* that defines the tag names.

Document-centric XML documents. These are documents with large amounts of text, such as news articles or books. There are few or no structured data elements in these documents.

Hybrid XML documents. These documents may have parts that contain structured data and other parts that are predominantly textual or unstructured. They may or may not have a predefined schema.

XML documents that do not follow a predefined schema of element names and corresponding tree structure are known as **schemaless XML documents**. It is important to note that datacentric XML documents can be considered either as semistructured data or as structured data as defined in Section 12.1. If an XML document conforms to a predefined XML schema or DTD (see Section 12.3), then the document can be considered as *structured data*. On the other hand, XML allows documents that do not conform to any schema; these would be considered as *semistructured data* and are *schemaless XML documents*. When the value of the standalone attribute in an XML document is yes, as in the first line in Figure 12.3, the document is standalone and schemaless.

XML attributes are generally used in a manner similar to how they are used in HTML (see Figure 12.2), namely, to describe properties and characteristics of the elements (tags) within which they appear. It is also possible to use XML attributes to hold the values of simple data elements; however, this is generally not recommended. An exception to this rule is in cases that need to **reference** another element in another part of the XML document. To do this, it is common to use attribute values in one element as the references. This resembles the concept of

foreign keys in relational databases, and is a way to get around the strict hierarchical model that the XML tree model implies. We discuss XML attributes further in Section 12.3 when we discuss XML schema and DTD.

5.XML - Documents

n 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](#)

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.

6.XML DTD

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.

A well-formed and valid XML document is one which have been validated against DTD.

valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

1. `<?xml version="1.0"?>`
2. `<!DOCTYPE employee SYSTEM "employee.dtd">`
3. `<employee>`
4. `<firstname>vimal</firstname>`
5. `<lastname>jaiswal</lastname>`
6. `<email>vimal@javatpoint.com</email>`
7. `</employee>`

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

1. `<!ELEMENT employee (firstname,lastname,email)>`
2. `<!ELEMENT firstname (#PCDATA)>`
3. `<!ELEMENT lastname (#PCDATA)>`
4. `<!ELEMENT email (#PCDATA)>`

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.

An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:

1. `<!ENTITY entity-name "entity-value">`

Let's see a code to define the ENTITY in doctype declaration.

author.xml

1. `<?xml version="1.0" standalone="yes" ?>`
2. `<!DOCTYPE author [`
3. `<!ELEMENT author (#PCDATA)>`
4. `<!ENTITY sj "Sonoo Jaiswal">`
5. `]>`
6. `<author>&sj;</author>`

In the above example, sj is an entity that is used inside the author element. In such case, it will print the value of sj entity that is "Sonoo Jaiswal".

: A single DTD can be used in many XML files.

7.XML Schema

XML schema is a language which is used for expressing constraint about XML documents. There are so many schema languages which are used now a days for example Relax- NG and XSD (XML schema definition).

An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure.

Checking Validation

An XML document is called "well-formed" if it contains the correct syntax. A well-formed and valid XML document is one which have been validated against Schema.

XML Schema Example

Let's create a schema file.

employee.xsd

1. `<?xml version="1.0"?>`
2. `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"`
3. `targetNamespace="http://www.javatpoint.com"`
4. `xmlns="http://www.javatpoint.com"`
5. `elementFormDefault="qualified">`
- 6.
7. `<xs:element name="employee">`
8. `<xs:complexType>`
9. `<xs:sequence>`
10. `<xs:element name="firstname" type="xs:string"/>`
11. `<xs:element name="lastname" type="xs:string"/>`
12. `<xs:element name="email" type="xs:string"/>`
13. `</xs:sequence>`
14. `</xs:complexType>`
15. `</xs:element>`
- 16.
17. `</xs:schema>`

Let's see the xml file using XML schema or XSD file.

employee.xml

1. `<?xml version="1.0"?>`
2. `<employee`
3. `xmlns="http://www.javatpoint.com"`
4. `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
5. `xsi:schemaLocation="http://www.javatpoint.com employee.xsd">`
- 6.
7. `<firstname>vimal</firstname>`
8. `<lastname>jaiswal</lastname>`
9. `<email>vimal@javatpoint.com</email>`
10. `</employee>`

Description of XML Schema

`<xs:element name="employee">` : It defines the element name employee.

`<xs:complexType>` : It defines that the element 'employee' is complex type.

`<xs:sequence>` : It defines that the complex type is a sequence of elements.

`<xs:element name="firstname" type="xs:string"/>` : It defines that the element 'firstname' is of string/text type.

`<xs:element name="lastname" type="xs:string"/>` : It defines that the element 'lastname' is of string/text type.

`<xs:element name="email" type="xs:string"/>` : It defines that the element 'email' is of string/text type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

simpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty.

complexType

The complexType allows you to hold multiple attributes and elements. It can contain additional sub elements and can be left empty

8.XML Documents and Database

Different approaches for storing XML documents are as given below:

- A RDBMS or object-oriented database management system is used to store XML document in the form of text.
- A tree model is very useful to store data elements located at leaf level in tree structure.
- The large amount of data is stored in the form of relational database or object-oriented databases. A middleware software is used to manage communication between XML document and relational database.

XML Query:

XML query is based on two methods.

1. Xpath:

- Xpath is a syntax for defining parts or elements of XML documents.
- Xpath is used to navigate between elements and attributes in the XML documents.
- Xpath uses path of expressions to select nodes in XML documents.

Example:

In this example, the tree structure of employee is represented in the form of XML document.

```
<Employee>
  <Name>
    <First name>Brian</First name>
    <Last name>Lara</Last name>
  </Name>
  <Contact>
    <Mobile>9800000000</Mobile>
    <Landline>020222222</Landline>
  </Contact>
  <Address>
    <City>Pune</city>
    <Street>Tilak road</Street>
    <Zip code>4110</Zip code>
  </Address>
</Employee>
```

In the above example, a (/) is used as </Employee> where Employee is a root node and Name, Contact and Address are descendant nodes.

2. Xquery:

- Xquery is a query and functional programming language. Xquery provides a facility to extract and manipulate data from XML documents or any data source, like relational database.

- The Xquery defines FLWR expression which supports iteration and binding of variable to intermediate results.

FLWR is an abbreviation of FOR, LET, WHERE, RETURN. Which are explained as follows:

FOR: Creates a sequence of nodes.

LET: Binds a sequence to variable.

WHERE: Filters the nodes on condition.

RETURN: It is query result specification.

Xquery comparisons:

The two methods for Xquery comparisons are as follows:

1. **General comparisons:** =, !=, <=, >, >=

Example:

In this example, the expression (Query) can return true value if any attributes have a value greater than or equal to 15000.
\$ TVStore//TV/price > =15000

2. Value comparisons: eq, ne, lt, le, gt, ge

Example:

In this example, the expression (Query) can return true value, if there is only one attribute returned by the expression, and its value is equal to 15000.

```
$ TVStore//TV/price eq 15000
```

Example:

Lets take an example to understand how to write a XMLquery.

```
FOR $x in doc ("student.xml")/student information /marks
WHERE $x/marks >700
RETURN <res>
      $x/Name
    </res>
```

The queries in the above example can return true value (Name of the students) only if the value (marks) is greater than 700.

9.XPath

XPath uses path expressions to select nodes or node-sets in an XML document.

The node is selected by following a path or steps

The XML Example Document

We will use the following XML document in the examples below.

```
<?xml version="1.0" encoding="UTF-8"?>
```

<bookstore>

<book>

<title lang="en">Harry Potter</title>

<price>29.99</price>

</book>

<book>

<title lang="en">Learning XML</title>

<price>39.95</price>

</book>

</bookstore>

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
<i>nodename</i>	Selects all nodes with the name " <i>nodename</i> "

/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"

/bookstore	Selects the root element bookstore
Note: If the path starts with a slash (/) it always represents an absolute path to an element!	
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Predicates

Predicates are used to find a specific node or a node that contains a specific value.

Predicates are always embedded in square brackets.

In the table below we have listed some path expressions with predicates and the result of the expressions:

Path Expression	Result
-----------------	--------

/bookstore/book[1]	<p>Selects the first book element that is the child of the bookstore element.</p> <p>Note: In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath:</p> <p><i>In</i> <i>JavaScript:</i></p> <pre>xml.setProperty("SelectionLanguage","XPath");</pre>
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"

<code>/bookstore/book[price>35.00]</code>	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
<code>/bookstore/book[price>35.00]/title</code>	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Selecting Unknown Nodes

XPath wildcards can be used to select unknown XML nodes.

Wildcard	Description
<code>*</code>	Matches any element node
<code>@*</code>	Matches any attribute node
<code>node()</code>	Matches any node of any kind

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

Selecting Several Paths

By using the | operator in an XPath expression you can select several paths.

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
<code>//book/title //book/price</code>	Selects all the title AND price elements of all book elements
<code>//title //price</code>	Selects all the title AND price elements in the document
<code>/bookstore/book/title //price</code>	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

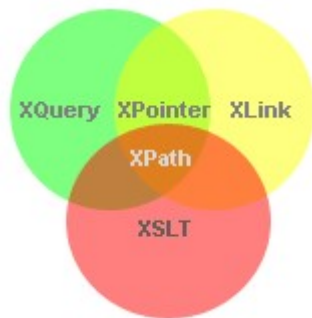
10.XQuery

XQuery is a functional query language used to retrieve information stored in XML format. It is same as for XML what SQL is for databases. It was designed to query XML data.

XQuery is built on XPath expressions. It is a W3C recommendation which is supported by all major databases.

The as it is definition of XQuery given by its official documentation is as follows:

"XQuery is a standardized language for combining documents, databases, Web pages and almost anything else. It is very widely implemented. It is powerful and easy to learn. XQuery is replacing proprietary middleware languages and Web Application development languages. XQuery is replacing complex Java or C++ programs with a few lines of code. XQuery is simpler to work with and easier to maintain than many other alternatives".



What does it do

XQuery is a functional language which is responsible for finding and extracting elements and attributes from XML documents.

