# Department of Information Technology
# Central University of Kashmir

**Tullmulla, Ganderbal, J&K-191131**
**www.cukashmir.ac.in**

# MTIT C 203:
# Advanced Database Management System

**UNIT 3**

## Sub Topic(s):

TRANSACTION – PROCESSING MONITORS, SERVICES, TP-MONITOR ARCHITECTURES, TP-MONITOR COMPONENTS, APPLICATION COORDINATION USING TP MONITOR, TRANSACTIONAL WORKFLOWS, FAILURE-ATOMICITY REQUIREMENTS, WORKFLOW MANAGEMENT SYSTEM ARCHITECTURES, LONG DURATION TRANSACTIONS, IMPACT OF CONCURRENCY PROTOCOLS, NESTED AND MULTILEVEL TRANSACTIONS, COMPENSATING TRANSACTIONS, REAL-TIME TRANSACTION SYSTEMS, HETEROGENEOUS DATABASES/ MULTI-DATABASES, UNIFIED VIEW OF DATA, QUERY PROCESSING.

**Course Title** **ADBMS**
**Course Code:** **MTIT C 203**
**Unit:** **3**
**Department:** **Department of IT**
**Year:** **2020**

**Compiled by: Aabiroo Bader**
**Email: baderabiro@gmai.com**
**Contact: 6005655573**
**Designation: Teaching Assistant**
**Department: Department of IT**

# Index

# Advanced Transaction Processing

## [1] TRANSACTION – PROCESSING MONITORS

TP Monitor – *Teleprocessing Monitor*

**Transaction-Processing monitors (TP monitors)** are the systems that were developed in the 1970s and 1980s, initially in response to a need to support a large number of remote terminals – *{Airline-Reservation Terminals}* from a single computer.

Provide infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.

TRANSACTION – PROCESSING MONITORS

### [2] SERVICES:

*Transaction-Processing Monitors provide services* such as:

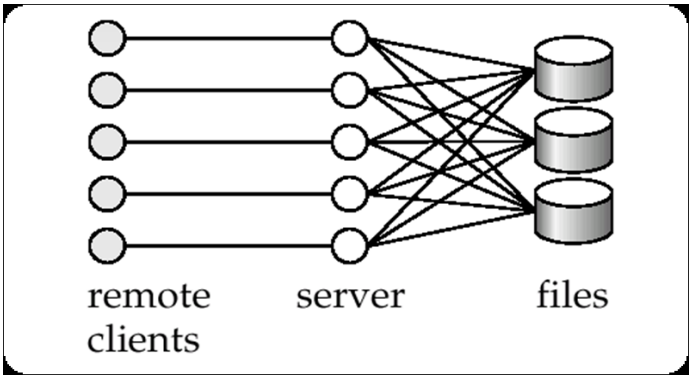
a) Presentation facilities to simplify creating user interfaces.
b) Persistent queuing of client requests and server responses.
c) Routing of client messages to servers.
d) Coordination of two-phase commit when transactions access multiple servers.

- Some commercial TP monitors: CICS from IBM, Pathway from Tandem, Top End from NCR, and Encina from Transarc.

## [3] TP-MONITOR ARCHITECTURES

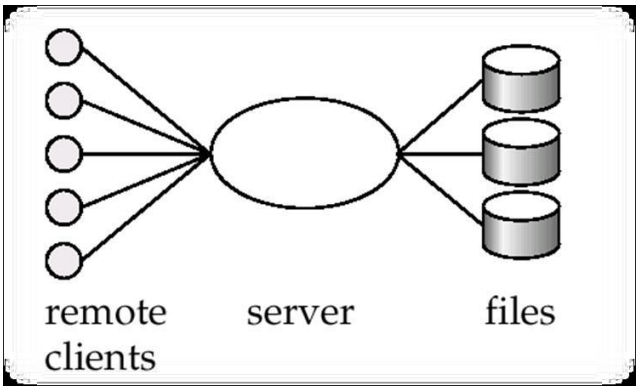
Large-scale transaction processing systems are built around client-server architecture and a number of different implementations exist in client-server architecture.

 ***Process-per-Client Model***



We have a Server Process for each client; the server performs the authentication and then executes actions requested by the client.
–Per-Process memory requirements are high.
–Operating system divides up available CPU time among processes by switching among them. –Overhead of Context Switching.

