New Syllabus
MU

Strictly as per the New Revised Syllabus (REV- 2019 'C' Scheme)
of Mumbai University w.e.f. academic year 2021-22

Advance Database Management System

(Code : CSDLO5013) (Department level optional course -I)

Semester V - Computer Engineering

FIRST Video **DBMS** Textbook in a "Easy to Understand Way" *

Mahesh Mali
Vaibhav Vasani

Includes :

- *Solved Latest University Question Papers.*

TechKnowledge™ Publications

Advance Database Management System

(Department Level Optional Course - I) (Code : CSDLO5013)

Semester V : Computer Engineering (Mumbai University)

Strictly as per the New Revised Syllabus (2019 Course) of Mumbai University w.e.f. academic year 2021-2022

Prof. Mahesh Mali

Ph.D. (Computer Engineering) (Pursuing),
M.E. (Computer Engineering), B.E. (Information Technology),
Oracle Certified PL/SQL Developer Associate (OCA),
SAS Certified Data Analyst.
Mumbai, Maharashtra, India.

Prof. Vaibhav Prakash Vasani

Assistant Professor
Department of Computer Engineering
K. J. Somaiya College of Engineering,
Somaiya University, Vidyavihar,
Mumbai, Maharashtra, India.

Scan the QR Code to access the Videos by Author

TechKnowledge™ Publications

MO178A Price ₹ 175/-

Advance Database Management System (Code : CSDLO5013)

Prof. Mahesh Mali, Prof. Vaibhav Prakash Vasani

(Semester V – Computer Engineering, (Mumbai University))

First Printed in India Edition : *April 2010 (For Mumbai University)*

First Edition : *August 2021*

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

ISBN : *978-93-91496-04-3*

Published By

TECHKNOWLEDGE PUBLICATIONS

Printed @

37/2, Ashtavinayak Industrial Estate,
Near Pari Company,
Narhe, Pune, Maharashtra State, India.
Pune - 411041

Head Office

B/5, First floor, Maniratna Complex, Taware Colony,
Aranyeshwar Corner, Pune - 411 009.
Maharashtra State, India

Ph : 91-20-24221234, 91-20-24225678.

Email : info@techknowledgebooks.com,

Website : www.techknowledgebooks.com

Subject Code : CSDLO5013
Book Code : MO178A

We dedicate this Publication soulfully and wholeheartedly,
in loving memory of our beloved founder director,
Late Shri. Pradeepji Lalchandji Lunawat, who will
always be an inspiration, a positive force & strong support behind us.

"My work is my prayer to God"

– Lt. Shri. Pradeepji L. Lunawat

Soulful Tribute and Gratitude for all Your Sacrifices, Hardwork and 40 years of Strong Vision...

Preface

My Dear Students,

*We are extremely happy to come out with this book on "**Advance Database Management System** " for you. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.*

*We present this book in the loving memory of **Late Shri. Pradeepji Lunawat,** our source of inspiration and a strong foundation of **"TechKnowledge Publications".** He will always be remembered in our heart and motivate us to achieve our milestone.*

We are thankful to Mr. Shital Bhandari, Shri. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they have extended. We also thankful to Seema Lunawat for technology enhanced reading, E-books support and the staff members of TechKnowledge Publications for their efforts to make this book as good as it is.

We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let me know, because that will help me to improve further.

- Prof. Mahesh Mali

- Vaibhav Prakash Vasani

❑❑❑

SYLLABUS

Advance Database Management System (CSDLO5013)

Prerequisite : Database Management System

Course Objectives :

1. *To provide insights into distributed database designing*
2. *To specify the various approaches used for using XML and JSON technologies.*
3. *To apply the concepts behind the various types of NoSQL databases and utilize it for Mongodb*
4. *To learn about the trends in advance databases*

Course Outcomes :

After the successful completion of this course learner will be able to :

1. *Design distributed database using the various techniques for query processing*
2. *Measure query cost and perform distributed transaction managemen*
3. *Organize the data using XML and JSON database for better interoperability*
4. *Compare different types of NoSQL databases*
5. *Formulate NoSQL queries using Mongodb*
6. *Describe various trends in advance databases through temporal, graph based and spatial based databases*

COURSE CONTENTS

Unit I : Distributed Databases **03 hrs**

Introduction, Distributed DBMS Architecture, Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design.

Unit II : Distributed Database Handling **08 hrs**

Distributed Transaction Management – Definition, properties, types, architecture

Distributed Query Processing - Characterization of Query Processors, Layers/ phases of query processing.

Distributed Concurrency Control- Taxonomy, Locking based, Basic TO algorithm,

Recovery in Distributed Databases: Failures in distributed database, 2PC and 3PC protocol.

Unit III : Data interoperability – XML and JSON **06 hrs**

XML Databases: Document Type Definition, XML Schema, Querying and Transformation: XPath and XQuery.

Basic JSON syntax, (Java Script Object Notation),JSON data types, Stringifying and parsing the JSON for sending & receiving, JSON Object retrieval using key-value pair and JQuery, XML Vs JSON

Unit IV : NoSQL Distribution Model **10 hrs**

NoSQL database concepts: NoSQL data modeling, Benefits of NoSQL, comparison between SQL and NoSQL database system.

Replication and sharding, Distribution Models Consistency in distributed data, CAP theorem, Notion of ACID Vs BASE, handling Transactions, consistency and eventual consistency

Types of NoSQL databases: Key-value data store, Document database and Column Family Data store, Comparison of NoSQL databases w.r.t CAP theorem and ACID properties.

Unit V : NoSQL using MongoDB **06 hrs**

NoSQL using MongoDB : Introduction to MongoDB Shell, Running the MongoDB shell, MongoDB client, Basic operations with MongoDB shell, Basic Data Types, Arrays, Embedded Documents

Querying MongoDB using find() functions, advanced queries using logical operators and sorting, simple aggregate functions, saving and updating document.

MongoDB Distributed environment : Concepts of replication and horizonal scaling through sharding in MongoDB

Unit VI : Trends in advance databases **06 hrs**

Temporal database : Concepts, time representation, time dimension, incorporating time in relational databases.

Graph Database : Introduction, Features, Transactions, consistency, Availability, Querying, Case Study Neo4J

Spatial database : Introduction, data types, models, operators and queries

□□□

Unit I

Introduction, Distributed DBMS Architecture, Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design.

Unit II

Distributed Transaction Management : Definition, properties, types, architecture

Distributed Query Processing : Characterization of Query Processors, Layers/ phases of query processing.

Distributed Concurrency Control : Taxonomy, Locking based, Basic TO algorithm,

Recovery in Distributed Databases : Failures in distributed database, 2PC and 3PC protocol.

Unit III

XML Databases: Document Type Definition, XML Schema, Querying and Transformation: XPath and XQuery.

Basic JSON syntax, (Java Script Object Notation),JSON data types, Stringifying and parsing the JSON for sending & receiving, JSON Object retrieval using key-value pair and JQuery, XML Vs JSON

Unit IV

NoSQL database concepts: NoSQL data modeling, Benefits of NoSQL, comparison between SQL and NoSQL database system.

Replication and sharding, Distribution Models Consistency in distributed data, CAP theorem, Notion of ACID Vs BASE, handling Transactions, consistency and eventual consistency

Types of NoSQL databases: Key-value data store, Document database and Column Family Data store, Comparison of NoSQL databases w.r.t CAP theorem and ACID properties.

Unit V

NoSQL using MongoDB: Introduction to MongoDB Shell, Running the MongoDB shell, MongoDB client, Basic operations with MongoDB shell, Basic Data Types, Arrays, Embedded Documents

Querying MongoDB using find() functions, advanced queries using logical operators and sorting, simple aggregate functions, saving and updating document.

MongoDB Distributed environment: Concepts of replication and horizonal scaling through sharding in MongoDB

Unit VI

Temporal database : Concepts, time representation, time dimension, incorporating time in relational databases.

Graph Database: Introduction, Features, Transactions, consistency, Availability, Querying, Case Study Neo4J

Spatial database: Introduction, data types, models, operators and queries

1 Distributed Database

Syllabus

Introduction, Distributed DBMS Architecture

1.1 Distributed Databases System Concepts

- Data in a distributed database system is stored across various sites in which every site is managed by its own DBMS that can run independently on that site.
- A distributed database is a collection of many logically related databases distributed over a computer network.
- A distributed database management system (DDBMS) manages a distributed database.
- It is used for organizational decentralization which required for multiple branches and also offers economical processing at greater speed.
- The main aim of distributed database system is to introduce all advantages of distributed computing system in DBMS.
- In Distributed computing each site may have own memory and own Database server which operates a single site. This type of architecture is also called as shared nothing architecture.
- Distributed databases are distributed over multiple sites.
- In case of distributed computing we may build a network architecture which has centralized server which has data of all other sites. Hence, this central site is treated as main storage server which is generally mainframe.
- E.g. : ABC Bank may have Central Network File System (CNFS) which stores all data of Local Network File System (LNFS) i.e. local site of Mumbai, Pune etc.

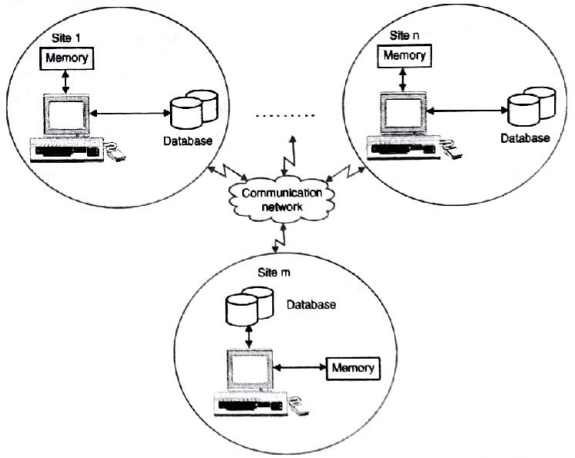


Fig. 1.1.1 : Shared nothing architecture of distributed database

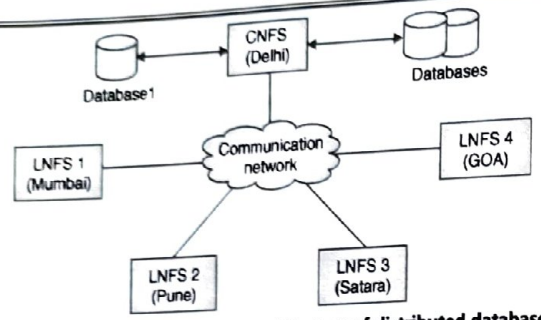


Fig. 1.1.2 : Centralized database architecture of distributed database

- A true distributed system have own database at each of site and communicate with each other through communication network.

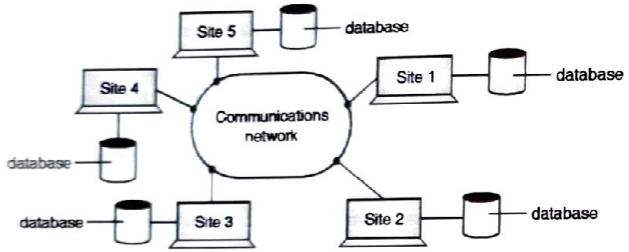


Fig. 1.1.3 : Distributed database system

1.2 Concept of Distributed Computing System

- Complex problems can be solved efficiently by partitioning it into smaller simple fragments and then solve it independently on different sites.
- As we are using multiple computers for solving a complex problem hence more computing power is generated at comparatively lower cost.
- Individual processing elements are autonomous and can be managed independently.
- As a result of distributed technology reliability and scalability of system is increased.
- User in this system gets feel that he is working in a single centralized database system.
- The main goal of distributed databases is to bring all advantages of distributed computing into distributed database system.

1.3 Advantages of Distributed Databases

In phase of traditional single server database system we can make use of DDBMS so that, complex problems can be solved efficiently by partitioning it into smaller simple fragments and then solve it independently on different sites. It also offers some advantages as below,

(a) Better storage ability using data fragmentation and replication.

(b) DDBMS offers better Query solving capability access to remote sites using high speed communication networks.

(c) DDBMS has potential to access data from various servers in order to offer **better Transaction management.**

(d) DDBMS keeps multiple replicas of same data to offer good database **Recovery.**

(e) Secure data environment is offered due to authorization and access privilege of multiple data servers.

(f) Directory : DDBMS manages metadata about all data items on database servers.

(g) Data Replication takes care about management of multiple consistent copies of same data items.

1.4 Features of Distributed Database System

Distributed database management has been introduced for various reasons ranging like organizational decentralization or data processing at lower cost.

1. Management of distributed data with different levels of transparency.

A DBMS should hide the detail of where each data item (like tables or relations) is physically stored within the system.

(a) Distribution / Network Transparency

- In this all internal network operations are hidden from the user.
- Network may be divided into location transparency and naming transparency.

(b) Fragmentation transparency

- The process of decomposing the database into smaller multiple units called as fragmentation.
- Fragmentation transparency makes the user unaware of the existence of data fragments.