It can be used for following things:

- To extract information to use in a web service.
- To generates summary reports.
- To transform XML data to XHTML.
- Search Web documents for relevant information.

Unit 5

1.Information Retrieval (IR) Concepts

Information retrieval is the process of retrieving documents from a collection in response to a query (or a search request) by a user. This section provides an overview of information retrieval (IR) concepts. In Section 27.1.1, we introduce information retrieval in general and then discuss the different kinds and levels of search that IR encompasses. In Section 27.1.2, we compare IR and database technologies. Section 27.1.3 gives a brief history of IR. We then present the different modes of user interaction with IR systems in Section 27.1.4. In Section 27.1.5, we describe the typical IR process with a detailed set of tasks and then with a simplified process flow, and end with a brief discussion of digital libraries and the Web.

1. Introduction to Information Retrieval

We first review the distinction between structured and unstructured data (see Section 12.1) to see how information retrieval differs from structured data management. Consider a relation (or table) called HOUSES with the attributes:

HOUSES(Lot#, Address, Square_footage, Listed_price)

This is an example of *structured data*. We can compare this relation with home-buying contract documents, which are examples of *unstructured data*. These types of documents can vary from city to city, and even county to county, within a given state in the United States. Typically, a contract document in a particular state will

have a standard list of clauses described in paragraphs within sections of the document, with some predetermined (fixed) text and some variable areas whose content is to be supplied by the specific buyer and seller. Other variable information would include interest rate for financing, down-payment amount, closing dates, and so on. The documents could also possibly include some pictures taken during a home inspection. The information content in such documents can be considered *unstructured data* that can be stored in a variety of possible arrangements and for-mats. By **unstructured information**, we generally mean information that does not have a well-defined formal model and corresponding formal language for representation and reasoning, but rather is based on understanding of natural language.

2. Databases and IR Systems: A Comparison

Within the computer science discipline, databases and IR systems are closely related fields. Databases deal with structured information retrieval through well-defined formal languages for representation and manipulation based on the theoretically founded data models. Efficient algorithms have been developed for operators that allow rapid execution of complex queries. IR, on the other hand, deals with unstructured search with possibly vague query or search semantics and without a well-defined logical schematic representation. Some of the key differences between databases and IR systems are listed in Table 27.1.

Table 27.1 A Comparison of Databases and IR Systems

Databases

- Structured data
- Schema driven
- Relational (or object, hierarchical, and network) model is predominant
- Structured query model
- Rich metadata operations
- Query returns data
- Results are based on exact matching (always correct)

IR Systems

- Unstructured data
 - No fixed schema; various data models (e.g., vector space model)
 - Free-form query models
 - Rich data operations
 - Search request returns list or pointers to documents
 - Results are based on approximate matching and measures of effectiveness (may be imprecise and ranked)
-

3. A Brief History of IR

Information retrieval has been a common task since the times of ancient civilizations, which devised ways to organize, store, and catalog documents and records. Media such as papyrus scrolls and stone tablets were used to record documented information in ancient times. These efforts allowed knowledge to be retained and transferred among generations. With the emergence of public libraries and the printing press, large-scale methods for producing, collecting, archiving, and distributing documents and books evolved. As computers and automatic storage systems emerged, the need to apply these methods to computerized systems arose. Several techniques emerged in the 1950s, such as the seminal work of H. P. Luhn, who proposed using words and their frequency counts as indexing units for documents, and using measures of word overlap between queries and documents as the retrieval criterion. It was soon realized that storing large amounts of text was not difficult. The harder task was to search for and retrieve that information selectively for users with specific information needs. Methods that

explored word distribution statistics gave rise to the choice of keywords based on their distribution properties and keyword-based weighting schemes.

4. Modes of Interaction in IR Systems

In the beginning of Section 27.1, we defined information retrieval as the process of retrieving documents from a collection in response to a query (or a search request) by a user. Typically the collection is made up of documents containing unstructured data. Other kinds of documents include images, audio recordings, video strips, and maps. Data may be scattered nonuniformly in these documents with no definitive structure. A **query** is a set of **terms** (also referred to as **keywords**) used by the searcher to specify an information need (for example, the terms ‘databases’ and ‘operating systems’ may be regarded as a query to a computer science bibliographic database). An informational request or a search query may also be a natural language phrase or a question (for example, “What is the currency of China?” or “Find Italian restaurants in Sarasota, Florida.”).

Web search combines both aspects—browsing and retrieval—and is one of the main applications of information retrieval today. Web pages are analogous to documents. Web search engines maintain an indexed repository of Web pages, usually using the technique of inverted indexing (see Section 27.5). They retrieve the most relevant Web pages for the user in response to the user’s search request with a possible ranking in descending order of relevance. The **rank of a Webpage** in a retrieved set is the measure of its relevance to the query that generated the result set.

5. Generic IR Pipeline

As we mentioned earlier, documents are made up of unstructured natural language text composed of character strings from English and other languages. Common examples of documents include newswire services (such as AP or Reuters), corporate manuals and reports, government notices, Web page articles, blogs, tweets, books, and journal papers. There are two main approaches to IR: statistical and semantic.

In a **statistical approach**, documents are analyzed and broken down into chunks of text (words, phrases, or n -grams, which are all subsequences of length n characters in a text or document) and each word or phrase is counted, weighted, and measured for relevance or importance. These words and their properties are then compared with the query terms for potential degree of match to produce a ranked list of resulting documents that contain the words. Statistical approaches are further classified based on the method employed. The three main statistical approaches are Boolean, vector space, and probabilistic (see Section 27.2).

Semantic approaches to IR use knowledge-based techniques of retrieval that broadly rely on the syntactic, lexical, sentential, discourse-based, and pragmatic levels of knowledge understanding. In practice, semantic approaches also apply some form of statistical analysis to improve the retrieval process.

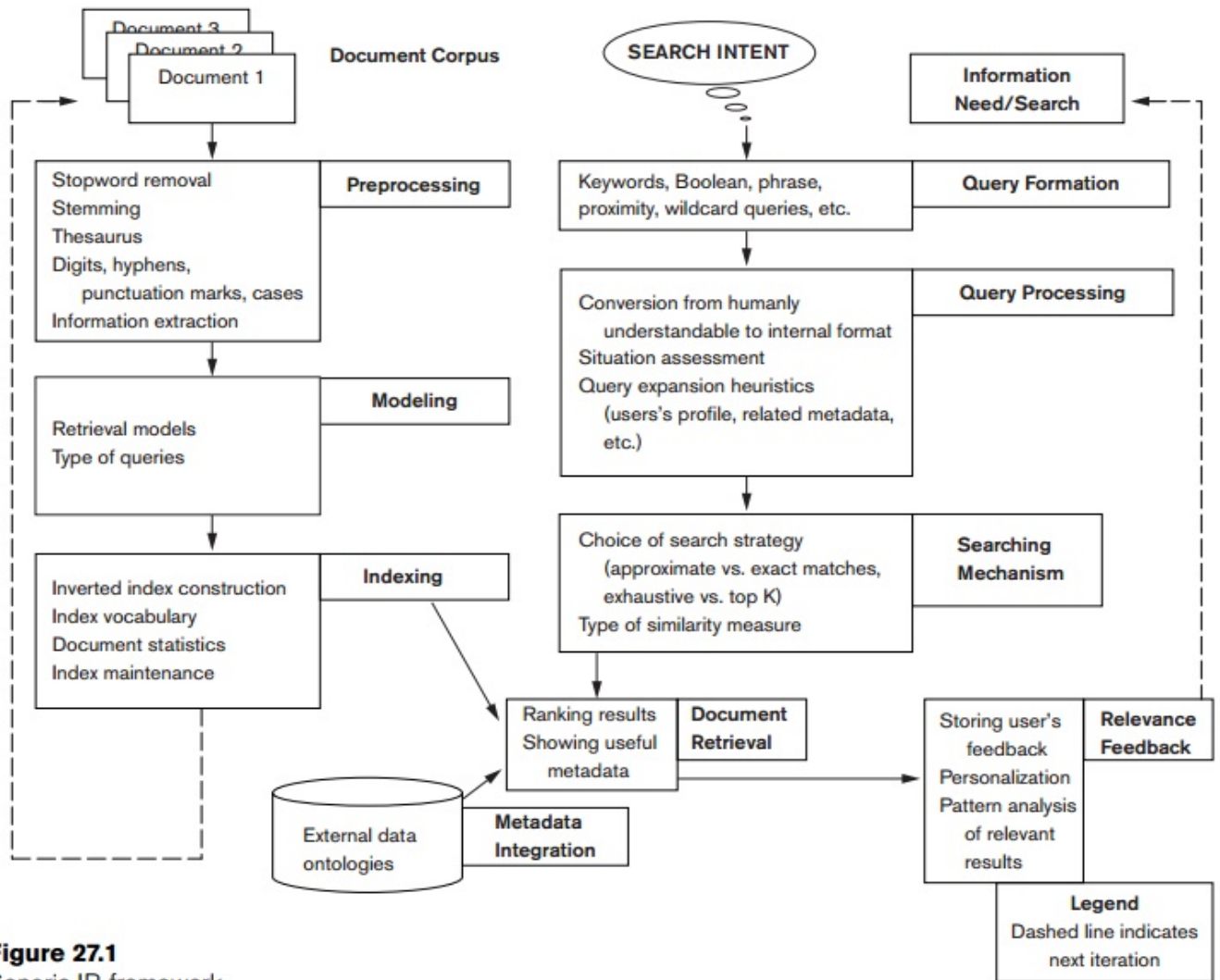


Figure 27.1
Generic IR framework.

2.Retrieval Models

In this section we briefly describe the important models of IR. These are the three main statistical models—Boolean, vector space, and probabilistic—and the semantic model.