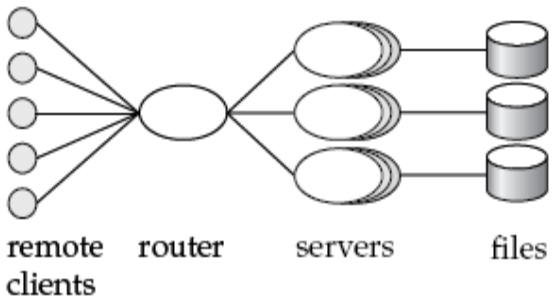
 ***Single Server Model***




We have a single-server process to which all remote clients connect.
• Remote Clients send requests to the server process which then executes those requests.
• The Server Process handles tasks – Authentication etc.
• To avoid blocking other clients when processing a long request for one client, the server process is multithreaded – Lesser overhead while switching between threads.

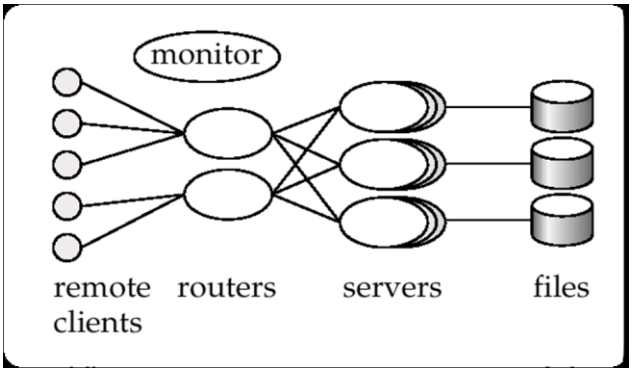
–No protection between threads.
–Not suited for parallel or distributed databases.

 **Many-Server, Single-Router Model**



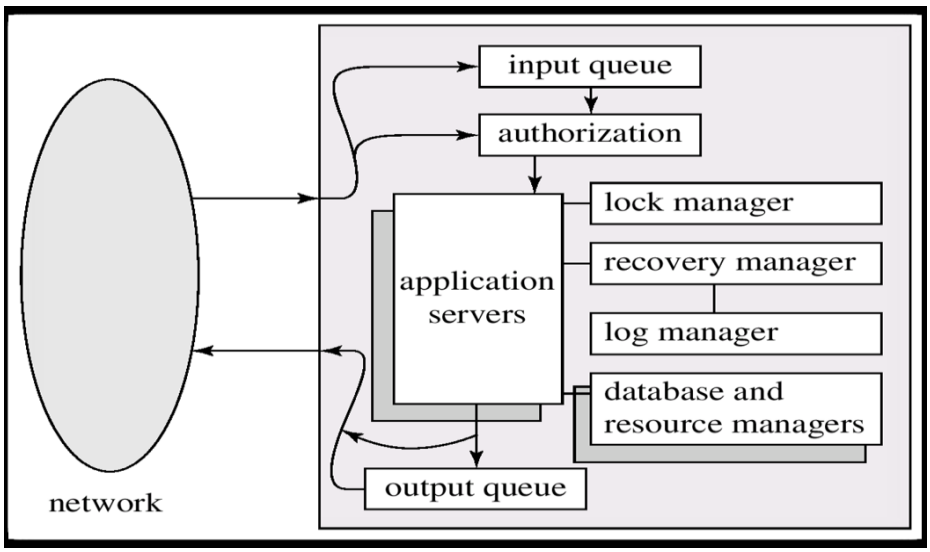

We have multiple application-server processes that access a common database and let the clients communicate with the application through a single communication process that routes requests.
• Independent server processes for multiple applications
• Multithread server process
• Application servers can run on different sites of a distributed database and communication process can handle the coordination among the processes.

**Many-Server, Many Router Model**




The client communication processes interact with one or more router processes, which route their requests to appropriate server.
•Controller process starts up and supervises other processes.
•Implemented in High Performance Web Servers.

## [4] TP-MONITOR COMPONENTS




•Queue manager handles incoming messages
•Some queue managers provide persistent or durable message queuing contents of queue are safe even if systems fail.
•Durable queuing of outgoing messages is important.

–application server writes message to durable queue as part of a transaction
–once the transaction commits, the TP monitor guarantees message is eventually delivered, regardless of crashes.

Many TP monitors provide locking, logging and recovery services, to enable application servers to implement ACID properties by themselves.