(c) Replication transparency

- Replication is coping data at multiple sites and at multiple locations for better availability, performance and reliability.
- Replication transparency makes the user unaware of the existence of copies of data.

2. Increase reliability

- Reliability is defined as the probability that a system is running (working) at any point of time.
- In case of distributed database systems, if one system fails to work or down for some time other system can take over its all functions hence overall system does not affected.

3. Increase availability

- Availability is defined as the probability that the system is continuously available (accessible) during a certain time interval for any database operations.
- In case of distributed database systems, if one system is not available (due to failure) for some time then other system can take over its all functions hence system is always available.

4. Improved performance

- A distributed DBMS fragments the database and try to keep data closer to the site where it is needed most of the time for operations.
- Data localization reduces the conflict for CPU and I/O services and simultaneously reduces access delays involved in wide area networks.
- As a large database is distributed over multiple sites. Hence, smaller databases exist at each site which is simple to handle and maintain.

- Therefore, local queries and transactions accessing data at a single site have better performance because of the smaller size of local databases.
- Interquery and intraquery parallelism can be achieved efficiently by executing multiple queries at different sites, or by breaking up a query into a number of small sub queries that execute in parallel to give faster results.

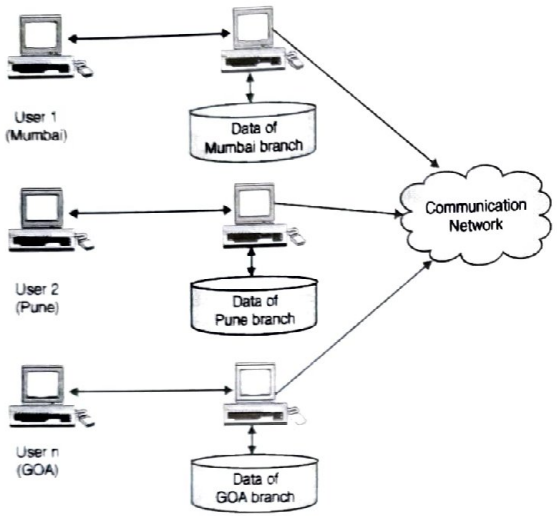


Fig. 1.4.1 : Data localization (increases performance of system)

5. Ease for expansion

Expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier.

1.5 Design Issues of Distributed Database System

Q. Design issues of distributed database. **MU - May 15, May 16, 5 Marks**

In order to understand challenges need to be overcome in design process of DDB.

(a) Distributed database design

The data is distributed across multiple sites so accessing them across multiple sites can be a big challenge. This leads to Data Fragmentation and Replication which are issues for distributed database design

(b) Distributed directory management

A directory contains information about data distribution in the database. This problem may be related with database placement problem.

A directory can be located on local or it may be located on global schema; it can be even centralized at single site or distributed over several sites; there can be a single copy or several copies of data.

(c) Distributed query processing

As data is distributed across sites hence processing query involves analyzing queries and convert them into a series of data queries with less cost. The cost can be communication costs.

(d) Distributed concurrency control

Concurrency control involves the management of multiple transactions which accesses distributed data, such that the database integrity should be maintained.

(e) Distributed deadlock management

The Distributed deadlock problem is similar to that of operating systems deadlocks. As there may be competition between database users for access to a data which can result in a deadlock.

(f) Reliability of distributed DBMS

It is one of the main advantages of DDB. These mechanisms should also ensure the consistency of the distributed database and detect failures and recover from them.

(g) Replication

If the distributed database sometimes data is partially or fully replicated for improving availability. This may introduce lot of challenges to manage replication without affecting data access.

1.6 Types of Distributed Databases

1. Homogeneous distributed database system

- If all data servers to which data is distributed are having same DBMS software then this system is called as a homogeneous distributed database system.
- These systems are easy to handle good performance and good data access speed.

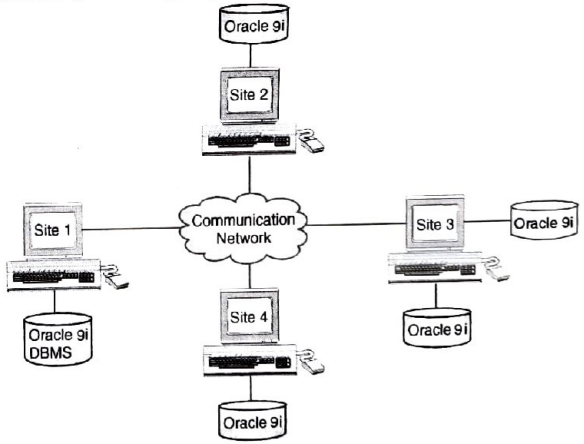


Fig. 1.6.1 : Homogeneous distributed database system

2. Heterogeneous distributed database system

- If different database servers are running under the control of different type of DBMS systems and are connected to enable access to data from different sites, such system is called as a heterogeneous distributed database system, also referred to as a multi database system.
- For constructing heterogeneous systems we have to use all standards for gateway protocols.

- A gateway protocol is an API that used when we exposes DBMS functionality to all other external applications.
- Examples connections using ODBC and JDBC are accessing database servers through gateway protocols.
- System comes at economic cost in terms of performance, software complexity, and administration difficulty.

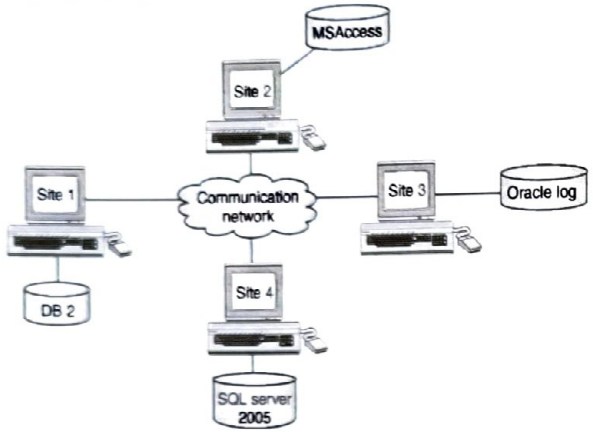


Fig. 1.6.2 : Heterogeneous distributed database system

1.7 Distributed DBMS Architectures

Q. Draw and Explain Architecture for distribute database system. **MU - Dec. 19, 10 Marks**

- The distributed database architecture of a database system defines structure of distributed data. Architecture first identifies the components of the database system, the function of each component and their relationship is defined.
- There are three alternative approaches to separating functionality across different DBMS-related processes. There are some parameters are considered for architecture implementation alternatives,

(a) Distribution

- It refers to the physical distribution of data among multiple sites. There are a multiple ways by which data can be distributed.
- Mainly there are two alternatives models,

1. **Client/server data distribution**
   - Server Responsibility : Data management
   - **Clients Responsibility** : To offer application environment with user interface.
2. **Peer-to-Peer data distribution**

   No separation of client and servers. Each machine has complete DBMS and it can communicate with other clients to execute queries and transactions.

(b) Autonomy

- Distributed database autonomy refers to the distribution of database control among multiple sites.
- It measures degree to which DBMS can operate independently.

- Design autonomy means individual DBMS is free to use any data models and transaction management techniques.
- Communication autonomy means DBMS can share any type of information with other DBMS.
- Execution autonomy explains DBMS can execute own transactions as per their own ideas.

(c) Heterogenity

- Heterogeneity can be for hardware, data model, and query language or with networking protocols.
- Architecture Models

1. Client-Server Systems
2. Collaborating Server Systems
3. Middleware Systems

1. **Client-Server systems**

(a) A Client-Server system has number of clients and some servers, a client process can send a query to any one of server process and server will manage to solve that query and replies with some result.

(b) Clients responsibility

User interface issues

(c) Servers responsibility

Servers manage data and execute transactions.

(d) Advantages

- It is relatively simple to implement due to centralized server system.
- Expensive Server machines are utilized by avoiding dull user interactions, which are now relegated to inexpensive client machines.
- Users can run simple a graphical user interface that they can understand, rather than the user interface on the server which is complex.

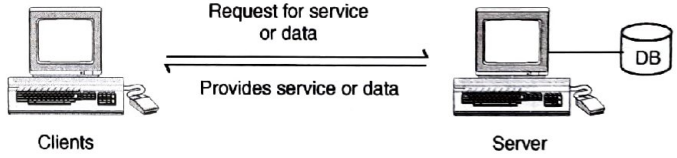


Fig. 1.7.1 : Typical client server system

(e) While writing Client-Server applications, it is important to remember the boundary between the client and the server and to keep the communication between them as simple as possible.

2. **Collaborating and Peer to Peer distributed database systems**

(a) The Client-Server architecture does not allow a single query to run on multiple servers systems as such client process would have to be capable of breaking a query into multiple small sub queries to be executed at different sites and then combining answers of such small sub queries together to solve the main queries.

(b) The client process is more complex and distinguishing between clients and servers becomes harder as functionality of both becomes much more similar. Eliminating this distinction leads us to an alternative to the Client-Server architecture that is a Collaborating Server system.

(c) We can have a collection of database servers, each of which is capable of running transactions against individual database, which execute transactions on many servers.

(d) When a server receives a query that requires access to data from some different servers, it generates appropriate sub queries to be executed by those servers and puts the results together to give results to client.

(e) The decomposition of the query should be done using cost-based optimization technique, considering the costs of network communication as well as local processing costs and other costs.

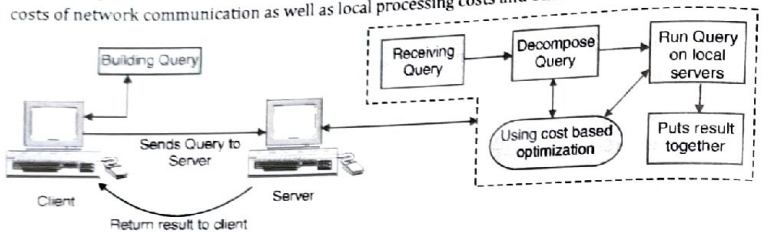


Fig. 1.7.2 : Collaborating Server Systems

(f) In peer to peer system all sites acts like client to serve application system. The detailed components of such distributed DBMS. The detailed components of a distributed DBMS are shown in Fig. 1.7.3.

User Processing :

1. **User interface :** User interface used to interpret user commands and formats the result coming from database site.
2. **Global query optimization :** It checks Global conceptual schema for user queries. It determines best execution strategy to minimize a cost of query.
3. **Global query executions :** It transfers global queries into local GCS and LCS in order to execute queries coming on global schema.

It executes queries in a distributed fashion, the execution monitors may communicate between various sites.

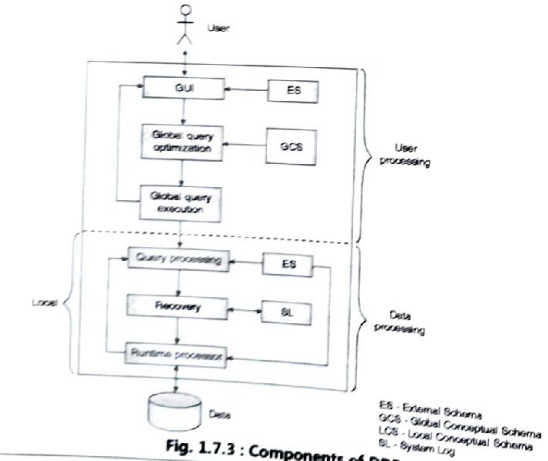


Fig. 1.7.3 : Components of DDB

Data Processing :

1. **Local query processing :** It selects access path and responsible for selecting best access path to access any data item in distributed environment.
2. **Recovery :** It keeps local database in consistent state even in case of failures.
3. **Runtime processor :** It acts like interface between operating system and database to provide buffer management, which is responsible for maintaining the buffer to manage the data accesses.

3. Middleware systems

(a) The Middleware system is designed to execute a single query on multiple servers without help of any database server it can be operated on different database management systems. It is also called Multi database system.

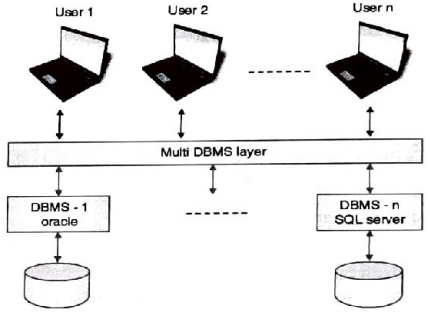


Fig. 1.7.4 : Multi database DDB

(b) It gives simplicity to integrate multiple legacy systems, whose basic capabilities cannot be extended with help of this system.

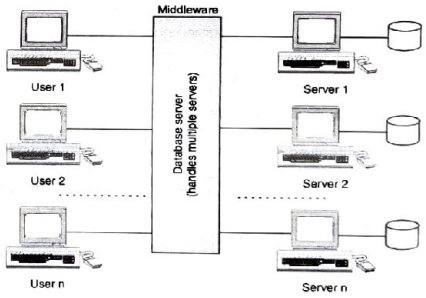


Fig. 1.7.5 : Middleware System

(c) We need just a single database server that is capable of managing queries and transactions from multiple servers. And all other servers only need to handle local queries and transactions.