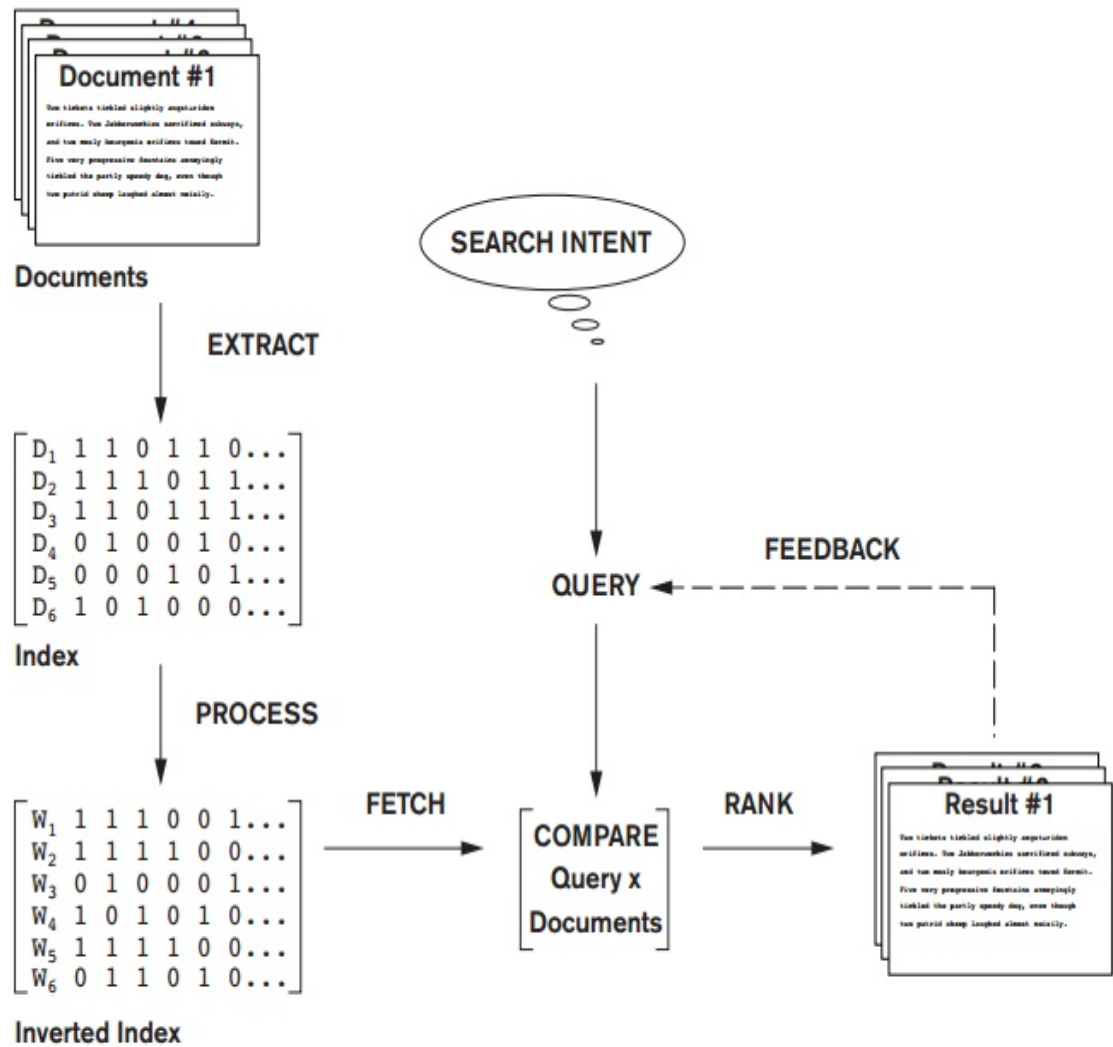


Figure 27.2
Simplified IR process pipeline.

1. Boolean Model

In this model, documents are represented as a set of *terms*. Queries are formulated as a combination of terms using the standard Boolean logic set-theoretic operators

such as AND, OR and NOT. Retrieval and relevance are considered as binary concepts in this model, so the retrieved elements are an “exact match” retrieval of relevant documents. There is no notion of ranking of resulting documents. All retrieved documents are considered equally important—a major simplification that does not consider frequencies of document terms or their proximity to other terms compared against the query terms.

Boolean retrieval models lack sophisticated ranking algorithms and are among the earliest and simplest information retrieval models. These models make it easy to associate metadata information and write queries that match the contents of the documents as well as other properties of documents, such as date of creation, author, and type of document.

2. Vector Space Model

The vector space model provides a framework in which term weighting, ranking of retrieved documents, and relevance feedback are possible. Documents are represented as *features* and *weights* of term features in an n -dimensional vector space of terms. **Features** are a subset of the terms in a *set of documents* that are deemed most relevant to an IR search for this particular set of documents. The process of selecting these important terms (features) and their properties as a sparse (limited) list out of the very large number of available terms (the vocabulary can contain hundreds of thousands of terms) is independent of the model specification. The query is also specified as a terms vector (vector of features), and this is compared to the document vectors for similarity/relevance assessment.

$$\text{cosine}(d_j, q) = \frac{\langle d_j \times q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

In the formula given above, we use the following symbols:

d_j is the document vector.

q is the query vector.

w_{ij} is the weight of term i in document j .

w_{iq} is the weight of term i in query vector q .

$|V|$ is the number of dimensions in the vector that is the total number of important keywords (or features).

3. Probabilistic Model

The similarity measures in the vector space model are somewhat ad hoc. For example, the model assumes that those documents closer to the query in cosine space are more relevant to the query vector. In the probabilistic model, a more concrete and definitive approach is taken: ranking documents by their estimated

probability of relevance with respect to the query and the document. This is the basis of the *Probability Ranking Principle* developed by Robertson:

In the probabilistic framework, the IR system has to decide whether the documents belong to the **relevant set** or the **nonrelevant set** for a query. To make this decision, it is assumed that a predefined relevant set and nonrelevant set exist for the query, and the task is to calculate the probability that the document belongs to the relevant set and compare that with the probability that the document belongs to the nonrelevant set.

Given the document representation D of a document, estimating the relevance R and nonrelevance NR of that document involves computation of conditional probability $P(R|D)$ and $P(NR|D)$. These conditional probabilities can be calculated using Bayes' Rule:

$$P(R|D) = P(D|R) \times P(R)/P(D)$$

$$P(NR|D) = P(D|NR) \times P(NR)/P(D)$$

A document D is classified as relevant if $P(R|D) > P(NR|D)$. Discarding the constant $P(D)$, this is equivalent to saying that a document is relevant if:

$$P(D|R) \times P(R) > P(D|NR) \times P(NR)$$

4. Semantic Model

However sophisticated the above statistical models become, they can miss many relevant documents because those models do not capture the complete meaning or information need conveyed by a user's query. In semantic models, the process of matching documents to a given query is based on concept level and semantic matching instead of index term (keyword) matching. This allows retrieval of relevant documents that share meaningful associations with other documents in the query result, even when these associations are not inherently observed or statistically captured.

Semantic approaches include different levels of analysis, such as morphological, syntactic, and semantic analysis, to retrieve documents more effectively. In **morphological analysis**, roots and affixes are analyzed to determine the parts of speech (nouns, verbs, adjectives, and so on) of the words. Following morphological analysis, **syntactic analysis** follows to parse and analyze complete phrases in documents. Finally, the semantic methods have to resolve word ambiguities and/or generate relevant synonyms based on the **semantic relationships** between levels of structural entities in documents (words, paragraphs, pages, or entire documents).

3.Types of Queries in IR Systems

During the process of indexing, many keywords are associated with document set which contains words, phrases, date created, author names, and type of document. They are used by an IR system to build an inverted index which is then consulted during the search. The queries formulated by users are compared to the set of index keywords. Most IR systems also allow the use of Boolean and other operators to build a complex query. The query language with these operators enriches the expressiveness of a user's information need. The [Information Retrieval \(IR\)](#) system finds the relevant documents from a large data set according to the user query. Queries submitted by users to search engines might be ambiguous, concise and their meaning may change over time. Some of the types of Queries in IR systems are – **1. Keyword Queries :**

- Simplest and most common queries.
- The user enters just keyword combinations to retrieve documents.
- These keywords are connected by logical AND operator.
- All retrieval models provide support for keyword queries.

2. Boolean Queries :

- Some IR systems allow using +, -, AND, OR, NOT, (), Boolean operators in combination of keyword formulations.
- No ranking is involved because a document either satisfies such a query or does not satisfy it.
- A document is retrieved for boolean query if it is logically true as exact match in document.

3. Phrase Queries :

- When documents are represented using an inverted keyword index for searching, the relative order of items in document is lost.
- To perform exact phrase retrieval, these phrases are encoded in inverted index or implemented differently.

- This query consists of a sequence of words that make up a phrase.
- It is generally enclosed within double quotes.

4. Proximity Queries :

- Proximity refers to search that accounts for how close within a record multiple items should be to each other.
- Most commonly used proximity search option is a phrase search that requires terms to be in exact order.
- Other proximity operators can specify how close terms should be to each other. Some will specify the order of search terms.
- Search engines use various operators names such as NEAR, ADJ (adjacent), or AFTER.
- However, providing support for complex proximity operators becomes expensive as it requires time-consuming pre-processing of documents and so it is suitable for smaller document collections rather than for web.

5. Wildcard Queries :

- It supports regular expressions and pattern matching-based searching in text.
- Retrieval models do not directly support for this query type.
- In IR systems, certain kinds of wildcard search support may be implemented. Example: usually words ending with trailing characters.

6. Natural Language Queries :

- There are only a few natural language search engines that aim to understand the structure and meaning of queries written in natural language text, generally as question or narrative.
- The system tries to formulate answers for these queries from retrieved results.
- Semantic models can provide support for this query type.

4.Text Preprocessing

In this section we review the commonly used text preprocessing techniques that are part of the text processing task in Figure 27.1.

1. Stopword Removal

Stopwords are very commonly used words in a language that play a major role in the formation of a sentence but which seldom contribute to the meaning of that sentence. Words that are expected to occur in 80 percent or more of the documents in a collection are typically referred to as *stopwords*, and they are rendered potentially useless. Because of the commonness and function of these words, they do not contribute much to the relevance of a document for a query search. Examples include words such as *the, of, to, a, and, in, said, for, that, was, on, he, is, with, at, by,* and *it*. These words are presented here with decreasing frequency of occurrence from a large corpus of documents called **AP89**. The first six of these words account for 20 percent of all words in the listing, and the most frequent 50 words account for 40 percent of all text.