## [5] APPLICATION COORDINATION USING TP MONITORS

•Applications need to communicate
–**Multiple Databases**.
–**Legacy Systems** – Special Data Storage systems built directly on file systems.
–Finally the users or other applications at remote sites.
–In addition to this, they still need to interact network that facilitates the communication among various subsystems.

•It is very important to coordinate data accesses and implement ACID Properties.
•Modern TP managers provide such support and facilities for construction and administration of such large applications.
•TP monitor treats each subsystem as a **Resource Manager** – providing transactional access to some set of resources.
•Interface between TP monitor and Resource Manager is defined by set of transaction primitives:

***Begin_transaction***
***Commit_transaction***
***Abort_transaction***
***Prepare_to_commit_transaction***

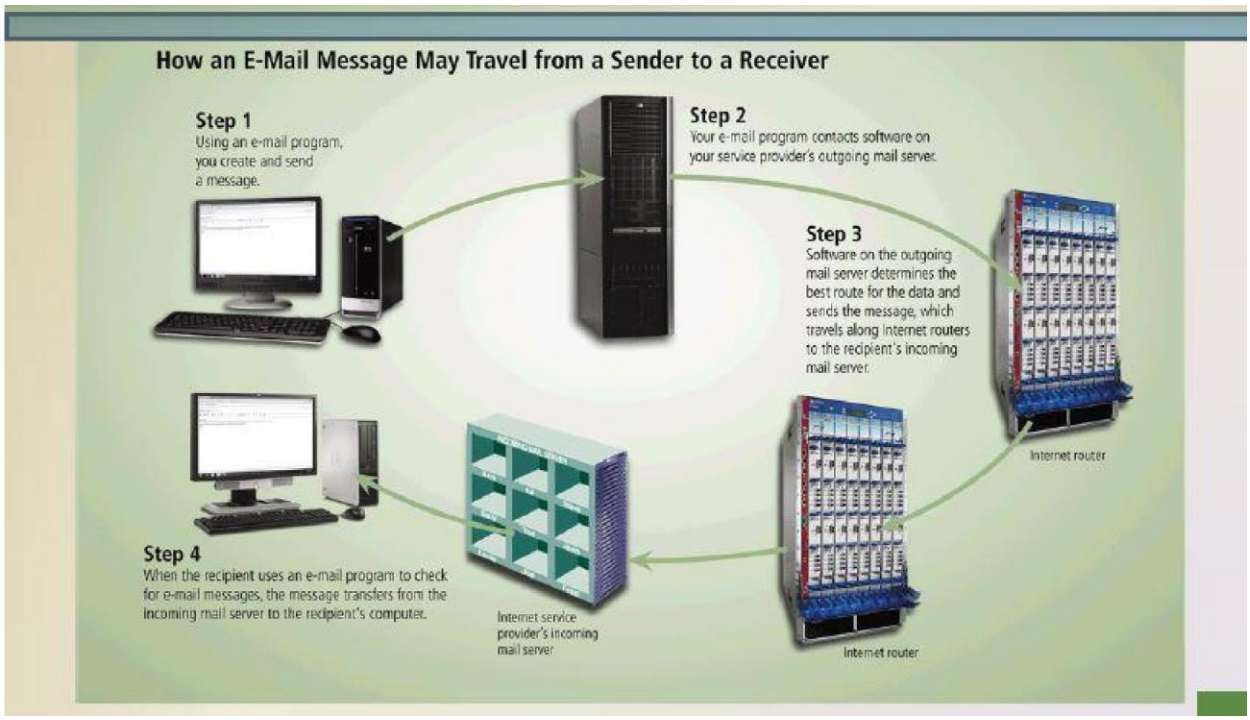
•***The resource manager must also provide other services*** **–** Supplying data to the application.
•The resource manager interface is defined by the X/Open Distributed Transaction Processing standard.
•TP monitor systems provide a transactional remote procedure call **(transactional RPC)** interface to their service

–Transactional RPC provides calls to enclose a series of RPC calls within a transaction.
–Updates performed by an RPC are carried out within the scope of the transaction, and can be rolled back if there is any failure.