(d) The software which helps the execution of queries and transactions across one or more than one independent database servers, such software is often called **middleware**.

(e) The middleware layer is capable of executing operations like joins and relational operations on data obtained from the many servers, but generally do not maintain any data by its own.

1.8 Advantages and Disadvantages of Distributed DBMS

Q. What are advantages and disadvantage of distributed DBMS ? **MU - May 16, 5 Marks**

Advantages of distributed database

1. In a distributed database, data can be stored in different systems like personal computers, servers, mainframes, etc.
2. A user doesn't know where the data is located physically. Database presents the data to the user as if it were located locally.
3. Database can be accessed over different networks.
4. Data can be joined and updated from different tables which are located on different machines.
5. Even if a system fails the integrity of the distributed database is maintained.
6. A distributed database is secure.

Disadvantages of distributed database

1. Since the data is accessed from a remote system, performance is reduced.
2. Static SQL cannot be used.
3. Network traffic is increased in a distributed database.
4. Database optimization is difficult in a distributed database.
5. Different data formats are used in different systems.
6. Different DBMS products are used in different systems which increase in complexity of the system.
7. Managing system catalog is a difficult task.
8. While recovering a failed system, the DBMS has to make sure that the recovered system is consistent with other systems.
9. Managing distributed deadlock is a difficult task.

Review Questions

Q. 1 Explain distributed database concepts in detail.

Q. 2 Explain different types of distributed database systems.

Q. 3 Describe various architectures of Distributed databases.

Q. 4 Give an overview of client server architecture and its relationship to distributed databases.

Q. 5 What is data transparency ? Explain types of transparencies distributed database should achieve.

Q. 6 Differentiate parallel and distributed databases.

Q. 7 Write short notes on : Promises Distributed DBMS.

Q. 8 Explain various issues in Distributed DBMS.

Q. 9 Write short note on : Distributed database architecture.

Q. 10 Explain in brief various architectures of Distributed DBMS.

Q. 12 What are the features of DDBMS ?

Q. 13 Write detailed notes on : Client server architecture.

□□□

2 Distributed Database Design

Syllabus

Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design

2.1 Objectives of Data Distribution

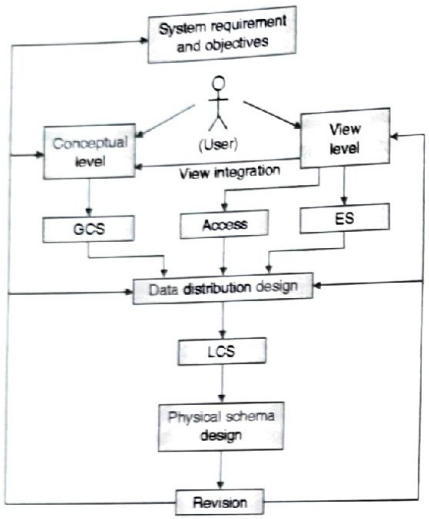
1. **Better query performance**
   - Distributed systems works on views rather than entire table.
   - Therefore, it will be better to fragment the table into sub relations which will be stored on multiple sites by distributing data.
2. **Good efficiency by locality of reference**
   - Data is stored at the place where it is most frequently required.
   - Data which is required most of times is stored locally, which makes data access faster.
3. **Improved concurrency rate**
   - Distributed data query is divided in multiple sub queries which operate on multiple data fragments in parallel.
   - This improves the degree of parallelism and number of queries executed in parallel.
   - Hence, it will also increases reliability and availability of database system.
4. **Better security**
   - Data which is not required to particular site may not be stored locally on that site.
   - So, No unauthorised access may occur.
5. **Performance improvement by distributing workload**
   - Performance of application may go down as data required for central application may be distributed at various sites.
   - Database load of one site is distributed using various sites is also one of the main advantage of distributed databases.
6. **Minimum cost of communication**
   - The cost of communication for processing remote request is considered as communication cost.
   - This cost will be minimum if we access local data.
   - Hence locality decrease cost of communication.

2.2 Top-Down Distributed Database Design

Introduction

(a) The process of decomposing the database into smaller multiple units called as **fragments.**

(b) **Design process**



GCS : Global Conceptual Schema

LCS : Local Conceptual Schema

ES : External Schema

Fig. 2.2.1 : Top Down Design Process

(c) **Activities in top-down design process**

1. **Analysis**
   - This phase defines system and application environment which will give idea about both data and functions required for database processing.
   - The Analysis phase defines expectations from distributed database system by specifying requirements respect to reliability and availability, cost effectiveness and scalability.
2. **View level design and conceptual level design**
   - The view level defines interfaces for database users.
   - The conceptual level defines the process to test entity types and relationships among them.
   - This process classified as,

   (i) Entity analysis : Entity analysis helps us to determine entities and their attributes and association between them.

   (ii) Functional analysis : Functional analysis deals with determining basic functionality of system.

3. **Data distribution design**
   - The Global Conceptual Schema (GCS) and access methods with some external schemas are produced by view design and conceptual design these acts like inputs to the distribution design.
   - This stem mainly designs the Local Conceptual Schemas (LCSs) by distributing the data over the multiple sites of fragmentation process.

2.3 Data Fragmentation

2.3.1 Introduction

Q. Explain data fragmentation for distributed database design. **MU - May 12,10 Marks**

(a) **Fragments :** The process of decomposing the database into smaller multiple units called as **fragments.**

(b) **Data fragmentation :** These fragments may be stored at various sites is called **data fragmentation**.

(c) **Completeness constraint :** The most important condition of data fragmentation process is that it must be complete i.e. once a database is fragmented, it must be always possible to reconstruct the original database from the fragments.

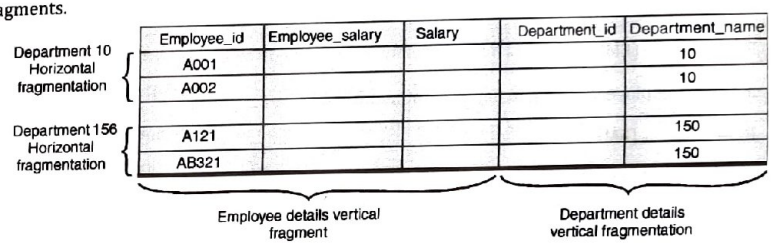


Fig. 2.3.1 : Horizontal and Vertical Fragmentation

2.3.2 Fragmentation Schema

It is a set of fragments that includes all attribute and tuples in the database and satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of database operations.

2.3.3 Types of Data Fragmentation

(a) Horizontal data fragmentation
(b) Vertical data fragmentation
(c) Mixed data fragmentation

2.3.3(A)Horizontal Fragmentation

1. **Introduction**
   - Horizontal fragmentation divides a relation horizontally into group of rows (tuple) to create subsets of tuples specified by a condition on one or more attributes of relation.
   - The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of the relation.

2. Overview

- Horizontal fragmentation is group of rows in relation.
- Horizontal fragments are specified by the 'SELECT' operation of the relational algebra on single or multiple attributes.
- Example selects all students of computer branch.

$\sigma_{Branch = "COMP"}$ (students)

3. Types

(a) Primary horizontal fragmentation

- Primary horizontal fragmentation is the fragmentation of primary relation.
- Relation on which other relations are dependent using foreign key is called as primary relation.

Example

Partition 1 : All employees belong to department number 10.

$R_1 \leftarrow \sigma_{Dept = 10}$ (EMP)

Partition 2 : All employees belong to department number 20.

$R_2 \leftarrow \sigma_{Dept = 20}$ (EMP)

(b) Derived horizontal fragmentation

- Horizontal fragmentation of a primary relation introduces derived horizontal fragmentation of other secondary relations that are dependent on primary relations.
- The fragmentation on some other fragmentation is called as derived horizontal fragmentation.

Example

Partition 1 : All employees belong to department number 10 and having age above 25.

$\sigma_{Age > 25}(R_1)$

Partition 2 : All employees belong to department number 20 and having age above 35.

$\sigma_{(Age>35)}(R_2)$

(c) Complete horizontal fragmentation

- It generates a set of horizontal fragments that include each and every tuple of original relation.
- For reconstruction of relation completeness is required. As every tuple must belongs to at least one of the partition.

- Consider relation below as R now subdivided in P_1, P_2, P_3 and P_4. In case of complete horizontal fragment, if relation R contains 50 tuples than total number of tuples in below 4 partitions should be 50 or more than 50.

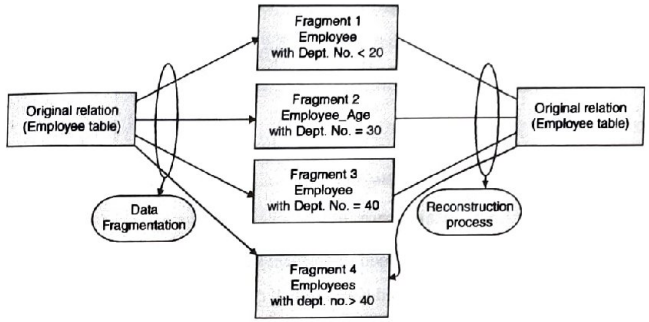


Fig. 2.3.2 : Complete horizontal fragmentation

Example

Partition 1 : All employees belong to department number less than 20.

$R_1 \leftarrow \sigma_{DeptNo <= 20}$ **(EMP)**

Partition 2 : All employees belong to department number less than 30.

$R_2 \leftarrow \sigma_{DeptNo = 30}$ (EMP)

Partition 3 : All employees belong to department number less than 40.

$R_3 \leftarrow \sigma_{DeptNo = 40}$ (EMP)

Partition 4 : All employees belong to department number above 40.

$R_4 \leftarrow \sigma_{Dept > 40}$ (EMP)

(d) Disjoint horizontal fragmentation

- It generates a set of horizontal fragments where no two fragments have common tuples.
- That means every tuple of relation belongs to one and only one fragment.

Example

Partition 1 : All employees having Age 18 or less.

$R_1 \leftarrow \sigma_{EmpAge <= 18}$ (EMP)

Partition 2 : All employees having Age above 18 and below 65.

$R_2 \leftarrow \sigma_{EmpAge > 18 \text{ AND } EmpAge < 65}$ (EMP)

Partition 3 : All employees having Age above 65.

$R_3 \leftarrow \sigma_{EmpAge >= 65}$ (EMP)

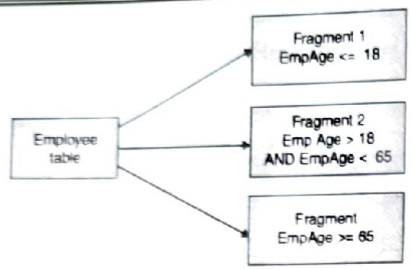


Fig. 2.3.3 : Disjoint horizontal partitioning

4. Reconstruction process for horizontal fragments

- To reconstruct the original relation we need to perform set UNION ($\cup$) operation on fragments.
- The original relation can be reconstructed if and only if completeness constraint is satisfied.
- **Example** : Consider relation shown in complete horizontal partitioning we can reconstruct relation as it is satisfying completeness constraints :

$$R \leftarrow R_1 \cup R_2 \cup R_3 \cup R_4$$

2.3.3(B) Vertical Fragmentation

1. Introduction

- Vertical fragmentation divides a relation vertically into group of columns.
- When each site does not need all the attributes of a relation, vertical fragmentation is used to fragment the relation vertically by columns.
- It is necessary to include primary key or some common candidate key in every vertical fragment to reconstruct the original relation from the fragments.

2. Overview

- Vertical fragmentation is group of columns in relation.
- Vertical fragmentation can be specified by 'PROJECT' operation of the relational algebra.
- Example Select Name and address of all students of computer branch.

$$\pi_{Name,\ Address}\ (student)$$

3. Types of vertical fragmentation

(a) Complete vertical fragmentation

- It generates a set of vertical fragments that include all the attributes of original relation and share only primary key of original relation.
- Consider relation with following schema :

Student (Sid, Name, Age, Address, Phone, Class, Fees)

$$P_1 \rightarrow \pi_{Sid,\ Name,\ Address}\ (student)$$

$$P_2 \rightarrow \pi_{Sid,\ Age,\ Phone}\ (student)$$

$$P_3 \rightarrow \pi_{Sid,\ class,\ Fees}\ (student)$$

4. Reconstruction

- To reconstruct the original relation, we need to perform FULL OUTER JOIN ($\bowtie$) operation on fragments.
- The original relation can be reconstructed if and only if completeness constraint is satisfied that means there should be either one column which is common between two partitions.
- Example consider above relation we can reconstruct relation as it is satisfying completeness constraints

$$R \leftarrow P_1 \bowtie P_2 \bowtie P_3$$

5. Affinity Matrix

Q. Write short note on Affinity matrix **MU - May 15, 10 Marks**

- The information required as input for VF is related to applications and attributes are places in one fragment which are usually accessed together.
- Such, relativity of attributes are obtained from queries and collected in the Attribute Usage Matrix and Attribute Affinity Matrix.