Removal of stopwords from a document must be performed before indexing. Articles, prepositions, conjunctions, and some pronouns are generally classified as stopwords. Queries must also be preprocessed for stopword removal before the actual retrieval process. Removal of stopwords results in elimination of possible

spurious indexes, thereby reducing the size of an index structure by about 40 percent or more. However, doing so could impact the recall if the stopword is an integral part of a query (for example, a search for the phrase ‘To be or not to be,’ where removal of stopwords makes the query inappropriate, as all the words in the phrase are stopwords). Many search engines do not employ query stopword removal for this reason.

2. Stemming

A **stem** of a word is defined as the word obtained after trimming the suffix and prefix of an original word. For example, ‘comput’ is the stem word for *computer*, *computing*, and *computation*. These suffixes and prefixes are very common in the English language for supporting the notion of verbs, tenses, and plural forms. **Stemming** reduces the different forms of the word formed by inflection (due to plurals or tenses) and derivation to a common stem.

A stemming algorithm can be applied to reduce any word to its stem. In English, the most famous stemming algorithm is Martin Porter’s stemming algorithm. The Porter stemmer is a simplified version of Lovin’s technique that uses a reduced set of about 60 rules (from 260 suffix patterns in Lovin’s technique) and organizes them into sets; conflicts within one subset of rules are resolved before going on to the next. Using stemming for preprocessing data results in a decrease in the size of the indexing structure and an increase in recall, possibly at the cost of precision.

3. Utilizing a Thesaurus

A **thesaurus** comprises a precompiled list of important concepts and the main word that describes each concept for a particular domain of knowledge. For each concept in this list, a set of synonyms and related words is also compiled. Thus, a synonym can be converted to its matching concept during preprocessing. This preprocessing step assists in providing a standard vocabulary for indexing and searching. Usage of a thesaurus, also known as a *collection of synonyms*, has a substantial impact on the recall of information systems. This process can be complicated because many words have different meanings in different contexts.

UMLS is a large biomedical thesaurus of millions of concepts (called the *Metathesaurus*) and a semantic network of meta concepts and relationships that organize the Metathesaurus (see Figure 27.3). The concepts are assigned labels from the semantic network. This thesaurus of concepts contains synonyms of medical terms, hierarchies of broader and narrower terms, and other relationships among words and concepts that make it a very extensive resource for information retrieval of documents in the medical domain. Figure 27.3 illustrates part of the UMLS Semantic Network.

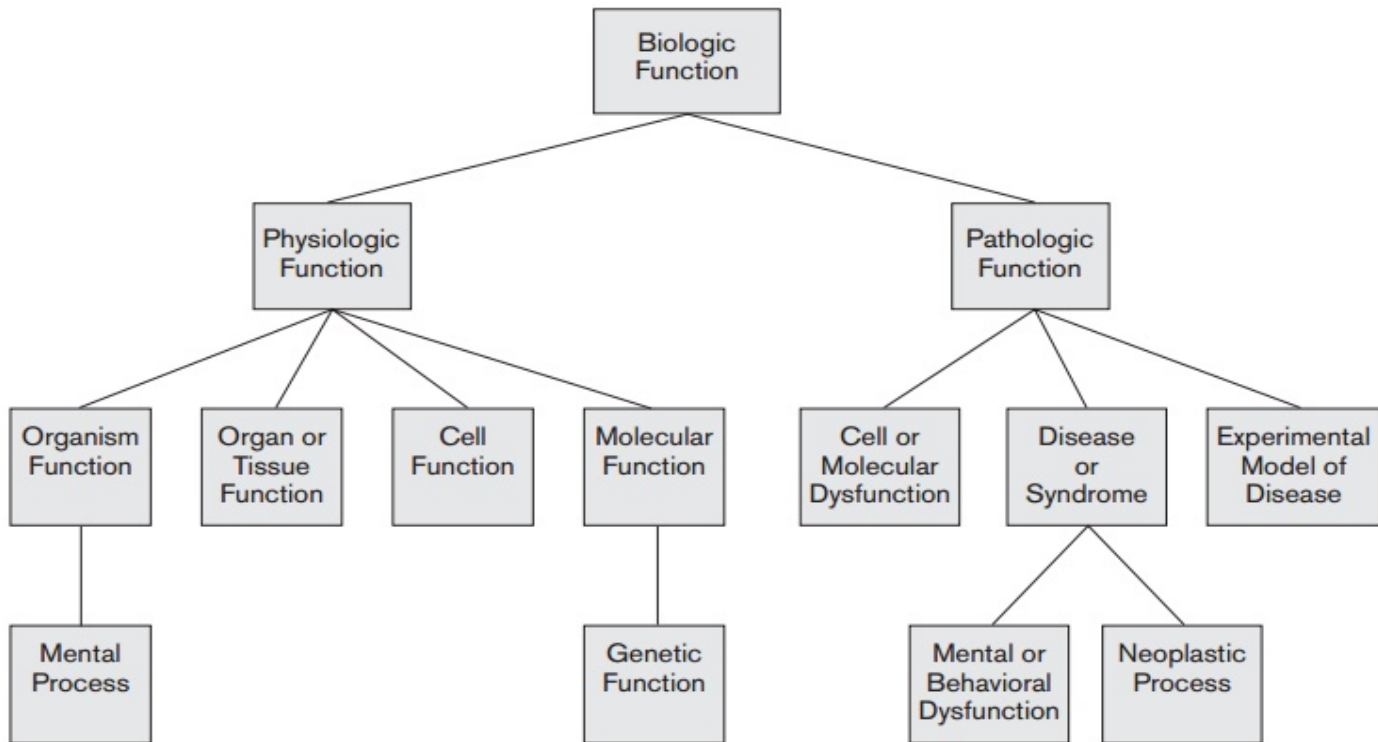


Figure 27.3

A Portion of the UMLS Semantic Network: "Biologic Function" Hierarchy

Source: UMLS Reference Manual, National Library of Medicine.

WordNet is a manually constructed thesaurus that groups words into strict synonym sets called *synsets*. These synsets are divided into noun, verb, adjective, and adverb categories. Within each category, these synsets are linked together by appropriate relationships such as class/subclass or "is-a" relationships for nouns.

WordNet is based on the idea of using a controlled vocabulary for indexing, thereby eliminating redundancies. It is also useful in providing assistance to users with locating terms for proper query formulation.

4. Other Preprocessing Steps: Digits, Hyphens, Punctuation Marks, Cases

Digits, dates, phone numbers, e-mail addresses, URLs, and other standard types of text may or may not be removed during preprocessing. Web search engines, however, index them in order to use this type of information in the document metadata to improve precision and recall (see Section 27.6 for detailed definitions of *precision* and *recall*).

Hyphens and punctuation marks may be handled in different ways. Either the entire phrase with the hyphens/punctuation marks may be used, or they may be eliminated. In some systems, the character representing the hyphen/punctuation mark may be removed, or may be replaced with a space. Different information retrieval systems follow different rules of processing. Handling hyphens automatically can be complex: it can either be done as a classification problem, or more commonly by some heuristic rules.

Most information retrieval systems perform case-insensitive search, converting all the letters of the text to uppercase or lowercase. It is also worth noting that many of these text preprocessing steps are language specific, such as involving accents and diacritics and the idiosyncrasies that are associated with a particular language.

5. Information Extraction

Information extraction (IE) is a generic term used for extracting structured content from text. Text analytic tasks such as identifying noun phrases, facts, events,

people, places, and relationships are examples of IE tasks. These tasks are also called *named entity recognition tasks* and use rule-based approaches with either a the-saurus, regular expressions and grammars, or probabilistic approaches. For IR and search applications, IE technologies are mostly used to identify contextually relevant features that involve text analysis, matching, and categorization for improving the relevance of search systems. Language technologies using part-of-speech tagging are applied to semantically annotate the documents with extracted features to aid search relevance.

5.Inverted Indexing

The simplest way to search for occurrences of query terms in text collections can be performed by sequentially scanning the text. This kind of online searching is only appropriate when text collections are quite small. Most information retrieval systems process the text collections to create indexes and operate upon the inverted index data structure (refer to the indexing task in Figure 27.1). An inverted index structure comprises vocabulary and document information. **Vocabulary** is a set of distinct query terms in the document set. Each term in a vocabulary set has an associated collection of information about the documents that contain the term, such as document id, occurrence count, and offsets within the document where the term occurs. The simplest form of vocabulary terms consists of words or individual tokens of the documents. In some cases, these vocabulary terms also consist of phrases, n-grams, entities, links, names, dates, or manually assigned descriptor terms from documents and/or Web pages. For each term in the vocabulary, the corresponding document ids, occurrence locations of the term in each document, number of occurrences of the term in each document, and other relevant information may be stored in the document information section.

Weights are assigned to document terms to represent an estimate of the usefulness of the given term as a descriptor for distinguishing the given document from other documents in the same collection. A term may be a better descriptor of one document than of another by the weighting process (see Section 27.2).

An **inverted index** of a document collection is a data structure that attaches distinct terms with a list of all documents that contains the term. The process of inverted index construction involves the extraction and processing steps shown in Figure 27.2. Acquired text is first preprocessed and the documents are represented with the vocabulary terms. Documents' statistics are collected in document lookup tables. Statistics generally include counts of vocabulary terms in individual documents as well as different collections, their positions of occurrence within the documents, and the lengths of the documents. The vocabulary terms are weighted at indexing time according to different criteria for collections. For example, in some cases terms in the titles of the documents may be weighted more heavily than terms that occur in other parts of the documents.

One of the most popular weighting schemes is the TF-IDF (term frequency-inverse document frequency) metric that we described in Section 27.2. For a given term this weighting scheme distinguishes to some extent the documents in which the term occurs more often from those in which the term occurs very little or never. These weights are normalized to account for varying document lengths, further ensuring that longer documents with proportionately more occurrences of a word are not favored for retrieval over shorter documents with proportionately fewer

occurrences. These processed document-term streams (matrices) are then inverted into term-document streams (matrices) for further IR steps.