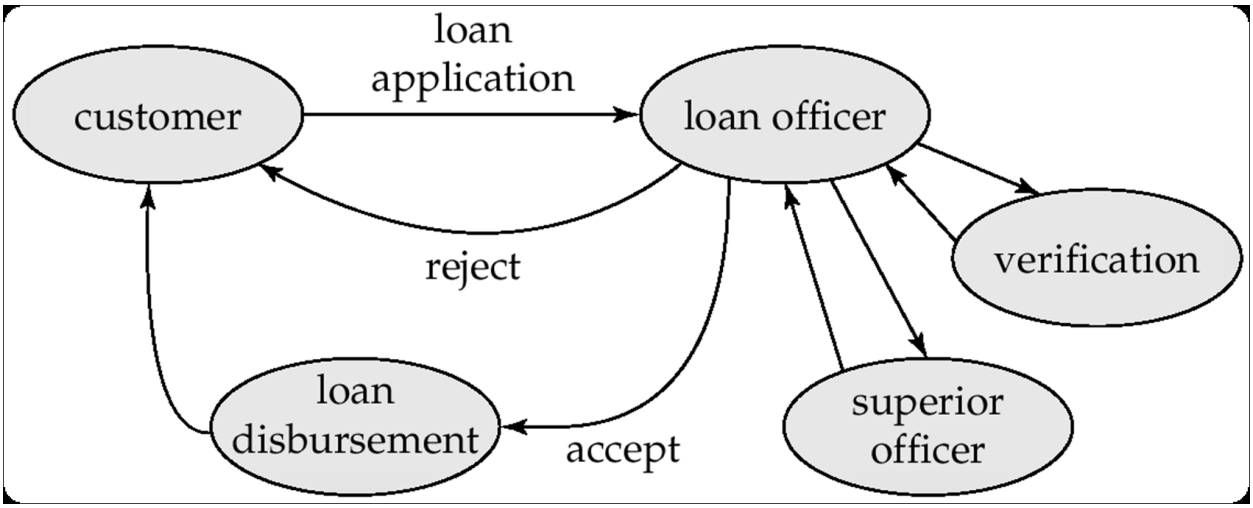
## [6] TRANSACTIONAL WORKFLOWS

•**Workflows** are activities that involve the coordinated execution of multiple tasks performed by different processing entities.
•With the growth of networks, and the existence of multiple autonomous database systems, workflows provide a convenient way of carrying out tasks that involve multiple systems.
Example of a workflow delivery of an email message, which goes through several mails systems to reach destination.
–Each mailer performs a tasks: forwarding of the mail to the next mailer.
–If a mailer cannot deliver mail, failure must be handled semantically (delivery failure message).
Workflows usually involve humans: e.g. loan processing, or purchase order processing.

**EXAMPLES OF WORKFLOWS**




**LOAN PROCESSING WORKFLOW**




In the past, workflows were handled by creating and forwarding paper forms. Computerized workflows aim to automate many of the tasks. But the humans still play role e.g. in approving loans. To automate the tasks involved in Loan Processing – We can store the Loan application and associated information in a database. Workflow itself then involves handling responsibility from one human to next. Workflows are very important in organizations and organizations today have multiple software systems that need to work together.

–Employee joins an organization and information about the employee is provided to

•Payroll System.

•Library System.
•Authentication Systems.

We need to address two activities in general to automate a workflow:

–**Workflow Specification**.
–**Workflow Execution.**

Both activities are complicated by the fact that many organizations use several independently managed information processing systems, that in most cases were developed separately to automate different functions.

Workflow activities may require interactions among several such systems each performing a task as well as interactions with humans.

## [7] FAILURE-ATOMICITY REQUIREMENTS

Traditional notion of ***'failure atomicity'*** requires that failure of any task result in failure of workflow.

*Designer of workflow is allowed to define failure atomicity requirements of workflow. Even if we may have a sub-transaction failing at any point of time, a workflow designer may execute a functionally equivalent task so that final transaction can commit.*

System must guarantee that every transaction must terminate in a state that satisfies the failure atomicity requirement defined by the designer.

•**Acceptable Termination State** :

–***Committed Acceptable Termination State****: Objectives achieved successfully.*
–***Aborted Acceptable Termination State****: Workflow fails to achieve objectives.*

•**Unacceptable Termination State**

–All other states fit in unacceptable termination states.

## [8] WORKFLOW MANAGEMENT SYSTEM ARCHITECTURES

Execution of workflow tasks may be controlled by ***Human Coordinator*** or by a ***Software system known as Workflow Management System.***

•Workflow Management System consists of:

- ***Scheduler***: is a program that processes workflows by submitting various tasks for execution, monitoring various events and evaluating conditions related to inter-task dependencies.
- ***Task Agents***: controls the execution of a task by a processing entity.
- ***Query Mechanism*:** A query mechanism to query the state of the workflow.