  E.g.

  For Given are the user queries

  $\mathbf{Q = (q_1, \ldots, q_n)}$

  It will run on relation,

  $\mathbf{R (A_1, \ldots, A_n)}$

- Attribute Usage Matrix
- It is used to represent use of Query attribute.

$$Use(Q_i, A_j) = 1; \quad \text{if } Q_i \text{ uses } A_j$$
$$= 0; \quad \text{Otherwise}$$

Example

EMP (ENO, ENAME, SAL, CITY)

Consider Quries,

Q_1 = SELECT SAL FROM EMP WHERE ENO=Value

Q_2 = SELECT ENAME, SAL FROM EMP

Q_3 = SELECT ENAME FROM EMP WHERE CITY=Value

Q_4 = SELECT SUM (SUM) FROM EMP WHERE CITY =Value

Consider Attributes,

A_1 = ENO

A_2 = ENAME

A_3 = SAL

A_4 = CITY

E.g.

Attribute Usage Matrix

	A_1	A_2	A_3	A_4
Q_1	1	0	1	0
Q_2	0	1	1	0
Q_3	0	1	0	1
Q_4	0	0	1	1

(d) Attribute Affinity Matrix

It Represents frequency of two attributes A_1 and A_2 with respect to query Q.

aff (Ai, Aj) = $\sum$ cost (Q_k).Freq(Q_k)

cost(Q_k) = Accesses to A_1 and A_2 with respect to query Q from site.

Assuming this parameter as 1,

Freq (Q_k) = Frequency of Query Q_k from site,

Assuming as,

	Site 1	Site 2	Site 3
Freq (Q_1)	15	20	10
Freq (Q_2)	5	0	0
Freq (Q_3)	25	25	25
Freq (Q_4)	3	0	0

E.g.

Attribute Affinity Matrix

	A_1	A_2	A_3	A_4
A_1	45	0	45	0
A_2	0	80	5	75
A_3	45	5	53	3
A_4	0	75	3	78

Aff (A_1, A_3) = Freq (Q_1) + Freq (Q_3) = Freq (Q_3)

= 15+20+10 = 45

(Q_3 is the only query to access both A_1 and A_3)

2.3.3(C) Mixed (Hybrid) Fragmentation

1. Introduction

- We can mix two types of fragmentation i.e. horizontal and vertical fragmentations yielding a mixed fragmentation.
- This fragmentation is generally used in many applications.

2. Overview

- Mixed fragmentation is group of columns and rows in relation.
- Mixed fragmentation can be specified by 'PROJECT' and 'SELECT' operation of the relational algebra.

3. Example

Mixed fragmentation can be applied to student table for following student schema;

Student table

Sid	SName	Age	Branchid	Bname

Student table contains information about student and branches associated with particular student. Branches can be Computer Science (CS) or IT branch.

Student database

	SName	Age	Sid	Branchid	Bname	
F_1→	Smriti	20	1	10	CS	← F_3
	Jay	24	2	10	CS	
	Ashok	22	3	10	CS	
	Neha	21	4	20	IT	
	Raj	20	5	20	IT	← F_4
F_2→	Harshad	23	6	20	IT	

(a) Fragment 1 : Student details of all students in 'CS' Branch.

$F_1 \rightarrow \pi_{Sid, SName, age} (\sigma_{Bname = 'CS'} (student))$

Sid	SName	Age
1	Smriti	20
2	Jay	24
3	Ashok	22

(b) Fragment 2 : Student details of all students in 'IT' Branch.

$F_2 \rightarrow \pi_{Sid, SName, age} (\sigma_{Bname = 'IT'} (student))$

Sid	SName	Age
4	Neha	21
5	Raj	20
6	Harshad	23

(c) Fragment 3 : Find all student's branch details of CS Branch

$F_3 \rightarrow \pi_{Sid, Branchid, BName} (\sigma_{Bname = 'CS'} (student))$

Sid	Branchid	Bname
1	10	CS
2	10	CS
3	10	CS

(d) Fragment 4 : Find all student's branch details of IT Branch

$F_4 \rightarrow \pi_{Sid, Branchid, BName}(\sigma_{Bname = 'IT'}(student))$

Sid	Branchid	Bname
4	20	IT
5	20	IT
6	20	IT

4. Reconstruction

- To reconstruct the original relation by performing Union and FULL OUTER JOIN (⋈) operation in appropriate order on fragments.
- The original relation can be reconstructed if and only if completeness constraint is satisfied that means there should be either one column which is common between two partitions.
- **Example** :

To reconstruct above fragmentation we will first find union of F_1 and F_2 which will give me details of all students.

$F_1 \cup F_2$

Sid	SName	Age
1	Smriti	20
2	Jay	24
3	Ashok	22
4	Neha	21
5	Raj	20
6	Harshad	23

Now, we will find details of or department in which student study by taking Union of F_3 and F_4.

$F_3 \cup F_4$

Sid	Branchid	Bname
1	10	CS
2	10	CS

Sid	Branchid	Bname
3	10	CS
4	20	IT
5	20	IT
6	20	IT

Now to reconstruct main table, we Join above two tables using Join (⋈) with help of 'Sid' as common column.

$$R \Rightarrow (F_1 \cup F_2) \bowtie (F_3 \cup F_4)$$

Above query will returns the original relation as in table : student database.

Ex. 2.3.1 : Using a snapshot of the following centralized schema of a database :

Departments (DN, DName, Budget, Location)

Employees (EN, EName, Title, DNo)

Salary (Title, Salary)

(a) Show 2 examples of horizontal fragmentation with fragmentation rules.

(b) Show 2 examples of vertical fragmentation with fragmentation rules.

(c) Show 2 examples of derived fragmentation with fragmentation rules.

(d) Demonstrate the correctness of your fragmentation rules. **MU - Dec. 15, Dec. 19. 10 Marks**

Soln. :

(a) Show 2 examples of horizontal fragmentation with fragmentation rules

Horizontal fragmentation

- Horizontal fragmentation divides a relation horizontally into group of rows (tuple) to create subsets of tuples specified by a condition on one or more attributes of relation.
- The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of the relation.

Fragment 1 : All employees working in Mumbai

a. **Relations** : Employee

b. **Attributes** : All (*) attributes of employee

c. **Guard condition** :Location = 'Mumbai'

d. **Query**

$F_1 \leftarrow (\sigma_{Location = 'Mumbai'}(Employee))$

Fragment 2 : All employees not working in Mumbai

a. **Relations** : Employee

b. **Attributes** : All (*) attributes of employee

c. **Guard condition** :Location<> 'Mumbai'

d. **Query**

$F_2 \leftarrow (\sigma_{Location <> 'Mumbai'}(Employee))$

(b) Show 2 examples of vertical fragmentation with fragmentation rules.

Vertical fragmentation

- Vertical fragmentation divides a relation vertically into group of columns.
- When each site does not need all the attributes of a relation, vertical fragmentation is used to fragment the relation vertically by columns.
- It is necessary to include primary key or some common candidate key in every vertical fragment to reconstruct the original relation from the fragments.

Fragment 1 : All Department names and name of their department

a. Relations : Project

b. Attributes :Employee name and Project Name

c. Guard condition : No

d. Query

$$F_3 \leftarrow \pi_{Ename, Pname}(Employee)$$

Fragment 2 : All Project names and name of their department

a. Relations : Project, Emp

b. Attributes :Employee name,Employee location and Budget

c. Guard condition : No

d. Query

$$F_4 \leftarrow \pi_{Ename, Location, Budget}(Emp)$$

(c) Show 2 examples of derived fragmentation with fragmentation rules

Fragment 1 : All Department names and name of their department

a. Relations : Project

b. Attributes : Employee name and Project Name

c. Guard condition : Location = 'Mumbai'

d. Query

$$F_5 \leftarrow \pi_{Ename, Pname}((\sigma_{Location <> 'Mumbai'}(Employee)))$$

Fragment 2 : All Project names and name of their department

a. Relations : Project, Emp

b. Attributes : Employee name, Employee location and Budget

c. Guard condition : Location <> 'Mumbai'

d. Query

$$F_6 \leftarrow \pi_{Ename, Location, Budget}((\sigma_{Location <> 'Mumbai'}(Employee)))$$

(d) Demonstrate the correctness of your fragmentation rules.

Reconstruction case (a): All employees working in company.

$Employee \leftarrow F_1 \cup F_2$

$Employee \leftarrow F_3 \bowtie F_4$

$Employee \leftarrow F_6 \bowtie F_5$

Ex. 2.3.2 : Perform horizontal fragmentation for student relation as given below.

Also give the correctness criteria for it.

Student (SrudentRollno., StudentName, CourseName, CourseFees, year) **MU - May 16, 10 Marks**

Soln. :

Horizontal fragmentation

- Horizontal fragmentation divides a relation horizontally into group of rows (tuple) to create subsets of tuples specified by a condition on one or more attributes of relation.
- The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of the relation.

Fragment 1 : All Students studies in Mumbai

a. Relations : Student

b. Attributes : All (*) attributes of employee

c. Guard condition : Location = 'Mumbai'

d. Query

$$F_1 \leftarrow (\sigma_{Location = 'Mumbai'}(Student))$$

Fragment 2 : All Students not studies in Mumbai

a. Relations : Student

b. Attributes : All (*) attributes of employee

c. Guard condition : Location <> 'Mumbai'

d. Query

$$F_2 \leftarrow (\sigma_{Location <> 'Mumbai'}(Student))$$

Demonstrate the correctness of your fragmentation rules.

Reconstruction of All employees working in company.

$Employee \leftarrow F_1 \cup F_2$

Ex. 2.3.3 : Using a snapshot of the following centralized schema of a database :

PROJ()

PAY()

EMP()

ASG()

(a) Show 2 examples of horizontal fragmentation with fragmentation rules.

(b) Show 2 examples of vertical fragmentation with fragmentation rules.

(c) Show 2 examples of derived fragmentation with fragmentation rules. **MU - May 19, 10 Marks**

Soln. :

(a) Show 2 examples of horizontal fragmentation with fragmentation rules

Horizontal fragmentation

- Horizontal fragmentation divides a relation horizontally into group of rows (tuple) to create subsets of tuples specified by a condition on one or more attributes of relation.

- The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of the relation.

Fragment 1 : All employees working in Mumbai

a. **Relations** : EMP

b. **Attributes**: All (*) attributes of employee

c. **Guard condition** : SALARY >=50000

d. **Query**

$F_1 \leftarrow (\sigma_{SALARY>=5000}(EMP))$

Fragment 2 : All employees not working in Mumbai

a. **Relations** : Employee

b. **Attributes** : All (*) attributes of employee

c. **Guard condition** :SALARY <50000

d. **Query**

$F_1 \leftarrow (\sigma_{SALARY<5000}(EMP))$

(b) Show 2 examples of vertical fragmentation with fragmentation rules.

Vertical fragmentation

- Vertical fragmentation divides a relation vertically into group of columns.
- When each site does not need all the attributes of a relation, vertical fragmentation is used to fragment the relation vertically by columns.
- It is necessary to include primary key or some common candidate key in every vertical fragment to reconstruct the original relation from the fragments.