Figure 27.4 shows an illustration of term-document-position vectors for the four illustrative terms—*example*, *inverted*, *index*, and *market*—which refer to the three documents and the position where they occur in those documents.

The different steps involved in inverted index construction can be summarized as follows:

1. Break the documents into vocabulary terms by tokenizing, cleansing, stopword removal, stemming, and/or use of an additional thesaurus as vocabulary.
2. Collect document statistics and store the statistics in a document lookup table.
3. Invert the document-term stream into a term-document stream along with additional information such as term frequencies, term positions, and term weights.

Searching for relevant documents from the inverted index, given a set of query terms, is generally a three-step process.

Vocabulary search. If the query comprises multiple terms, they are separated and treated as independent terms. Each term is searched in the vocabulary. Various data structures, like variations of B⁺-tree or hashing, may be

Document 1

This example shows an example of an inverted index.

Document 2

Inverted index is a data structure for associating terms to documents.

Document 2

Stock market index is used for capturing the sentiments of the financial market.

ID	Term	Document: position
1.	example	1:2, 1:5
2.	inverted	1:8, 2:1
3.	index	1:9, 2:2, 3:3
4.	market	3:2, 3:13

Figure 27.4
Example of an inverted index.

used to optimize the search process. Query terms may also be ordered in lexicographic order to improve space efficiency.

Document information retrieval. The document information for each term is retrieved.

Manipulation of retrieved information. The document information vector for each term obtained in step 2 is now processed further to incorporate various forms of query logic. Various kinds of queries like prefix, range, context, and proximity queries are processed in this step to construct the final result based on the document collections returned in step 2.

6.Evaluation Measures of Search Relevance

Without proper evaluation techniques, one cannot compare and measure the relevance of different retrieval models and IR systems in order to make improvements.

Evaluation techniques of IR systems measure the *topical relevance* and *user relevance*. **Topical relevance** measures the extent to which the topic of a result matches the topic of the query. Mapping one's information need with "perfect" queries is a cognitive task, and many users are not able to effectively form queries that would retrieve results more suited to their information need. Also, since a major chunk of user queries are informational in nature, there is no fixed set of right answers to show to the user. **User relevance** is a term used to describe the "goodness" of a retrieved result with regard to the user's information need. User relevance includes other implicit factors, such as user perception,

context, timeliness, the user's environment, and current task needs. Evaluating user relevance may also involve subjective analysis and study of user retrieval tasks to capture some of the properties of implicit factors involved in accounting for users' bias for judging performance.

In Web information retrieval, no binary classification decision is made on whether a document is relevant or nonrelevant to a query (whereas the Boolean (or binary) retrieval model uses this scheme, as we discussed in Section 27.2.1). Instead, a ranking of the documents is produced for the user. Therefore, some evaluation measures focus on comparing different rankings produced by IR systems. We discuss some of these measures next.

1. Recall and Precision

Recall and precision metrics are based on the binary relevance assumption (whether each document is relevant or nonrelevant to the query). **Recall** is defined as the number of relevant documents retrieved by a search divided by the total number of existing relevant documents. **Precision** is defined as the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search. Figure 27.5 is a pictorial representation of the terms *retrieved* vs. *relevant* and shows how search results relate to four different sets of documents.





		Relevant?	
		Yes	No
Retrieved?	Yes	 Hits TP	 False Alarms FP
	No	Misses FN 	Correct Rejections TN 

Figure 27.5
Retrieved vs. relevant
search results.

The notation for Figure 27.5 is as follows:

TP: true positive

FP: false positive

FN: false negative

TN: true negative

The terms *true positive*, *false positive*, *false negative*, and *true negative* are generally used in any type of classification tasks to compare the given classification of an item with the desired correct classification. Using the

term *hits* for the documents that truly or “correctly” match the user request, we can define:

$$\text{Recall} = |\text{Hits}|/|\text{Relevant}|$$

$$\text{Precision} = |\text{Hits}|/|\text{Retrieved}|$$

Recall and precision can also be defined in a ranked retrieval setting. The Recall at rank position i for document d_i^q (denoted by $r(i)$) (d_i^q is the retrieved document at position i for query q) is the fraction of relevant documents from d_1^q to d_i^q in the result set for the query. Let the set of relevant documents from d_1^q to d_i^q in that set be S_i with cardinality $|S_i|$. Let $(|D_q|)$ be the size of relevant documents for the query. In this case, $|S_i| \leq |D_q|$). Then:

$$\text{Recall } r(i) = |S_i|/|D_q|$$

The Precision at rank position i or document d_i^q (denoted by $p(i)$) is the fraction of documents from d_1^q to d_i^q in the result set that are relevant:

$$\text{Precision } p(i) = |S_i|/i$$

Table 27.2 illustrates the $p(i)$, $r(i)$, and average precision (discussed in the next section) metrics. It can be seen that recall can be increased by presenting more results to the user, but this approach runs the risk of decreasing the precision. In the

Table 27.2 Precision and Recall for Ranked Retrieval

Doc. No.	Rank Position i	Relevant	Precision(i)	Recall(i)
10	1	Yes	1/1 = 100%	1/10 = 10%
2	2	Yes	2/2 = 100%	2/10 = 20%
3	3	Yes	3/3 = 100%	3/10 = 30%
5	4	No	3/4 = 75%	3/10 = 30%
17	5	No	3/5 = 60%	3/10 = 30%
34	6	No	3/6 = 50%	3/10 = 30%
215	7	Yes	4/7 = 57.1%	4/10 = 40%
33	8	Yes	5/8 = 62.5%	5/10 = 50%
45	9	No	5/9 = 55.5%	5/10 = 50%
16	10	Yes	6/10 = 60%	6/10 = 60%

example, the number of relevant documents for some query = 10. The rank position and the relevance of an individual document are shown. The precision and recall value can be computed at each position within the ranked list as shown in the last two columns.

2. Average Precision

Average precision is computed based on the precision at each relevant document in the ranking. This measure is useful for computing a single precision value to compare different retrieval algorithms on a query q .

$$P_{\text{avg}} = \sum_{d_i^q \in D_q} p(i) / |D_q|$$

Consider the sample precision values of relevant documents in Table 27.2. The average precision (P_{avg} value) for the example in Table 27.2 is $P(1) + P(2) + P(3) + P(7) + P(8) + P(10)/6 = 79.93$ percent (only relevant documents are considered in this calculation). Many good algorithms tend to have high top-k average precision for small values of k, with correspondingly low values of recall.

3. Recall/Precision Curve

A recall/precision curve can be drawn based on the recall and precision values at each rank position, where the x-axis is the recall and the y-axis is the precision. Instead of using the precision and recall at each rank position, the curve is commonly plotted using recall levels $r(i)$ at 0 percent, 10 percent, 20 percent...100 per-cent. The curve usually has a negative slope, reflecting the inverse relationship between precision and recall.

4. F-Score

F-score (F) is the harmonic mean of the precision (p) and recall (r) values. High precision is achieved almost always at the expense of recall and vice versa. It is a matter of the application's context whether to tune the system for high precision or high recall. F-score is a single measure that combines precision and recall to compare different result sets:

$$F = \frac{2pr}{p+r}$$

One of the properties of harmonic mean is that the harmonic mean of two numbers tends to be closer to the smaller of the two. Thus F is automatically biased toward the smaller of the precision and recall values. Therefore, for a high F-score, both precision and recall must be high.

$$F = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

7.Web Search and Analysis

The emergence of the Web has brought millions of users to search for information, which is stored in a very large number of active sites. To make this information accessible, search engines such as Google and Yahoo! have to crawl and index these sites and document collections in their index databases. Moreover, search engines have to regularly update their indexes given the dynamic nature of the Web as new Web sites are created and current ones are updated or deleted. Since there are many millions of pages available on the Web on different topics, search

engines have to apply many sophisticated techniques such as link analysis to identify the importance of pages.

There are other types of search engines besides the ones that regularly crawl the Web and create automatic indexes: these are human-powered, vertical search engines or metasearch engines. These search engines are developed with the help of computer-assisted systems to aid the curators with the process of assigning indexes. They consist of manually created specialized Web directories that are hierarchically organized indexes to guide user navigation to different resources on the Web. **Vertical search engines** are customized topic-specific search engines that crawl and index a specific collection of documents on the Web and provide search results from that specific collection. **Metasearch engines** are built on top of search engines: they query different search engines simultaneously and aggregate and provide search results from these sources.

1. Web Analysis and Its Relationship to Information Retrieval

The goals of Web analysis are to improve and personalize search results relevance and to identify trends that may be of value to various businesses and organizations. We elaborate on these goals next.

Finding relevant information. People usually search for specific information on the Web by entering keywords in a search engine or browsing information portals and using services. Search services are constrained by search relevance

problems since they have to map and approximate the information need of millions of users as an *a priori* task. Low *precision* (see Section 27.6) ensues due to results that are nonrelevant to the user. In the case of the Web, high *recall* (see section 27.6) is impossible to determine due to the inability to index all the pages on the Web. Also, measuring recall does not make sense since the user is concerned with only the top few documents. The most relevant feedback for the user is typically from only the top few results.

Personalization of the information. Different people have different content and presentation preferences. By collecting personal information and then generating user-specific dynamic Web pages, the pages are personalized for the user. The customization tools used in various Web-based applications and services, such as click-through monitoring, eyeball tracking, explicit or implicit user profile learning, and dynamic service composition using Web APIs, are used for service adaptation and personalization. A personalization engine typically has algorithms that make use of the user's personalization information—collected by various tools—to generate user-specific search results.

Finding information of commercial value. This problem deals with finding interesting patterns in users' interests, behaviors, and their use of products and services, which may be of commercial value. For example, businesses such as the automobile industry, clothing, shoes, and cosmetics may improve their services by identifying patterns such as usage trends and user preferences using various Web analysis techniques.