•We have three architectural approaches for development of a workflow management system:

- **Centralized.**
- **Partially distributed.**
- **Fully distributed**.

a) Centralized - a single scheduler schedules the tasks for all concurrently executing workflows.

–used in workflow systems where the data is stored in a central database.
–easier to keep track of the state of a workflow.

b) Partially distributed - has one (instance of a) scheduler for each workflow.

c) Fully distributed - has no scheduler, but the task agents coordinate their execution by communicating with each other to satisfy task dependencies and other workflow execution requirements.

–used in simplest workflow execution systems
–based on electronic mail.

## [9] LONG DURATION TRANSACTIONS

Depending on the lifetime or duration, transactions can be classified as:

–**Short Duration Transaction**: *Short duration transaction is also known as online transaction requiring very short execution/response time and access small portion of the database.*

–**Long Duration Transaction:** *A long duration transaction also known as Batch transaction requires a longer execution/response time and generally accesses larger portion of the database*.

Alternately we can define long duration in context of database systems as the on that involve human intervention, while short duration transactions are more or less Non-Interactive.

•The long-duration transactions exhibit following properties:

a) **Long Duration**: *When interacting with Humans, it is quite natural the response will be very slow relative to computer speed.*

- In some applications, the human activity may involve hours, days, even longer periods of time.
- So overall the transactions will be of longer duration.

b) **Exposure to Uncommitted Data:** *In many cases of long duration transactions, the other transactions may be forced to read uncommitted data.*

- If several users are cooperating on a project, user transactions may need to exchange data prior to transaction commit**.**

c) **Subtasks:** *Interactive transaction will consist of set of subtasks initiated by the user.*

- At some point of time, user may wish to abort a subtask, without necessarily causing the entire transaction to Abort.

d) **Recoverability:** *It is un-acceptable to abort a Long-duration interactive transaction given a system crash.*

- Even if we have a system crash, the active transaction must be recovered to a state that existed shortly before the crash, so that relatively human work is lost.

e) **Performance:** Good performance in Long Duration Transaction is defined as how fast the response has been generated for a particular transaction.

- However, in case of non-interactive system, performance is measured as *HIGH THROUGHPUT*.

## [10] IMPACT OF CONCURRENCY PROTOCOLS

Given the properties of Long Duration Transactions, various Concurrency protocols are very difficult to implement:

➢ **2 Phase Locks:** If a lock *isn't granted, the transaction requesting the lock is forced to wait for the data item to be unlocked.*

- *If the transaction holding the lock on data item happens to be a Long Duration Transaction, Response Time increases leading to increased chances of Deadlock.*

➢ **Time-Stamp-Based Protocols:** *Although timestamp based protocols never a transaction to wait; however they require transaction to abort under certain circumstances.*

- *If a long duration transaction is aborted a substantial amount of work is lost.*

➢ **Validation Protocols**: With the enforcement of Serializibility, the result is a **Long Wait** or **Abortion of Long Duration Transactions** or **Both.**

- Further difficulties arise, with the enforcement of serializibility, when considering Recovery Issues.
- A case of **CASCADING ROLLBACK** finally has undesirable effects on Long Duration Transaction.

**Snapshot Isolation** is a solution for achieving Concurrency when dealing with Long Duration Transactions.

## [11] NESTED AND MULTILEVEL TRANSACTIONS

A **long duration transaction**, can be viewed as a collection of related Subtasks or Sub-transactions. By structuring the Long Duration transaction as a set of Sub-transactions, we are able to run several sub-transactions in parallel, provided parallelism doesn‟t lead to conflicts. When Long duration transactions are modelled as a set of sub-transactions, a system crash or a failure doesn‟t necessarily mean roll back of entire Long Duration Transaction.

•Suppose some transaction ti in T may abort; it doesn‟t mean we will abort T, instead T will simply restart ti.
• Suppose transaction ti commits, this commit isn‟t permanent, ti commits to T. If T aborts ti will need to abort.
•A nested or multilevel Transaction **T** = {*t*1, *t*2, ..., *tn*} of sub-transactions and a partial order *P* on *T*.

Execution of T mustn‟t violate the partial order.

- Multi-level Transaction is a T in which a sub-transaction is allowed to release locks on completion.
- Nested Transaction is a T, in which a sub-transaction *t*i holding the locks, upon completion of *t*i, Locks are automatically assigned to T.