Fragment 1 : All Department names and name of their department

a. **Relations** : PROJ

b. **Attributes** : PROJECT NUMBER AND NAME

c. **Guard condition** : No

d. **Query**

$F_3 \leftarrow \pi_{PNO, Pname}(PROJ)$

Fragment 2 : All Project names and name of their department

a. **Relations** : Project, Emp

b. **Attributes** : PROJECT NUMBER AND Budget

c. **Guard condition** : No

d. **Query**

$F_4 \leftarrow \pi_{PNO, BUDGET}(PROJ)$

(c) Show 2 examples of derived fragmentation with fragmentation rules

Fragment 1 : All Department names and name of their department

a. **Relations** :ASG

b. **Attributes** : Employee NUMBER, Project NUMBER AND DURATION

c. **Guard condition** :RESPONSIBILITY = "MANAGER"

d. **Query**

$F_5 \leftarrow \pi_{PNO, ENO, DURATION}((\sigma_{RESPONSIBILITY = "MANAGER}(Employee)))$

Fragment 2 : All Project names and name of their department

a. Relations :ASG

b. Attributes :Employee NUMBER, Project NUMBER AND DURATION

c. Guard condition :RESPONSIBILITY <> "MANAGER"

d. Query

$F_6 \leftarrow \pi_{PNO, ENO, DURATION}((\sigma_{RESPONSIBILITY <> "MANAGER}(Employee)))$

2.3.4 Bond Energy Algorithm

Q. Write short note on Bond energy algorithm. **MU - May 16, 10 Marks**

Vertical Splitting

Bond Energy Algorithm

Given : The following access characteristics and access frequencies for Q1,...,Q4, calculate the optimal vertical splitting using the Bond Energy Algorithm (BEA),

Steps :

1. Prepare an affinity matrix.
2. Apply BEA algorithm.
3. Perform vertical splitting by maximizing the split quality.

	Name	Family	Age	Position	Location
q_1	1	1	1	0	0
q_2	0	0	1	1	0
q_3	0	1	0	1	1
q_4	0	0	1	0	1

	Site A	Site B	Site C
q_1	20	1	0
q_2	10	5	9
q_3	80	1	9
q_4	2	5	4

Solution :

q_1 :	21
q_2 :	24
q_3 :	90
q_4 :	11

q_1 :	A1	A2	A3	21
q_2 :	A3	A4	24	
q_3 :	A2	A4	A5	90
q_4 :	A3	A5	11	

	A1	A2	A3	A4	A5
A1	21	21	21	0	0
A2	21	111	21	90	90
A3	21	21	56	24	11
A4	0	90	24	114	90
A5	0	90	11	90	101

Place attributes

Place A1

Contribution at pos 0 = 2121

Contribution at pos 1 = - 1598

Contribution at pos 2 = 2058

Attribute A1 is placed at pos 0 : [A1, A5, A3]

Place A2

Contribution at pos 0 = 3213

Contribution at pos 1 = 28503

Contribution at pos 2 = 28732

Contribution at pos 3 = 7098

Attribute A2 is placed at pos 2: [A1, A5, A2, A3]

place A4

Contribution at pos 0 = 2394

Contribution at pos 1 = 27987

Contribution at pos 2 = 29157

Contribution at pos 3 = 28716

Contribution at pos 4 = 6960

Attribute A4 is placed at pos 2: [A1, A5, A4, A2, A3]

Resulting order: [A1, A5, A4, A2, A3]

Find fragments :

split at [A1, A2, A3, A4] | [A5]

Accesses frag1 alone: 45

Accesses frag2 alone: 0

Accesses frag1 and frag2: 101

Split quality = -10201

Split at [A1, A2, A3] | [A4, A5]

Accesses frag1 alone: 21

Accesses frag2 alone: 0

Accesses frag1 and frag2: 125

Split quality = -15625

Split at [A1, A3] | [A2, A4, A5]

Accesses frag1 alone: 0

Accesses frag2 alone: 90

Accesses frag1 and frag2: 56

split quality = -3136

split at [A1] | [A2, A3, A4, A5]

Accesses frag1 alone: 0

Accesses frag2 alone: 125

Accesses frag1 and frag2: 21

Split quality = -441

Split at [A1, A2, A3, A5] | [A4]

Accesses frag1 alone: 32

Accesses frag2 alone: 0

Accesses frag1 and frag2: 114

Split quality = -12996

Split at [A1, A3, A5] | [A2, A4]

Accesses frag1 alone: 11

Accesses frag2 alone: 0

Accesses frag1 and frag2: 135

Split quality = -18225

Split at [A1, A5] | [A2, A3, A4]

Accesses frag1 alone: 0

Accesses frag2 alone: 24

Accesses frag1 and frag2: 122

Split quality = -14884

Split at [A1, A3, A4, A5] | [A2]

Accesses frag1 alone: 35

Accesses frag2 alone: 0

Accesses frag1 and frag2 : 111

Split quality = -12321

Split at [A1, A4, A5] | [A2, A3]

Accesses frag1 alone: 0

Accesses frag2 alone: 0

Accesses frag1 and frag2: 146

Split quality = -21316

Split at [A1, A2, A4, A5] | [A3]

Accesses frag1 alone: 90

Accesses frag2 alone: 0

Accesses frag1 and frag2: 56

Split quality = -3136

Optimal split(s) (sq = -441):

[A1] | [A2, A3, A4, A5]

2.5 Data Replication for Distributed Database Design

Q. Explain data replication technique for distributed database design. **MU - May 12, Dec. 19, 10 Marks**

2.5.1 Introduction

- To improve availability of data, data may be replicated at multiple sites by copying data at multiple places.
- An entire relation can be replicated at one or more then one sites. Similarly, one or more fragments of a relation can be replicated at other sites.
- Replication schema is description about method replication of fragments.
- There are three basic ideas : No replication, Full replication and Partial replication.

2.5.2 Goals

(a) Increasing Availability of Data : If one site with data goes down we can fetch data from any other replicated site so data availability will not go down, although site is down.

(b) Faster Query Evaluation : Data of site is replicated, hence each site having own local copy of data, there will be faster execution of query.

2.5.3 Types

(a) Synchronous Replica will be modified when as soon as original relation is modified. Hence no difference between replica and original data.

(b) Asynchronous Replica will be not modified quickly as original relation is modified. Replica will be modified after commit is fired on to the databases.

(c) Schemes of replication

1. Full replication
2. No replication
3. Partial replication

1. Full replication :

(a) Fully replicated distributed database is extreme case of replication, where each site has entire database.

(b) Advantages

- High availability of data as same data available on multiple sites.
- Faster execution for retrieval type of queries.

(c) Disadvantages

- Slows down update operation.
- Makes concurrency control and recovery tasks difficult as there are multiple sites.

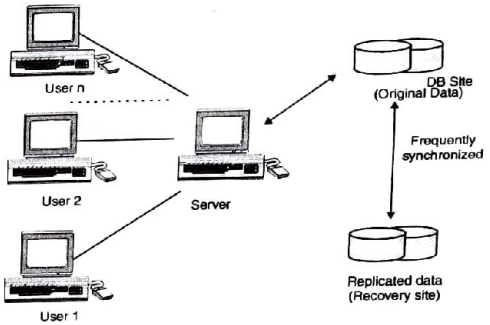


Fig. 2.5.1 : Fully replicated database

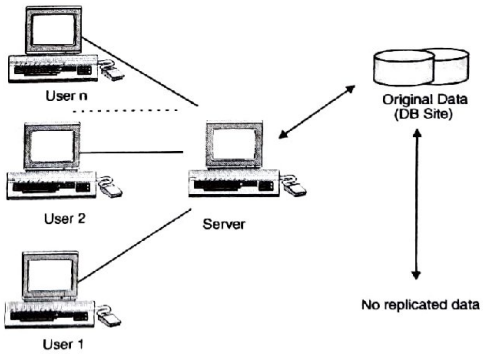


Fig. 2.5.2 : No replicated data

2. No replication :

(a) No replication means, each fragment is stored exactly at one site.

- In this case generally all fragments are disjoint.
- If vertical fragment used then primary key may be replicated.

(b) Advantages

- Concurrency has been minimized as only one site to be updated.
- Only one site hence easy to recover data.

(c) Disadvantages

- Poor availability of data as centralized server only has data.
- Slow down query execution as multiple clients accessing same server.

3. Partial replication :

(a) Partial replication means, some fragments are replicated whereas others are not.

(b) Advantages

- Number of replicas created for a fragment directly depends upon the importance of data in that fragment.
- Optimized architecture give advantages of both full replication and no replication scheme.

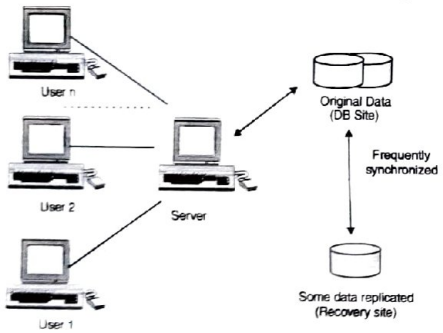


Fig. 2.5.3 : Partially replicated database

2.6 Allocation Techniques for Distributed Database Design

Q. Explain data allocation technique for distributed database design. **MU - May 12, 10 Marks**

Q. List and Explain allocation techniques for distributed database. **MU - May 19, Dec. 19, 5 Marks**

- Fragmentation process results in number of fragments; once fragments are generated they are assigned to various sites.
- Allocation schema describes the allocation of fragments to various sites of the distributed databases.
- **Data Distribution :** Each data fragment is assigned to any one site in the distributed system. This process is called as data distribution or data allocation.
- The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.
- Finding a good solution to data allocation is a complex optimization problem.

2.7 Transparencies in Distributed Database Design

Q. Explain the different types of transparencies in a Distributed Database System. **MU - May 15, Dec. 15, May 16**

A DBMS should hide the detail of where each data item (like tables or relations) is physically stored within the system. This concept is also referred as transparencies.

2.7.1 Distribution Transparency

Data distribution considers the database distributed at multiple sites as a single entity. Distribution transparency refers to extent to which data distribution is hidden from users.

(a) Fragmentation transparency

- The process of decomposing the database into smaller multiple units called as fragmentation.
- Fragmentation transparency makes the user unaware of the existence of data fragments present whether it is horizontal fragmentation or vertical fragments.
- User accesses the data as like normal non fragmented data.

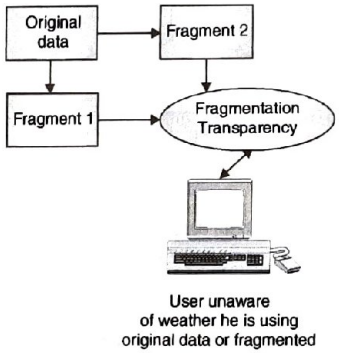


Fig. 2.7.1 : Fragmentation transparency

Example,

Student (Sid, Stud_Name, Stud_age, Stud_mob, Stud_Year, Stud_Location) relation divided in multiple fragments for distribution.

Now, if management wish to access all students located in 'Mumbai'.

Then SQL Query will be,

```
SELECT * FROM Student WHERE Stud_Location LIKE 'MUMBAI';
```

(b) Network transparency

In this all internal network operations are hidden from the user.

Network may be divided into location transparency and naming transparency.

(i) Location transparency refers to a task performed by user is independent of the location of data and the location of the system.

Example,

If above relation is divided in two fragments :

Student1 ($\sigma_{Bname = 'IT'}$ (student))

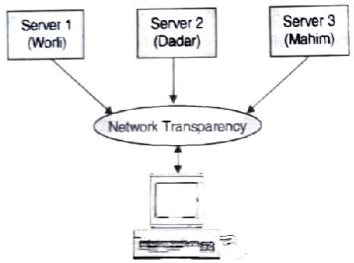
Student2 ($\sigma_{Bname <> 'IT'}$ (student)) using horizontal fragmentation.

Query : Find all students located in 'Mumbai'

Students may be in Student1 or may be in Student2

```
SELECT *    FROM Student1
WHERE Stud_Location    LIKE 'MUMBAI'
UNION
SELECT *    FROM Student2
WHERE Stud_Location    LIKE 'MUMBAI';
```

(ii) Naming transparency states that once a name is specified, these objects can be accessed unambiguously.



(User unaware about data comes from which server)

Fig. 2.7.2 : Network transparency

(c) Replication transparency

- Replication is copying data at multiple sites and at multiple locations for better availability, performance and reliability.
- Replication transparency makes the user unaware of the existence of copies of data.
- User is unaware whether data accessed is original or replicated.

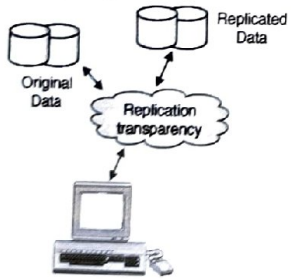


Fig. 2.7.3 : Replication transparency

Example,

If above relation is divided in two fragments :

Student1 ($\sigma_{Bname = 'IT'}$ (student))

Student2 ($\sigma_{Bname = 'IT'}$ (student))

Student2 replica of student1 fragment stored at another site.

Query: Find all students located in 'Mumbai'

Students may be in Student1 or may be in Student2 :

```
SELECT *    FROM Student1
WHERE Stud_Location    LIKE 'MUMBAI'
UNION
SELECT *    FROM Student2
WHERE Stud_Location    LIKE 'MUMBAI';
```

(d) Transaction transparency

- Transaction transparency is useful for maintaining integrity and consistency of data.
- A single distributed transaction can update data stored at multiple sites connected by network.
- In such cases, transaction is divided in multiple sub transactions.

(i) Concurrent transparency :

Distributed databases may have multiple users connected concurrently to distributed system.

- In this case also user should be unaware about degree of concurrency at that instance of time.
- It also makes sure that all individual transactions are executed independent of each other.

(ii) Failure transparency :

- Distributed database system is subject may fail as like a centralized database system.
- There can be some additional risk of failure of a communication network.
- Distributed databases must be able to detect a failure, than reconfigure the system in order to recover when a processor or link is repaired.
- Distributed DBMS is responsible for database recovery when any type of failure has occurred.
- The distributed DBMS at each site has a transaction manager responsible for :
  1. Maintains a log of transactions
  2. Maintains an appropriate concurrency control during parallel execution

2.7.2 Performance Transparency

- Performance Transparency makes sure that the distributed database offers performance as like centralised database.
- Distributed system should not suffer from any performance degradation due to its structure, it should always choose cost effective strategy to execute query.

2.7.3 DBMS Transparency

- Distributed databases may have multiple users connected to distributed system with different or similar DBMS software.
- In this case also user should be unaware about the DBMS software he is using for accessing data at that instance of time.

2.8 Design Problems

Ex. 2.8.1 : Employee working in department. Employee can work on more then one project for some hours. Each department have its location which stores its city. Each employee have some dependents.