Based on the above goals, we can classify Web analysis into three categories: **Web content analysis**, which deals with extracting useful information/knowledge from Web page contents; **Web structure analysis**, which discovers knowledge from hyperlinks representing the structure of the Web; and **Web usage analysis**, which mines user access patterns from usage logs that record the activity of every user.

2. Searching the Web

The World Wide Web is a huge corpus of information, but locating resources that are both high quality and relevant to the needs of the user is very difficult. The set of Web pages taken as a whole has almost no unifying structure, with variability in authoring style and content, thereby making it more difficult to precisely locate needed information. Index-based search engines have been one of the prime tools by which users search for information on the Web. Web search engines **crawl** the Web and create an index to the Web for searching purposes. When a user specifies his need for information by supplying keywords, these Web search engines query their repository of indexes and produce links or URLs with abbreviated content as search results. There may be thousands of pages relevant to a particular query. A problem arises when only a few most relevant results are to be returned to the user. The discussion we had about querying and relevance-based ranking in IR systems in Sections 27.2 and 27.3 is applicable to Web search engines. These ranking algorithms explore the link structure of the Web.

3. Analyzing the Link Structure of Web Pages

The goal of **Web structure analysis** is to generate structural summary about the Website and Web pages. It focuses on the inner structure of documents and deals with the link structure using hyperlinks at the interdocument level. The structure and content of Web pages are often combined for information retrieval by Web search engines. Given a collection of interconnected Web documents, interesting and informative facts describing their connectivity in the Web subset can be discovered. Web structure analysis is also used to reveal the structure of Web pages, which helps with navigation and makes it possible to compare/integrate Web page schemes. This aspect of Web structure analysis facilitates Web document classification and clustering on the basis of structure.

The *PageRank* Ranking Algorithm. As discussed earlier, ranking algorithms are used to order search results based on relevance and authority. Google uses the well-known **PageRank** algorithm, which is based on the “importance” of each page. Every Web page has a number of forward links (out-edges) and backlinks (in-edges). It is very difficult to determine all the backlinks of a Web page, while it is relatively straightforward to determine its forward links. According to the PageRank algorithm, highly linked pages are more important (have greater authority) than pages with fewer links. However, not all backlinks are important. A backlink to a page from a credible source is more important than a link from some arbitrary page. Thus a page has a high rank if the sum of the ranks of its backlinks is high. PageRank was an attempt to see how good an approximation to the “importance” of a page can be obtained from the link structure.

$$P(A) = (1 - d) + d (P(T_1)/C(T_1) + \dots + P(T_n)/C(T_n))$$

Here T_1, T_2, \dots, T_n are the pages that point to Page A (that is, are citations to page A). PageRank forms a probability distribution over Web pages, so the sum of all Web pages' PageRanks is one.

The *HITS* Ranking Algorithm. The HITS algorithm proposed by Jon Kleinberg is another type of ranking algorithm exploiting the link structure of the Web. The algorithm presumes that a good hub is a document that points to many hubs, and a good authority is a document that is pointed at by many other authorities. The algorithm contains two main steps: a sampling component and a weight-propagation component. The sampling component constructs a focused collection S of pages with the following properties:

S is relatively small.

S is rich in relevant pages.

S contains most (or a majority) of the strongest authorities.

The weight component recursively calculates the hub and authority values for each document as follows:

Initialize hub and authority values for all pages in S by setting them to 1.

While (hub and authority values do not converge):

For each page in S , calculate authority value = Sum of hub values of all pages *pointing to* the current page.

For each page in S , calculate hub value = Sum of authority values of all pages *pointed at* by the current page.

Normalize hub and authority values such that sum of all hub values in S equals 1 and the sum of all authority values in S equals 1.

4. Web Content Analysis

As mentioned earlier, **Web content analysis** refers to the process of discovering useful information from Web content/data/documents. The **Web content data** consists of unstructured data such as free text from electronically stored documents, semi-structured data typically found as HTML documents with

embedded image data, and more structured data such as tabular data, and pages in HTML, XML, or other markup languages generated as output from databases. More generally, the term *Web content* refers to any real data in the Web page that is intended for the user accessing that page. This usually consists of but is not limited to text and graphics.

Structured Data Extraction. Structured data on the Web is often very important as it represents essential information, such as a structured table showing the airline flight schedule between two cities. There are several approaches to structured data extraction. One includes writing a **wrapper**, or a program that looks for different structural characteristics of the information on the page and extracts the right content. Another approach is to manually write an extraction program for each Website based on observed format patterns of the site, which is very labor intensive and time consuming. It does not scale to a large number of sites. A third approach is **wrapper induction** or **wrapper learning**, where the user first manually labels a set of training set pages, and the learning system generates rules—based on the learning pages—that are applied to extract target items from other Web pages. A fourth approach is the automatic approach, which aims to find patterns/grammars from the Web pages and then uses **wrapper generation** to produce a wrapper to extract data automatically.

Web Information Integration. The Web is immense and has millions of documents, authored by many different persons and organizations. Because of this, Web pages that contain similar information may have different syntax and different words that

describe the same concepts. This creates the need for integrating information from diverse Web pages. Two popular approaches for Web information integration are:

5. Approaches to Web Content Analysis

The two main approaches to Web content analysis are

(1) agent based (IR view) and

(2) database based (DB view).

The agent-based approach involves the development of sophisticated artificial intelligence systems that can act autonomously or semi-autonomously on behalf of a particular user, to discover and process Web-based information. Generally, the agent-based Web analysis systems can be placed into the following three categories:

Intelligent Web agents are software agents that search for relevant information using characteristics of a particular application domain (and possibly a user profile) to organize and interpret the discovered information. For example, an intelligent agent that retrieves product information from a variety of vendor sites using only general information about the product domain.

Information Filtering/Categorization is another technique that utilizes Web agents for categorizing Web documents. These Web agents use methods from information retrieval, and semantic information based on the links among various documents to organize documents into a concept hierarchy.

Personalized Web agents are another type of Web agents that utilize the personal preferences of users to organize search results, or to discover information and documents that could be of value for a particular user. User preferences could be learned from previous user choices, or from other individuals who are considered to have similar preferences to the user.

The database-based approach aims to infer the structure of the Website or to transform a Web site to organize it as a database so that better information management and querying on the Web become possible. This approach of Web content analysis primarily tries to model the data on the Web and integrate it so that more sophisticated queries than keyword-based search can be performed. These could be achieved by finding the schema of Web documents, building a Web document ware-house, a Web knowledge base, or a virtual database. The database-based approach may use a model such as the Object Exchange Model (OEM) that represents semi-structured data by a labeled graph. The data in the OEM is viewed as a graph, with objects as the vertices and labels on the edges. Each object is identified by an object identifier and a value that is either atomic—such as integer, string, GIF image, or HTML document—or complex in the form of a set of object references.

6. Web Usage Analysis

Web usage analysis is the application of data analysis techniques to discover usage patterns from Web data, in order to understand and better serve the needs of Web-based applications. This activity does not directly contribute to information retrieval; but it is important to improve or enhance the users' search experience. **Web usage data** describes the pattern of usage of Web pages, such as IP addresses, page references, and the date and time of accesses for a user, user group, or an application. Web usage analysis typically consists of three main phases: preprocessing, pattern discovery, and pattern analysis.

Preprocessing. Preprocessing converts the information collected about usage statistics and patterns into a form that can be utilized by the pattern discovery methods. We use the term “page view” to refer to pages viewed or visited by a user. There are several different types of preprocessing techniques available:

Usage preprocessing analyzes the available collected data about usage patterns of users, applications, and groups of users. Because this data is often incomplete, the process is difficult. Data cleaning techniques are necessary to eliminate the impact of irrelevant items in the analysis result. Frequently, usage data is identified by an IP address, and consists of clicking streams that are collected at the server. Better data is available if a usage tracking process is installed at the client site.

Content preprocessing is the process of converting text, image, scripts and other content into a form that can be used by the usage analysis. Often, this consists of performing content analysis such as classification or clustering. The clustering or classification techniques can group usage information for similar types of Web pages, so that usage patterns can be discovered for specific classes of Web pages that describe particular topics. Page views can also be classified according to their intended use, such as for sales or for discovery or for other uses.

Structure preprocessing: The structure preprocessing can be done by parsing and reformatting the information about hyperlinks and structure between viewed pages. One difficulty is that the site structure may be dynamic and may have to be constructed for each server session.

7. Practical Applications of Web Analysis

Web Analytics. The goal of **web analytics** is to understand and optimize the performance of Web usage. This requires collecting, analyzing, and performance monitoring of Internet usage data. On-site Web analytics measures the performance of a Website in a commercial context. This data is typically compared against key performance indicators to measure effectiveness or performance of the Website as a whole, and can be used to improve a Website or improve the marketing strategies.

Web Spamming. It has become increasingly important for companies and individuals to have their Websites/Web pages appear in the top search results. To achieve this, it is essential to understand search engine ranking algorithms and to present the information in one's page in such a way that the page is ranked high when the respective keywords are queried. There is a thin line separating legitimate page optimization for business purposes and spamming. **Web Spamming** is thus defined as a deliberate activity to promote one's page by manipulating the results returned by the search engines. Web analysis may be used to detect such pages and discard them from search results.

Web Security. Web analysis can be used to find interesting usage patterns of Websites. If any flaw in a Website has been exploited, it can be inferred using Web analysis thereby allowing the design of more robust Websites. For example, the backdoor or information leak of Web servers can be detected by using Web analysis techniques on some abnormal Web application log data. Security analysis techniques such as intrusion detection and denial of service attacks are based on Web access pattern analysis.

Web Crawlers. **Web crawlers** are programs that visit Web pages and create copies of all the visited pages so they can be processed by a search engine for indexing the downloaded pages to provide fast searches. Another use of crawlers is to automatically check and maintain the Websites. For example, the HTML code and the links in a Website can be checked and validated by the crawler. Another unfortunate use of crawlers is to collect e-mail addresses from Web pages, so they can be used for spam e-mails later.