## [12] COMPENSATING TRANSACTIONS

Long Duration Transactions generally suffer from *long waiting time*. To reduce the waiting times, we expose the uncommitted updates to other concurrently executing transactions.

This exposure of uncommitted updates to transactions have serious issues – Cascading Rollback.

- To overcome the issues, concept of Compensating Transactions were implemented.
- We have a Long Duration Transaction „T‟, divided into several Sub-Transactions *t*1, *t*2, ..., *tn.*
- If outer level sub-transaction of T commits, it releases its locks.
- If the outer level sub-transaction ti of transaction T has aborted effect of its sub-transactions must be undone.

Suppose that sub-transactions *t*1, *t*2, ..., *tk have committed and tk+1 was executing when decision to abort was made.*
*Since tk+1 has aborted we simply undo the effects of that sub transaction.*
Instead of undoing all changes made by the failed transaction, action is taken to “compensate” for the failure.

- Consider a long-duration transaction *Ti* representing a travel reservation, with sub-transactions *Ti*,1, which makes airline reservations, *Ti*,2 which reserves rental cars, and *Ti*,3 which reserves a hotel room.
- Hotel cancels the reservation.
- Instead of undoing all of *Ti*, the failure of *Ti*,3 is compensated for by deleting the old hotel reservation and making a new one.

## [13] REAL-TIME TRANSACTION SYSTEMS

In systems with real-time constraints, correctness of execution involves both database consistency and the satisfaction of deadlines.

A. **Hard deadline** – Serious problems may occur if task is not completed within deadline
B. **Firm deadline** - The task has zero value if it completed after the deadline.
C. **Soft deadline** - The task has diminishing value if it is completed after the deadline.

The wide variance of execution times for read and write operations on disks complicates the transaction management problem for time-constrained systems
–main-memory databases are thus often used
–Waits for locks, transaction aborts, contention for resources remain as problems even if data is in main memory

Design of a real-time system involves ensuring that enough processing power exists to meet deadline without requiring excessive hardware resources.

## [14] HETEROGENEOUS DATABASES/ MULTI-DATABASES

Many new database applications require data from a variety of pre-existing databases located in a heterogeneous collection of Hardware and Software.

Manipulation of Information located in Heterogeneous Distributed Database requires an additional software layer on top of existing database systems known as *Multi-database System.*

- Some Characteristics of Multi-database Systems are:
- Different sites use dissimilar schemas and software.
- System may be composed of a variety of DBMS like Relational, Network, Hierarchical or object oriented.

## [15] UNIFIED VIEW OF DATA

Each local database management system may use a different data model. Some may use Relational Model while others may use Network Model or Hierarchical Model.

Since Multi-database system is supposed to provide the illusion of a single integrated database system, common data model must be used.

Each Local System provides its own Conceptual Schema. The multi-database system integrate these separate schemas into a common schema. This Schema integration is a complicated task due to semantic integrity.

- Schema integration isn‟t simple straight forward translation between data definition languages, same attributes may appear with different meanings in different Local Databases.
- Integer value of Length may be in Inches in one system and millimeters in another.

## [16] QUERY PROCESSING

Query Processing in Heterogeneous databases can be complicated. Some of the issues are:

- Given a Query on Global Schema, the query may be translated into queries on Local Schemas at each sites where the query has to be executed. The query results have to be translated back into Global Schema.
- We have **Wrappers** for each data source.

Wrappers provide view of data in global schema and also translate queries on global schema into queries on the local schema and translate results back into global schema.

Wrappers can even be used to provide Relational view of Non-relational data sources.

- Some data sources may provide only limited query capabilities
- They may support Selections, but not Joins or they may even support Selections on only certain fields. So queries may be broken up to be partly performed at the data source and partly at site issuing the query.
- Global Query Optimization in Heterogeneous Database is difficult, since the query execution systems may not know alternative query plans at different sites.

- Mediator systems are the systems that integrate multiple heterogeneous data sources, providing an integrated global view of the data and providing query facilities on the global view.
- Mediator Systems are also known as Multi-databases, since they provide the appearance of a single database with a global schema, although data exist in multiple sites in local schemas.

## References & Bibliography

- Silberschatz−Korth−Sudarshan, "Database System Concepts", The McGraw−Hill Companies.
- Ramez Elmasri, Shamkant B. Navathe, "Fundamental OF Database Systems", Pearson.