1. Write global distributed schema for above statements.
2. Show 2 examples of horizontal fragmentation.
3. Show 2 examples of vertical fragmentation.
4. Show 2 examples of derived fragmentation.

Soln. :

Step I : Designing global distributed database schema

As we have studied in EER diagrams chapter apply concepts and identify all entities and attribute which acts like global distributed schema.

Employee	(eid, did, ename , age, salary)
Department	(did, mid, dname)
Projects	(pid, did, pname)
Work_on	(eid, pid, hrs)
Locations	(lid, did, city)
Dependents	(eid, dname, age)

Step II : Horizontal fragmentation

- Horizontal fragmentation divides a relation horizontally into group of rows (tuple) to create subsets of tuples specified by a condition on one or more attributes of relation.
- The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of the relation.

Fragment 1 : All employees working in Mumbai

a. **Relations** : Employee, Dept, Location

b. **Attributes** : All (*) attributes of employee

c. **Guard condition** : city = 'Mumbai'

d. **Query**

$F_1 \leftarrow (\sigma_{Location.city = 'Mumbai'}$ (Employee ⋈ Dept ⋈ Location))

Fragment 2 : All employees not working in Mumbai

a. **Relations** : Employee, Dept, Location

b. **Attributes** : All (*) attributes of employee

c. **Guard condition** : city <> 'Mumbai'

d. **Query**

$F_2 \leftarrow (\sigma_{Location.city <> 'Mumbai'}$ (Employee ⋈ Dept ⋈ Location))

Reconstruction: All employees working in company.

Employee $\leftarrow F_1 \cup F_2$

Step III : Vertical fragmentation

- Vertical fragmentation divides a relation vertically into group of columns.
- When each site does not need all the attributes of a relation, vertical fragmentation is used to fragment the relation vertically by columns.
- It is necessary to include primary key or some common candidate key in every vertical fragment to reconstruct the original relation from the fragments.

Fragment 1 : All Project names and name of their department

a. **Relations** : Project, Dept

b. **Attributes** : Department name and Project Name

c. **Guard condition** : No

d. **Query**

$F_1 \leftarrow \pi_{Dname, Pname}$ (Dept ⋈ Project)

Fragment 2 : All Project names and name of their department

a. **Relations** : Project, Emp

b. **Attributes** : Employee name and Project Name

c. **Guard condition** : No

d. **Query**

$F_1 \leftarrow \pi_{Ename, Pname}$ (Emp ⋈ Project)

Step IV : Mixed or Derived fragmentation

- Horizontal fragmentation of a primary relation introduces Derived horizontal fragmentation of other secondary relations that are dependent on primary relations.
- The fragmentation on some other fragmentation is called as Derived horizontal fragmentation.

Fragment 1 : Name of All employees working in Mumbai

a. **Relations** : Employee, Dept, Location

b. **Attributes** : Name of employee

c. **Guard condition** : city = 'Mumbai'

d. **Query**

$F_{a1} \leftarrow \pi_{Ename}(\sigma_{Location.city = 'Mumbai'}$ (Employee ⋈ Dept ⋈ Location))

$F_{a1} \leftarrow \pi_{Ename}(F_1)$

Fragment 2 : Name of All employees not working in Mumbai

a. **Relations** : Employee, Dept, Location

b. **Attributes** : Name of employee

c. **Guard condition** : city <> 'Mumbai'

d. **Query**

$F_{a2} \leftarrow \pi_{Ename}(\sigma_{Location.city <> 'Mumbai'}$ (Employee ⋈ Dept ⋈ Location))

$F_{a2} \leftarrow \pi_{Ename}(F_2)$

Ex. 2.8.2 : Consider following global schema of an company database who keep track of company's employees, department and projects. **MU - May 15, 20 Marks**

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

(a) Perform Primary Horizontal Fragmentation (PHF) of relation PROJ with pname and budget of projects given their number issued at three sites and access project information according to budget one site accesses ≤ 200000 other accesses > 200000.

(b) Explain how the above resulting PHF fulfill the correctness rules of fragmentation.

(c) Explain how the above resulting DHF fulfill the correctness rules of fragmentation.

Soln. :

(a) Perform Primary Horizontal Fragmentation (PHF) :

PHF of relation PROJ

Applications :

Find the name and budget of projects given their no.

Issued at three sites

Access project information according to budget

one site accesses ≤200000 other accesses >200000

Simple predicates

For application (1)

p_1 : LOC = "Montreal"

p_2 : LOC = "New York"

p_3 : LOC = "Paris"

For application (2)

p_4 : BUDGET ≤ 200000

p_5 : BUDGET > 200000

$Pr = Pr' = \{p_1, p_2, p_3, p_4, p_5\}$

Fragmentation of relation PROJ continued

Minterm fragments left after elimination

m_1 : (LOC = "Montreal") ∧ (BUDGET ≤ 200000)

m_2 : (LOC = "Montreal") ∧ (BUDGET > 200000)

m_3 : (LOC = "New York") ∧ (BUDGET ≤ 200000)

m_4 : (LOC = "New York") ∧ (BUDGET > 200000)

m_5 : (LOC = "Paris") ∧ (BUDGET ≤ 200000)

m_6 : (LOC = "Paris") ∧ (BUDGET > 200000)

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

PNO	PNAME	BUDGET	LOC
P2	Database develop	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenace	310000	paris

(b) Perform Vertical Fragmentation (VHF) :

VHF of relation PROJ

Applications :

Find the name of PNO related to budget from PROJ

PROJ1: information about project budgets

PROJ2: information about project names

$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database develop	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

(c) Perform derived horizontal fragmentation (DHF) of relation EMP with respect to PAY (p1:sal>30000 and p2:sal≤30000).

Perform Derived Horizontal Fragmentation (DHF) :

DHF of relation PROJ

⇒ **Applications :**

Find the name of employees whose salary is greater than 30000.

Given link L_1 where owner (L_1)=SKILL and member(L_1)=EMP

$EMP_1 = EMP \propto PAY_1$

$EMP_2 = EMP \propto PAY_2$

Where,

$PAY_1 = \sigma_{SAL \leq 30000}$ (PAY)

$PAY_2 = \sigma_{SAL > 30000}$ (PAY)

EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

For correctness rules, show how the fragments fulfill completeness reconstruction and disjointness property.

Review Questions

Q. 1 Explain distributed database concepts in detail.

Q. 2 Give and explain various objectives of data distribution.

Q. 3 Explain data fragmentation in distributed database design.

Q. 4 Consider the global schema

PATIENT (Number, Name, SSN, Amount_Due, Dept, Doctor, Med_treatment)

DEPARTMENT (Dept, Location, Director)

STAFF (Staffnum, Director, Task)

(a) Show 2 examples of horizontal fragmentation.

(b) Show 2 examples of vertical fragmentation.

(c) Show 2 examples of derived fragmentation.

Q. 5 Explain data fragmentation, replication and allocation techniques for Distributed Database design.

Q. 6 Write short notes on

(a) Distributed DBMS

(b) Data Replication in distributed DBMS

Q. 7 What is data transparency ? Explain types of transparencies distributed database should achieve.

□□□

3

Distributed Transaction Management

Syllabus

Distributed Transaction Management : Definition, properties, types, architecture

Distributed Query Processing : Characterization of Query Processors, Layers/ phases of query processing.

3.1 Distributed Transaction Management

1. Introduction/ Definition

- In simple SQL Query processing, each SQL command is sent to database server as a query and server will reply with result set as a response to it.
- Simple query fired on DBMS is called **SQL operation.**
- Collection of multiple operations that forms a single logical unit is called as **transaction.**
- A transaction is a sequence of one or more SQL statements that combined together to form a single logical unit of work.

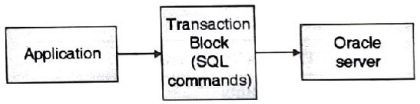


Fig. 3.1.1 : Executing Transaction in DBMS

- In case of transaction, we send multiple SQL commands (DML, DRL etc.) to database server which executed one after other.

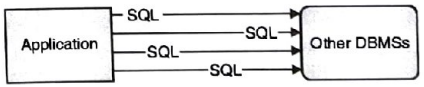


Fig. 3.1.2 : Executing Single Operation in DBMS

- In place of sending one by one SQL command to server we can combine multiple operations which are performing logically similar task and then send to server as a single logical unit called **transaction.**
- **E.g.** Transferring Rs.100 from one account to other
  1. Withdraw Rs.100 from account_1
  2. Deposit Rs.100 to account_2

2. Transaction operations

Types of operations that can be done inside transaction,

(a) Read operation

- Read operation transfers data item from (e.g. table) the database memory to a local buffer of the transaction that executed the read operation.
- **Ex.** Data Selection / Retrieval Language

```
SELECT *
FROM   Students
```

(b) Write

- Write operation transfer data from local memory to database.
- Write operation transfers one data item from the local buffer of the transaction that executed the write back to the database.
- Ex. Data Manipulation Language (DML)

```
UPDATE   Student
SET      Name = 'Bhavna'
WHERE    Sid  = 1186
```

- Information processing in DBMS divides operation individual, indivisible operational logical units, called transactions.
- Transaction is one of the mechanisms for managing changes to the database.
- Any Application programs make use transactions to execute set of SQL operations when it is essential to complete all SQL operations at once.

3. Transaction example

- **For example**, during the transfer of money between two bank accounts it is unacceptable that updates the second account to fail. This would lead to the transferred money being lost. It will have been withdrawn from one account but not inserted into the second account.

```
BEGIN TRANSACTION transfer
    UPDATE accounts
    SET balance=balance - 100
    WHERE account=A

    UPDATE accounts
    SET balance=balance + 100
    WHERE account=B

If no errors then
    Commit Transaction
Else
    Rollback Transaction
End If
END TRANSACTION transfer
```

Fig. 3.1.3 : Sample implementation of Transaction in SQL

4. Transaction structure and boundaries

(a) Transaction structure

- The transaction consists of all SQL operations executed between the begin transaction and end transaction.
- Transaction is started by a BEGIN TRANSACTION command.
- Once this command is executed the transaction monitor starts monitoring the transaction.
- All operations executed after a BEGIN TRANSACTION command till END TRANSACTION are treated as a single large operation.

(b) Transaction boundaries

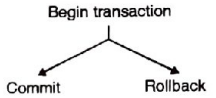
- When a transaction is completed it must either be committed, by executing a COMMIT command, or rolled back, by executing a ROLLBACK command.
- Until a transaction commits or rolls back the database remains unchanged to other users of the system.
- Therefore, the transaction can make as many changes as it wishes but none of the updates are reflected in the database until the transaction completes or fails.

(i) Commit transaction

- A transaction that is successful and has encountered no errors is committed by issuing commit.
- That is, all changes to the database are made permanent and become visible to other users of the database.

(ii) Rollback transaction

- A transaction that is unsuccessful and has encountered some type of error should be rolled back. That is, all changes to the database are undone and the database remains unchanged by the transaction.



- The DBMS guarantees that all the operations in the transaction either complete or fail. When a transaction fails all its operations are undone and the database is returned to the state it was in before the transaction started.

3.2 Distributed Database Transaction Processing

- A Distributed Database Management System (DDBMS) should be able to survive in a system failure conditions i.e. if a DDBMS is having a system failure then it should be possible to restart the system without losing any of the information contained in the database. This property can be offered by transaction processing as in centralised databases.
- In a distributed database system, any transaction can be either local or global transaction.
  - **(a) Local Transaction :** A local transaction is any transaction which works only on own site where it is located.
  - **(b) Global Transaction :** Any transaction designed for working on multiple sites is a Global transaction. Global transaction may be a set of multiple local transactions each runs at their own site. So each local transaction is sub part of global transaction.
  - **(c) Transaction Monitor :** Transaction is assigned to a Transaction Monitor (TM) at the time when the transaction first enters the distributed database system.
  - **(d) Distributed Commit :** A global site is set of many local transactions due to which committing a global transaction becomes more difficult.
- Distributed Transaction processing is designed to maintain a database in a consistent state even when data is distributed over multiple sites. It ensures that any operations carried out on the system that is interdependent.

- Transactions are either completed successfully or all aborted successfully.
- Transaction processing is meant for concurrent execution and recovery from possible system failure in a DBMS.

3.3 Transaction Properties (ACID Properties)

To understand distributed transaction properties we consider a transaction of transferring 100 rupees from account A to account B as below. Let T_i be a transaction that transfers 100 from account A to account B. This transaction can be defined as,

(a) Read balance of account A.

(b) Withdraw 100 rupees from account A and write back result of balance update.

(c) Read balance of account B.

(d) Deposit 100 rupees to account B and write back result of balance update.