5.mango operator

Insert

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

Syntax

The basic syntax of **insert()** command is as follows –

```
>db.COLLECTION_NAME.insert(document)
```

Example

```
> db.users.insert({
... _id : ObjectId("507f191e810c19729de860ea"),
... title: "MongoDB Overview",
... description: "MongoDB is no sql database",
... by: "tutorials point",
... url: "http://www.tutorialspoint.com",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
>
```

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows –

```
>db.COLLECTION_NAME.insertOne(document)
```

Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
> db.empDetails.insertOne(
    {
        First_Name: "Radhika",
        Last_Name: "Sharma",
        Date_Of_Birth: "1995-09-26",
        e_mail: "radhika_sharma.123@gmail.com",
        phone: "9848022338"
    })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

```
> db.empDetails.insertMany([
  {
    First_Name: "Radhika",
    Last_Name: "Sharma",
    Date_Of_Birth: "1995-09-26",
    e_mail: "radhika_sharma.123@gmail.com",
    phone: "9000012345"
  },
  {
    First_Name: "Rachel",
    Last_Name: "Christopher",
    Date_Of_Birth: "1990-02-16",
    e_mail: "Rachel_Christopher.123@gmail.com",
    phone: "9000054321"
  },
  {
    First_Name: "Fathima",
    Last_Name: "Sheik",
```



```

        Date_Of_Birth: "1990-02-16",
        e_mail: "Fathima_Sheik.123@gmail.com",
        phone: "9000054321"
    }
]
)
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5dd631f270fb13eec3963bed"),
        ObjectId("5dd631f270fb13eec3963bee"),
        ObjectId("5dd631f270fb13eec3963bef")
    ]
}
>

```

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows –

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA,
UPDATED_DATA)
```

Example

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}  
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}  
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB  
Tutorial'}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>db.mycol.find()  
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}  
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}  
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}  
>
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},  
  {$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

MongoDB updateOne() method

This methods updates a single document which matches the given filter.

Syntax

The basic syntax of updateOne() method is as follows –

```
>db.COLLECTION_NAME.updateOne(<filter>, <update>)
```

Example

```
> db.empDetails.updateOne(  
    {First_Name: 'Radhika'},  
    { $set: { Age: '30', e_mail: 'radhika_newemail@gmail.com' } }  
)  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }  
>
```

MongoDB updateMany() method

The updateMany() method updates all the documents that matches the given filter.

Syntax

The basic syntax of updateMany() method is as follows –

```
>db.COLLECTION_NAME.update(<filter>, <update>)
```

Example

```
> db.empDetails.updateMany(  
    {Age: { $gt: "25" } },  
    { $set: { Age: '00' } }  
)  
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

You can see the updated values if you retrieve the contents of the document using the find method as shown below –

```
> db.empDetails.find()
{ "_id" : ObjectId("5dd6636870fb13eec3963bf5"), "First_Name" : "Radhika",
"Last_Name"      :      "Sharma",      "Age"      :      "00",      "e_mail"      :
"radhika_newemail@gmail.com", "phone" : "9000012345" }
{ "_id" : ObjectId("5dd6636870fb13eec3963bf6"), "First_Name" : "Rachel",
"Last_Name"      :      "Christopher",      "Age"      :      "00",      "e_mail"      :
"Rachel_Christopher.123@gmail.com", "phone" : "9000054321" }
{ "_id" : ObjectId("5dd6636870fb13eec3963bf7"), "First_Name" : "Fathima",
"Last_Name"      :      "Sheik",      "Age"      :      "24",      "e_mail"      :
"Fathima_Sheik.123@gmail.com", "phone" : "9000054321" }
>
```

MongoDB - Delete Document

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. **remove()** method accepts two parameters. One is deletion criteria and second is **justOne** flag.

- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of **remove()** method is as follows –

```
>db.COLLECTION_NAME.remove(DELETION_CRITTERIA)
```

Example

Consider the mycol collection has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},  
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},  
{_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})  
WriteResult({"nRemoved" : 1})  
  
> db.mycol.find()  
  
{ "_id" : ObjectId("507f191e810c19729de860e2"), "title" : "NoSQL Overview" }  
{ "_id" : ObjectId("507f191e810c19729de860e3"), "title" : "Tutorials Point Overview" }
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
> db.mycol.remove({})  
WriteResult({ "nRemoved" : 2 })
```

```
> db.mycol.find()
```

```
>
```

MongoDB - Query Document

The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

Syntax

The basic syntax of **find()** method is as follows –

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-structured way.

Example

Assume we have created a collection named mycol as –

```
> use sampleDB
```

```
switched to db sampleDB
```

```
> db.createCollection("mycol")
```

```
{ "ok" : 1 }
```

```
>
```

And inserted 3 documents in it using the insert() method as shown below –

```
> db.mycol.insert([
```

```

{
  title: "MongoDB Overview",
  description: "MongoDB is no SQL database",
  by: "tutorials point",
  url: "http://www.tutorialspoint.com",
  tags: ["mongodb", "database", "NoSQL"],
  likes: 100
},
{
  title: "NoSQL Database",
  description: "NoSQL database doesn't have tables",
  by: "tutorials point",
  url: "http://www.tutorialspoint.com",
  tags: ["mongodb", "database", "NoSQL"],
  likes: 20,
  comments: [
    {
      user: "user1",
      message: "My first comment",
      dateCreated: new Date(2013,11,10,2,35),
      like: 0
    }
  ]
}
]

```

Following method retrieves all the documents in the collection –

```
> db.mycol.find()
{ "_id" : ObjectId("5dd4e2cc0821d3b44607534c"), "title" : "MongoDB
Overview", "description" : "MongoDB is no SQL database", "by" : "tutorials
point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database",
"NoSQL" ], "likes" : 100 }
{ "_id" : ObjectId("5dd4e2cc0821d3b44607534d"), "title" : "NoSQL Database",
"description" : "NoSQL database doesn't have tables", "by" : "tutorials point", "url"
: "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ],
"likes" : 20, "comments" : [ { "user" : "user1", "message" : "My first comment",
"dateCreated" : ISODate("2013-12-09T21:05:00Z"), "like" : 0 } ] }
>
```

The pretty() Method

To display the results in a formatted way, you can use pretty() method.

Syntax

```
>db.COLLECTION_NAME.find().pretty()
```

Example

Following example retrieves all the documents from the collection named mycol and arranges them in an easy-to-read format.

```
> db.mycol.find().pretty()
{
    "_id" : ObjectId("5dd4e2cc0821d3b44607534c"),
    "title" : "MongoDB Overview",
    "description" : "MongoDB is no SQL database",
```



```
"by" : "tutorials point",
"url" : "http://www.tutorialspoint.com",
"tags" : [
    "mongodb",
    "database",
    "NoSQL"
],
"likes" : 100
}
{
  "_id" : ObjectId("5dd4e2cc0821d3b44607534d"),
  "title" : "NoSQL Database",
  "description" : "NoSQL database doesn't have tables",
  "by" : "tutorials point",
  "url" : "http://www.tutorialspoint.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 20,
  "comments" : [
    {
      "user" : "user1",
      "message" : "My first comment",
      "dateCreated" : ISODate("2013-12-09T21:05:00Z"),
      "like" : 0
    }
  ]
}
```

```
}  
]  
}
```

The findOne() method

Apart from the find() method, there is **findOne()** method, that returns only one document.

Syntax

```
>db.COLLECTIONNAME.findOne()
```

Example

Following example retrieves the document with title MongoDB Overview.

```
> db.mycol.findOne({title: "MongoDB Overview"})  
{  
  "_id" : ObjectId("5dd6542170fb13eec3963bf0"),  
  "title" : "MongoDB Overview",  
  "description" : "MongoDB is no SQL database",  
  "by" : "tutorials point",  
  "url" : "http://www.tutorialspoint.com",  
  "tags" : [  
    "mongodb",  
    "database",  
    "NoSQL"  
  ],  
  "likes" : 100  
}
```

```
}
```

MongoDB - Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

Syntax

The basic syntax of **createIndex()** method is as follows().

```
>db.COLLECTION_NAME.createIndex({KEY:1})
```

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

```
>db.mycol.createIndex({"title":1})
```

```
{
```

```
"createdCollectionAutomatically" : false,  
"numIndexesBefore" : 1,  
"numIndexesAfter" : 2,  
"ok" : 1  
}  
>
```

In **createIndex()** method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.createIndex({"title":1,"description":-1})  
>
```

This method also accepts list of options (which are optional). Following is the list

—

Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is false .
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is false .

name	string	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is false .
expireAfterSeconds	integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
weights	document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
default_language	string	For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is English .
language_override	string	For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.

The dropIndex() method

You can drop a particular index using the dropIndex() method of MongoDB.

Syntax

The basic syntax of DropIndex() method is as follows().

```
>db.COLLECTION_NAME.dropIndex({KEY:1})
```

Here, "key" is the name of the file on which you want to remove an existing index. Instead of the index specification document (above syntax), you can also specify the name of the index directly as:

```
dropIndex("name_of_the_index")
```

Example

```
> db.mycol.dropIndex({"title":1})
{
  "ok" : 0,
  "errmsg" : "can't find index with key: { title: 1.0 }",
  "code" : 27,
  "codeName" : "IndexNotFound"
}
```

The dropIndexes() method

This method deletes multiple (specified) indexes on a collection.

Syntax

The basic syntax of DropIndexes() method is as follows() –

```
>db.COLLECTION_NAME.dropIndexes()
```

Example

Assume we have created 2 indexes in the named mycol collection as shown below

–

```
> db.mycol.createIndex({"title":1,"description":-1})
```

Following example removes the above created indexes of mycol –

```
>db.mycol.dropIndexes({"title":1,"description":-1})  
{ "nIndexesWas" : 2, "ok" : 1 }  
>
```

The getIndexes() method

This method returns the description of all the indexes in the collection.