T_i	Read (A); A := A – 100; Write(A); Read(B); B := B + 100; Write(B)

Fig. 3.3.1 : Sample Transaction

1. Atomicity

- Transaction must be treated as a single unit of operation.
- That means when a sequence of operations is performed in a single transaction, they are treated as a single large operation.
- In distributed environment there are many local transaction performing changes on same data items makes a single global transaction which should be executed completely or else all sites operation should be roll back.
- Examples,
  (a) Withdrawing money from your account.
  (b) Making an airline reservation.
- Execution of a distributed transaction should be either complete or nothing should be executed at all.
- No partial transaction executions are allowed. (No half done transactions)
- **Example :** Money transfer in above example.
- Suppose some type of failure occurs after Write (A) but before Write (B) then system may lost 100 rupees in calculation which may cause error as sum of original balance (A+B) in accounts A and B is not preserved (such situation is called as inconsistency explained later), In such case database should automatically restore original value of data items.
- In above case either all above changes are made to database or nothing should be done as half done transaction may leave data as incomplete state.
- If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.

2. Consistency

- If there are multiple sites performing changes on distributed data but only valid and correct changes should be reflected on distributed database.

- Consistent state is a state in which only correct data will be written to the database.
- If due to some reason, a transaction violates the database's data consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules.
- On the other hand, if a transaction is executed successfully than it will take the database from one consistent state to another consistent state.
- DBMS should handle an inconsistency and also ensure that the database is clean at the end of each transaction.
- Consistency guarantees that a transaction will never leave your database in a half finished (inconsistent) state.
- Consistency means if one part of the transaction fails, all of the pending changes made by that transaction are rolled back, leaving the database as it was before you initiated the transaction.
- **Example :** Money transfer in above example :
- Initially in total balance in accounts A is 1000 and B is 5000 so sum of balance in both accounts is 6000 and while carrying out above transaction some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation.
- As now sum of balance in both accounts is 5900 (which should be 6000) which is not a consistent result which introduces inconsistency in database.
- All the data involved into data operation must be in a consistent or correct state upon completion of the transaction; database integrity cannot be compromised. This means that during a transaction the database may not be consistent.

3. Isolation

- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- Isolation property keeps multiple transactions separated from each other until they are completed.
- Operations occurring in a transaction (example, insert or update statements) are invisible to other transactions until the transaction commits (on transaction success) or rolls back (on transaction fails).
- For example, when a transaction changes a bank account balance other transactions cannot see the new balance until the transaction commits.
- Different isolation levels can be set to modify this default behaviour.
- Transaction isolation is generally configurable in a variety of modes. For example, in one mode, a transaction locks until the other transaction finishes.
- Even though many transactions may execute concurrently in the system. System must guarantees that, for every transactions (T_i) other all transactions has finished before transactions (T_i) started, or other transactions are started execution after transactions (T_i) finished.
- That means each transaction is unaware of other transactions executing in the system simultaneously.
- **Example :** Money transfer in above example.
- The database is temporarily inconsistent while above transaction is executing, with the deducted total written to A and the increased total is not written to account B. If some other concurrently running transaction reads balance of account A and B at this intermediate point and computes A+B, it will observe an inconsistent value (Rs. 5900). If that other transaction wants to perform updates on accounts A and B based on the inconsistent values (Rs. 5900) that it read, the database may be left database in an inconsistent state even after both transactions have completed.
- A way to avoid the problem of concurrently executing transactions is to execute one transaction at a time.

4. Durability

- The results of a transaction that has been successfully committed to the database will remain unchanged even after database fails.
- Changes made during a transaction are permanent once the transaction commits.
- The distributed database handles durability by storing uncommitted transactions in a global transaction log. So that, a partially completed transaction won't be written to the database in the event of an abnormal termination.
- However, when the database is restarted after such a termination, it examines the global transaction log for completed transactions that had not been committed, and applies them.
- Once the execution of the above transaction completes successfully, and the user will be notified that the transfer of amount has taken place, if there is no system failure in this transfer of funds.
- The durability property guarantees that, once a transaction completes successfully, all the changes made by transaction on the database persist, even if there is a system failure after the transaction completes execution.

3.4 Model for Distributed Transaction Processing

Q. Draw and explain model of transaction management in DDB. **MU - May 15, 10 Marks**

Q. Transaction management model for distributed system. **MU - May 16, 10 Marks**

1. Introduction

- If transaction complete successfully it may be saved on to database server called as **committed** transaction.
- A committed transaction transfers database from one consistent state to other consistent state, which must persist even if there is a system failure.
- Transaction may not always complete its execution successfully. Such a transaction must be **aborted**.
- An aborted transaction must not have any change that the aborted transaction made to the database. Such changes must be undone or **rolled back**.
- Once a transaction is committed, we cannot undo its effects by aborting it.
- **Distributed Transaction Manager :** Responsible for coordinating the execution of the database operations on behalf of an application system.
- **Scheduler :** Responsible for the implementation of a specific concurrency control algorithm for synchronizing database access.
- **Local Recovery Manager :** It implements procedures which helps local database to recover from failure conditions.

2. Transaction states

A transaction must be in one of the following states :

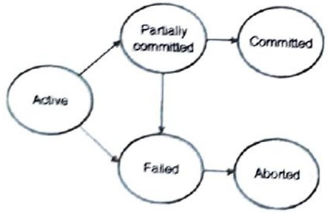


Fig. 3.4.1 : State diagram of a transaction

(a) Active

- This is initial state of transaction execution.
- As soon as transaction execution starts it is in active state and it informs transaction manager.
- Transaction remains in this state till transaction finishes.
- **Read :** If the data item to be read is distributed over multiple sites, transaction manager finds where the data item is stored and requests data from it.
- **Write :** If the data item to be read is distributed over multiple sites, transaction manager finds where the data item is located and requests the update operation to be carried out by taking care of concurrency.

(b) Partially committed

- As soon as last operation in transaction is executed transaction goes to partially committed state.
- At this condition the transaction has completed its execution and ready to commit on database server. But, it is still possible that it may be aborted, as the actual output may be their in main memory, and thus a hardware failure may prohibit its successful completion.

(c) Failed

- A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution.
- Example, In case of Hardware or logical errors occurs while execution.

(d) Aborted

- Failed transaction must be rolled back. Then, it enters the aborted state.
- The transaction manger makes sure that no effects of the transaction are reflected in any of the databases at the sites where it write data items
- Transaction has been rolled back restoring into prior state. In this stage system have two options :
- **Restart the Transaction :** A restarted transaction is considered to be a new transaction which may recover from possible failure.
- **Kill the Transaction :** Because the bad input or because the desired data were not present in the database an error can occurs. In this case we can kill transaction to recover from failure.

(e) Committed

- When last data item is written out, the transaction enters into the committed state.
- This state occurs after successful completion of transaction.
- A transaction is said to have terminated if has either committed or aborted.
- The transaction manager helps the sites involved in updating data items on behalf of this transaction so that the updates are made permanent at all site.

3.5 Transaction Types

1. Introduction

- Schedule is a sequence of instructions that specify the sequential order in which instructions of transactions are executed.
- A schedule for a set of transactions must consist of all instructions present in that transactions, it must save the order in which the instructions appear in each individual transaction.

- A transaction that successfully completes its execution will have to commit all instructions executed by it at the end of execution.
- A transaction that fails to successfully complete its execution will have to abort all instructions executed by transaction at the end of execution.

2. Representation

- We denote this transaction T as,
  $R_T(X)$: Denotes read operation performed by the transaction T on object X.
  $W_T(X)$: Denotes write operation performed by the transaction T on object X.
- Each transaction must specify its final action as commit or abort.

3.5.1 Serial Transactions / Schedules / Executions

1. Introduction

- This is simple model in which transactions executed in a serial order that means after finishing first transaction second transaction starts its execution.
- This approach specifies first my transaction, then your transaction other transactions should not see preliminary results.

2. Example

- Consider below two transactions T_1 and T_2.
- **Transaction T_1 :** deposits Rs.100 to both accounts A and B.
- **Transaction T_2 :** doubles the balance of accounts A and B.

T_1:	Read(A)
	A ← A + 100
	Write(A)
	Read(B)
	B ← B + 100
	Write(B)

T_2:	Read(A)
	A ← A*2
	Write(A)
	Read(B)
	B ← B*2
	Write(B)

- Serial schedule for above transaction can be represented as below.
- **Schedule A :** A consistent serial schedule
- A consistent serial schedule is obtained by executing T_1 right after T_2.

T_1	T_2	A	B	Operations
		25	25	Initial Balance
	Read(A); A ← A*2			
	Write(A)	**50**	25	
	Read(B); B ← B*2			
	Write(B)	50	**50**	
Read(A); A ← A + 100				

T_1	T_2	A	B	Operations
Write(A)		**150**	50	
Read(B); B ← B + 100				
Write(B)		150	**150**	
		150	150	Final Balance

- In above serial schedule for we can first execute transaction T_2 then T_1 which may results in some final values.
- Representation : $T_1 \rightarrow T_2$
- **Schedule B :** A consistent serial schedule
- A serial schedule that is also consistent is obtained by executing T_2 right after T_1.

T_1	T_2	A	B	Operations
		25	25	Initial Balance
Read(A); A ← A + 100				
Write(A)		**125**	25	
Read(B); B ← B + 100				
Write(B)		25	**125**	
	Read(A); A ← A*2			
	Write(A)	**250**	125	
	Read(B); B ← B*2			
	Write(B)	250	**250**	
		250	250	Final Balance

- Representation : $T_2 \rightarrow T_1$
- In above schedule T_1 is executed entirely then T_2 has started. Assume account A with Rs.25 and B with Rs.25. Then transaction T_1 will update A as 125 and B as 125. Now T_2 will read updated values of A and B. T_2 will update value of A as 250 and B as 250. The consistency constraint is A + B should remain unchanged. So at end of T_2, A + B i.e. 250 + 250 = 500 remains unchanged so execution of this schedule keeps database in consistent state.

3.5.2 Concurrent Transactions / Schedules / Executions

1. Introduction

- Transactions executed concurrently, that means operating system executes one transaction for some time then context switches to second transaction and so on.
- Transaction processing can allows multiple transactions to be executed simultaneously on database server.
- Allowing multiple transactions to change data in database concurrently causes several complications with consistency of the data in database.
- It was very simple to maintain consistency in case of serial execution as compare to concurrent execution of transactions.

2. Advantages of concurrency

Improved throughput

- Throughput of transaction is defined as the number of transactions executed in a given amount of time.
- If we are executing multiple transactions simultaneously that may increase throughput considerably.

Resource utilization

- Resource utilization defined as the processor and disk performing useful work or not (in idle state).
- The processor and disk **utilization** increase as number of concurrent transactions increases.

Reduced waiting Time

- There may be some small transaction and some long transactions may be executing on a system.
- If transactions are running serially, a short transaction may have to wait for an earlier long transaction to complete, which can lead to random delays in running a transaction.

3. Example

- Consider below two transactions T_1 and T_2.
- **Transaction T_1 :** deposits Rs.100 to both accounts A and B.

T_1 :	Read(A)
	A ← A + 100
	Write(A)
	Read(B)
	B ← B + 100
	Write(B)

- **Transaction T_2 :** doubles the balance of accounts A and B.

T_2 :	Read(A)
	A ← A*2
	Write(A)
	Read(B)
	B ← B*2
	Write(B)

- Above transaction can be executed concurrently as below.
- Schedule C : A schedule that IS NOT SERIAL but is still consistent
- Obtained by interleaving the actions of T_1 with those of T_2.

Table 3.5.1

T_1	T_2	A	B	Operations
		25	25	Initial Balance
Read(A); A ← A + 100;				
Write(A);		125		
	Read(A); A ← A*2;			
	Write(A);	250		
Read(B); B ← B + 100;				
Write(B);			125	
	Read(B); B ← B*2;			
	Write(B);			
		250	250	Final Balance

- In above given schedule Part of T_1 is executed which updates A to 125. Then processor switches to T_2 and part of T_2 which updates A to 250 is executed. Then context switch to T_1 and remaining part of T_1 which updates B to 250 is executed. At the end remaining part of T_2 which reads B as 125 and updates it to 250 by multiplying value of B by two. This concurrent schedule also maintains consistency of database as ultimately A + B is 250 + 250 = 500 (unchanged).
- The result of above transaction is exactly same as serial schedule shown in above Table. Therefore, above schedule can be converted to equivalent serial schedule and hence it is consistent schedule.

3.6 Transaction Management Architecture

Transaction processing components of a DBMS are listed below (refer Fig. 3.6.1) :

- **Transactions management:** The transaction manager is responsible for co-ordinating the execution of all the transactions being executed by the DBMS.
- **Scheduler :** The scheduler is responsible for selecting the best method of executing each transaction and ensuring that a transaction does not interfere with the execution of other transactions.
- **Recovery manager :** When a transaction fails and is rolled back, it is the recovery manager's responsibility to return the database to the state it was in before the transaction was executed.
- **Buffer manager :** The buffer manager manages the disc space and main memory that is required to process the transactions.