Syntax

Following is the basic syntax of the getIndexes() method –

```
db.COLLECTION_NAME.getIndexes()
```

Example

Assume we have created 2 indexes in the named mycol collection as shown below

–

```
> db.mycol.createIndex({"title":1,"description":-1})
```

Following example retrieves all the indexes in the collection mycol –

```
> db.mycol.getIndexes()  
[  
  {  
    "v" : 2,
```

```
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.mycol"
    },
    {
        "v" : 2,
        "key" : {
            "title" : 1,
            "description" : -1
        },
        "name" : "title_1_description_-1",
        "ns" : "test.mycol"
    }
]
>
```

MongoDB - Replication

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers. Replication protects a database from the loss of a single server. Replication also allows you to recover from hardware

failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Why Replication?

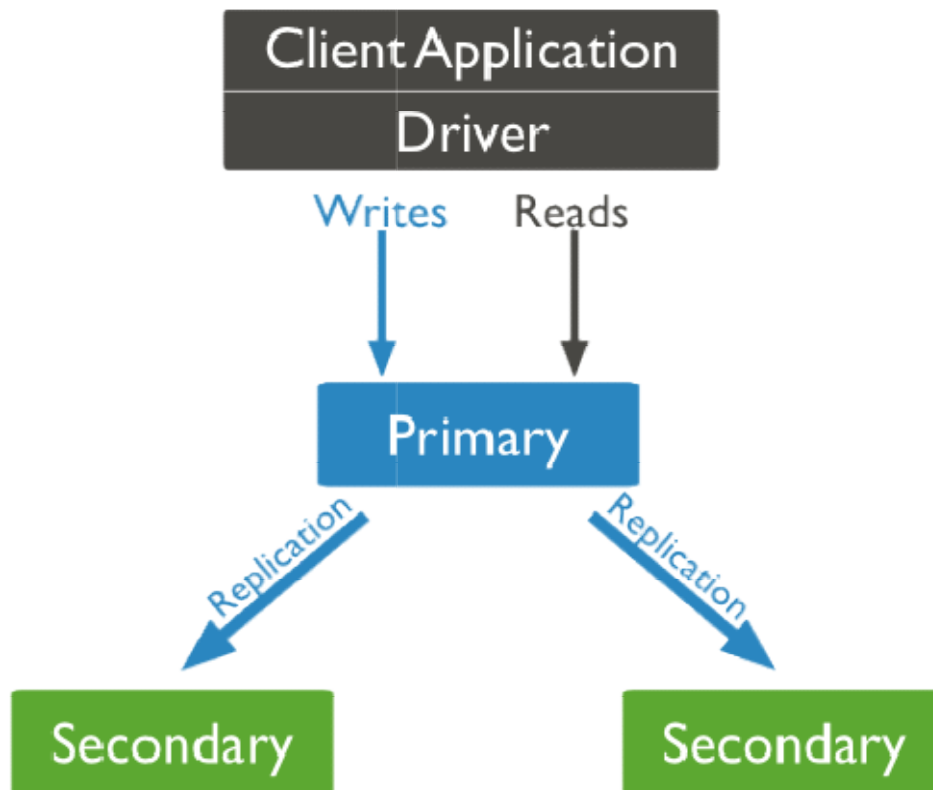
- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)
- Read scaling (extra copies to read from)
- Replica set is transparent to the application

How Replication Works in MongoDB

MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.
- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.

A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.



Replica Set Features

- A cluster of N nodes
- Any one node can be primary
- All write operations go to primary
- Automatic failover
- Automatic recovery
- Consensus election of primary

Set Up a Replica Set

In this tutorial, we will convert standalone MongoDB instance to a replica set. To convert to replica set, following are the steps –

- Shutdown already running MongoDB server.
-
- Start the MongoDB server by specifying -- replSet option. Following is the basic syntax of --replSet –

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```

Example

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

- It will start a mongod instance with the name rs0, on port 27017.
- Now start the command prompt and connect to this mongod instance.
- In Mongo client, issue the command **rs.initiate()** to initiate a new replica set.
- To check the replica set configuration, issue the command **rs.conf()**. To check the status of replica set issue the command **rs.status()**.

Add Members to Replica Set

To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command **rs.add()**.

Syntax

The basic syntax of **rs.add()** command is as follows –

```
>rs.add(HOST_NAME:PORT)
```

Example

Suppose your mongod instance name is **mongod1.net** and it is running on port **27017**. To add this instance to replica set, issue the command **rs.add()** in Mongo client.

```
>rs.add("mongod1.net:27017")  
>
```

You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not, issue the command **db.isMaster()** in mongo client.

MongoDB - Sharding

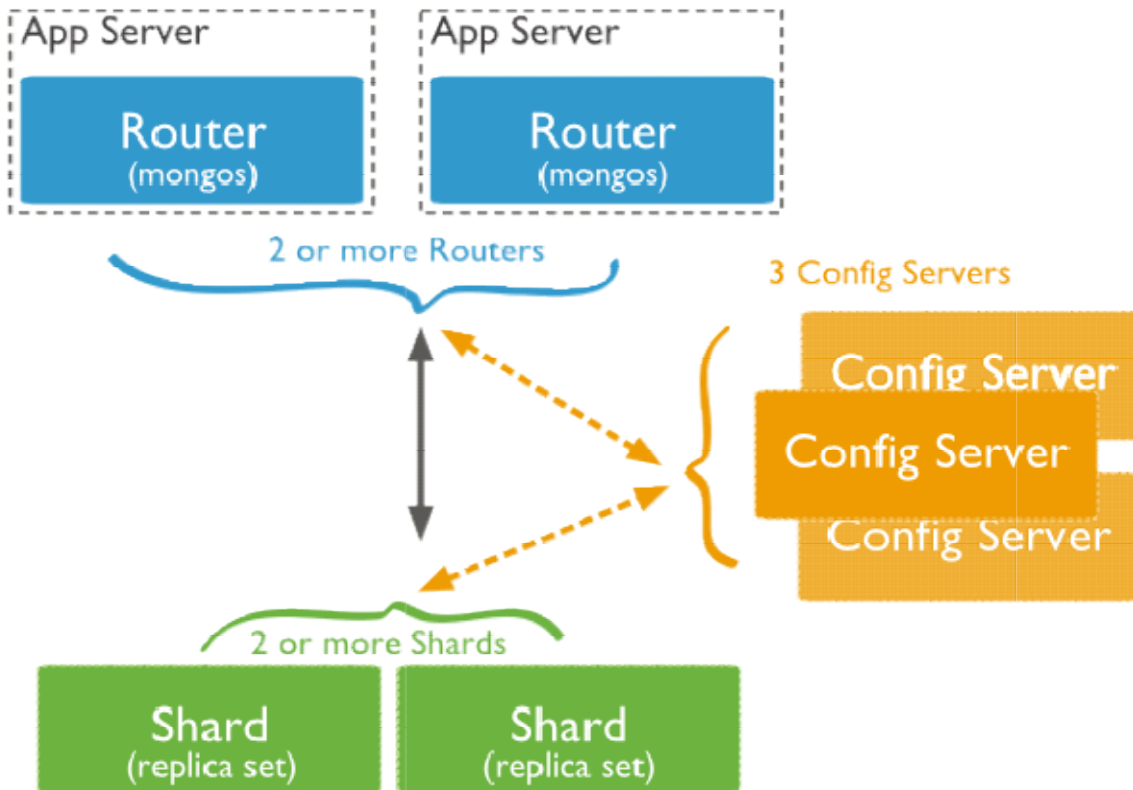
Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local disk is not big enough
- Vertical scaling is too expensive

Sharding in MongoDB

The following diagram shows the Sharding in MongoDB using sharded cluster.



In the following diagram, there are three main components –

- **Shards** – Shards are used to store data. They provide high availability and data consistency. In production environment, each shard is a separate replica set.
- **Config Servers** – Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.
- **Query Routers** – Query routers are basically mongo instances, interface with client applications and direct operations to the appropriate shard. The

query router processes and targets the operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally, a sharded cluster have many query routers.

MongoDB Applications

There are many MongoDB applications, and here they are:

1. Balanced Features

One can use MongoDB to get multiple balanced features. For example, that one wants to use some features like Queuing, Map/Reduce, FTS but don't require it a lot, which is easily possible through MongoDB.

2. Consistency over Availability

If one prefers consistency over availability, then he can get a specific version of Consistency in MongoDB applications.

3. Denormalizing the Data

Re-denormalizing the data is tough to do and also very expensive. Also, you will not be able to change the shard keys when you are running MongoDB.

The mix of Secondary indexes and Key/Value lookups

If you want to use a blend of secondary indexes and key/value lookups, then you can use MongoDB. But you cannot use it for too many secondary indexes because it will start scaling poorly.

4. Data on Single Server

One of the best features of MongoDB is that it was made intentionally sub-optimal to enable sharding on a single server. But you can use PostgreSQL, If the data fits on a single server comfortably.

5. Ideal for Querying

If the rate of querying is very strong to the database, then Mongo is ideal to be used because it resembles a DWH cube in its basic data structure.

6. Ideal for Documented-oriented

MongoDB is the right choice only when there are few relations, and one wants to scale it. It might not be suitable to use when there are too many relations, such as a social network. But it can be amazing to see as to how it will handle the document-oriented store. MongoDB is ideal for storing loads of documents that can be sorted by a tag or category. It stores records as documents in the form of compressed BSON files. We can directly retrieve these documents in the JSON format that can be easily read by humans. MongoDB, being a document-oriented database, enables storing both structured and unstructured data in the same document.

7. Polyglot Database System

MongoDB has an excellent capability to pick up the best part of all the databases, which makes it even more amazing to use as large-scale systems that are not using only a single database.

Conclusion

The demand for NoSQL databases like MongoDB has gone up in the recent times. Here are some real-world use cases of MongoDB along with some companies that

use MongoDB for their practical applications. We hope this article will help you understand MongoDB and its practical applications.

If you are curious to learn about big data, data science, check out IIIT-B & upGrad's [Executive PG Programme in Data Science](#) which is created for working professionals and offers 10+ case studies & projects, practical hands-on workshops, mentorship with industry experts, 1-on-1 with industry mentors, 400+ hours of learning and job assistance with top firms.