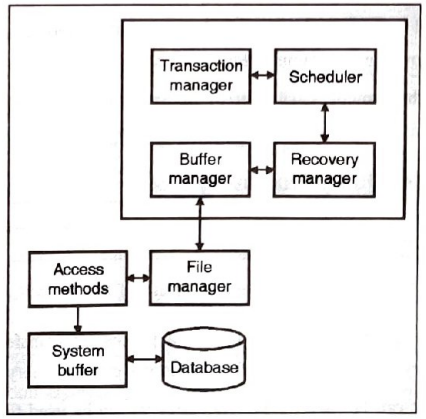


Fig. 3.6.1 : Transaction architecture

- A transaction is started by issuing a BEGIN TRANSACTION command. Once this command is executed the DBMS starts monitoring the transaction.
- All operations executed after a BEGIN TRANSACTION command are treated as a single large operation.
- Application programs use transactions to execute sequences of operations when it is important that all the operations are successfully completed.

Example :

- During the transfer of money between two bank accounts it is unacceptable for the operation that updates the second account to fail. This would lead to the transferred money being lost. It will have been withdrawn from one account but not inserted into the second account.

```
BEGIN TRANSACTION transfer
Deduct Rs.100 from account A
Add Rs.100 to account B
If no errors then
Save changes made by this operation
Else
Delete changes made by this operation
End If
END TRANSACTION transfer
```

Fig. 3.6.2 : Sample implementation of transaction in PL/SQL

- Transaction management component will ensure the atomicity and durability properties.
- To detect system failures and restore the database to the state that existed prior to the occurrence of the failure database system must therefore perform failure **recovery**.
- It is the responsibility of the **concurrency-control manager** to control the interaction between various concurrent transactions, to ensure the consistency of the database.

3.7 Distributed Databases Query Processing

Q. Explain query processing in distributed database. MU - Dec. 19, 5 Marks

1. Introduction

- The main function of a query processing involves a process to transform a high-level query into an equivalent low level query. It means mapping from relational calculus query to relational algebra query.
- The low-level queries (it can be relational calculus query) can be used to planning execution strategy of query.

- This query transformation should be able to achieve both correctness and efficiency.
- It is correct if the low-level query has the same semantics as the original high level query, that is, if both queries produce the same result.
- The well-defined mapping from relational calculus to relational algebra makes the correctness issue easy.
- For best Distributed Databases (DDBMS) it must optimize a query processing in terms of communication costs of processing a distributed query and other parameters.

2. Parameters for query processing

The following things may be considered while moving for query processing and optimization.

1. Data transfer cost
2. Distributed query processing using semi-join
3. Query and update decomposition
4. Query processing

3. Data transfer costs

- The main factor that complicates query processing is the cost of transferring data over the network.
- Data Transfer Stages,
  a) The intermediate data may be transferred to other sites for next set of data processing.
  b) After all processing final result has to be transferred to the site where the query result is actually needed.
- These costs may not be very high if the sites are connected via a high performance connecting network.
- DDBMS query optimization algorithms tries for reducing the amount of data transfer for optimization criteria.

3.7.1 Objectives of Distributed Databases Query Processing

1. Query transformation

The main function of a query processing involves a process to transform a high-level query into an equivalent low level query. It means mapping from relational calculus query to relational algebra query.

2. Data transfer costs / data communication cost

- The main factor that complicates query processing is the cost of transferring data over network.

  Data transfer stages,
  a) The intermediate data may be transferred to other sites for next set of data processing.
  b) After all processing final result has to be transferred to the site where the query result is actually needed.
- These costs may not be very high if the sites are connected via a high performance connecting network.
- DDBMS query optimization algorithms tries for reducing the amount of data transfer for optimization criteria.

3. Resource utilization

- Resources required for distributed databases must be utilised efficiently for better resource utilization.
- Response time of the query :
  - i) Time required for complete query execution.
  - ii) Various operators are executed in parallel at different sites, Hence response time of a query may be significantly less than its total cost.

Total cost :

i) Total cost is the sum of all times required for processing the query at various sites and inter communication.

ii) Total cost includes CPU, I/O and communication costs.

iii) The CPU cost is cost for performing operations on data in main memory.

iv) The I/O cost is the time required for disk accesses.

v) This cost can be reduced by minimizing the number of disk accesses with faster access methods and efficient buffer management.

vi) Now a days, we consider weighted average of CPU, I/O and communication cost.

3.8 Architecture of Distributed Query Processing : Characterization of Query Processors

The Query processing in centralized database is simple to understand as compare to query processing in distributed databases. The following are few characteristics first four applicable to both databases and remaining are only applicable for distributed databases.

1. Languages

- Input language can be relational algebra or relational calculus and output language can be relational algebra.
- The query processor must be able to convert input language to output language effectively.

2. Types of Optimizations

- The output language specifies the execution strategy of query.
- The optimization will be done by selecting best query execution strategy from exhaustive search or with help of query heuristics and with help minimal query tree.
- Exhaustive search targeting to find the best query plan from all available execution strategies with help of finding query cost os all strategies and select plan with minimum query cost.
- Query Heuristics are used to select best plan by regrouping query tree and performing selection, projection. It can also use semi join in place of join.
- The distributed database can use semi join for minimizing the cost of data transfer.

3. Optimization Time

- There are two timings where we can perform the optimization first before executing the query and second is while query execution.
- First timing is called as Static and second is dynamic optimization time.

(a) Static Optimization Time

- It is done at query compilation time
- This time is most suitable for exhaustive search method
- Size of query tree is not known in advance until run time, it can be just estimated by statistics.

(b) Dynamic Optimization Time

- It is done at query run time
- Database statistics are not required for estimating size of intermediate query tree
- It is required to repeat every time when query is running.

4. Statistics

- The effectiveness of query optimization mainly based on database statistics.
- Dynamic query optimization uses the database statistics to select relational operation must be completed first.
- The accuracy of statistics can be achieved by periodic updating.
- To minimize probability of error statistics can be used like histogram.

5. Decision Sites

The decision site is one which takes the decision in distributed databases.

(a) Central Decision Approach

- Single site that generates the best schedule i.e. best strategy of execution
- It needs the knowledge about entire distributed database

(b) Distributed Decision Approach

Cooperation among various sites to find out the database schedule

6. Exploitation of Network Topology

- Distributed query optimization be divided into global execution strategy and communication between participating sites.
- Selection of each local execution strategy based on centralised query processing.

WAN

- Point to point with higher communication cost will dominate over all other factors.
- Global schedule is used for minimizing communication cost.

LAN

- For smaller networks the IO cost is comparable to communication cost
- Special algorithm is existing for star networks
- Broadcast capability of some local area network can be exploited to optimize the processing of join operation.

7. Exploitation of replicated fragments

- Larger number of replicated fragments will produce larger number of query execution strategies.
- The exploitation of replicated fragments will increase number of messages and local processing by reducing cost of data transfer.

8. Use of Semi Joins

- The semi join will reduce the cost of data transfer in query execution strategies.
- The cost of join and data transfer will be reduced with semi join.

3.9 Layers / Phases of Distributed Query Processing

1. Introduction

- The problem of query processing can be decomposed into several phases; each phase addresses a well-defined sub problem.
- The input query is on global data expressed in terms of relational calculus.
- This data distribution is hidden for such queries.
- The first three steps map query into an optimized distributed query execution plan.
- First three steps performs query decomposition, data localization, and global query optimization. These steps are coordinated by a central control site and uses global directory for information.
- The fourth phase performs distributed query execution by executing the plan and returns the answer to the query. It is performed by the local sites along with the control site.

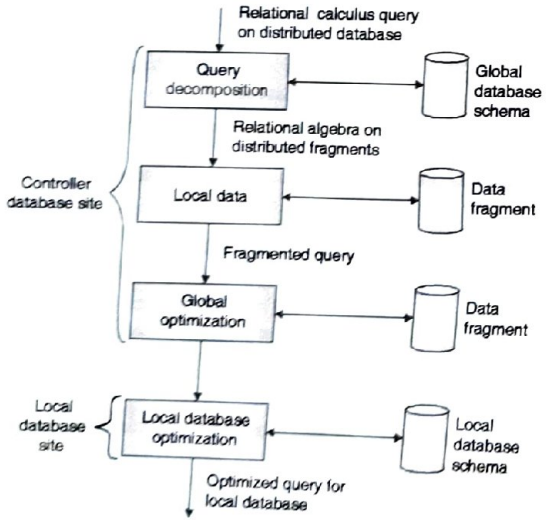


Fig. 3.9.1 : Phases of query processing

2. Steps of Query Processing

a. Query Decomposition
b. Data Localization
c. Global Query Optimization
d. Distributed Query Execution

a) Query decomposition

- This step will decompose the calculus query into relational algebra query on global relations.
- The information required for this transformation is found in the global conceptual schema represented for global relations.
- The techniques used by this site are same as that of a centralized DBMS.
- The relational algebra query generated is made better by avoiding worse executions.

Steps for query decomposition

1. Relational Calculus query is restructured in a more normalized form suitable for subsequent manipulation. Normalization of a query generally involves the query qualification by applying logical operator priority.
2. The normalized query is evaluated for incorrect queries. Generally, they use some sort of graph that captures the semantics of the query.
3. The correct query is simplified by eliminating redundant conditions on data.
4. The calculus query is updated as relational algebra query which are derived from the same calculus query.

b) Data localization

- This step determines which fragments are involved in the particular query and transforms the distributed query into a query designed for fragments.
- A global relation can be reconstructed by applying various operators and then deriving a program, called a localization program, of relational algebra operators, which then act on fragments.

In this step

1. Query is mapped into a fragment query by substituting in each relation.
2. Fragment query is simplified and restructured to produce another better query.

c) Global query optimization

- Query optimization is strategy to find an execution strategy for the query which is optimal.
- Query optimization finds ordering of operators in the query and communication operators which minimize a cost function.
- The cost of query referred as query execution time, disk space, disk I/Os, CPU cost, communication cost etc.
- The basic optimization involves distributed join operators using semi join operator. Semi join in a distributed system used to reduce the size of the join and communication cost.
- Optimization techniques may consider local processing costs and communication costs.
- The output of the query optimization is optimized algebraic query.

d) Distributed query execution

- Sub query executing at one site, called a local query.
- Local Query is optimized using the local schema of the site.
- Local optimization may use the algorithms of centralized systems.

Ex. 3.9.1 : Explain heuristic query optimization with given example.

SELECT e_lname
FROM employee e, works_on w, project p
Where p.name = " database" and p.number = w.pno and
e.essn = w.ssn and e.bdate >" 1997-12-31"

Soln. :

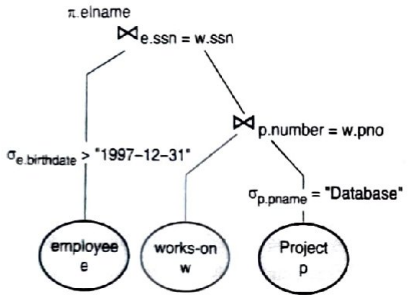


Fig. P. 3.9.1

Review Questions

Q. 1 What is transaction ? Discuss transition model.

Q. 2 List the ACID properties. Explain the usefulness of each in distributed databases.

Q. 3 Comment on two actions conflict if they operate on the same data object and at least one of them is a write action.

Q. 4 Explain detailed system architecture of database system.

Q. 5 Write a short notes on :

a. Query processor

b. Transaction management

c. Storage management

Q. 6 Describe working of detailed system architecture of DBMS.

Q. 7 What do you understand by transaction management?

❑❑❑

4 Distributed Concurrency Control

Syllabus

Distributed Concurrency Control : Taxonomy, Locking based, Basic TO algorithm,
Recovery in Distributed Databases : Failures in distributed database, 2PC and 3PC protocol.

4.1 Distributed Concurrency Control : Taxonomy

- In a distributed database only many users are accessing the data at same point of time.
- This means that the DDBMS is responsible for changes made to the database and its effects on other users.

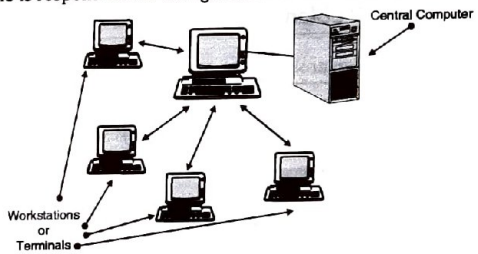


Fig. 4.1.1 : Concurrent database access

- Concurrency control is concerned with preventing loss of data integrity due to interference between users in a distributed environment.
- Concurrency control should provide a mechanism for avoiding and managing conflict between various database users operating same data item at same time.

4.2 Failure in Distributed Databases

1. Dealing with multiple copies of data items

- Maintaining consistency among multiple copies of distributed data is responsibility of this method.
- The recovery method is makes a copy consistent with other copies if the site on which it is stored fails.

2. Failure of individual sites

- When a site recovers from crash or failure their databases is out dated as compared to other sites.
- After site is up local database must be brought up to date with the rest of the sites.

3. Failure of communication site

- The system must be able to manage temporary failure of network that connects the multiple sites in distributed databases.
- In this case network partitioning may occur.
- This breaks up the one site into number of partitions.
- In this case sites within one partition can communicate only with another site in same partition and not with sites in other partitions.

4. Distributed commit

- Committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process then that type of problem is called as distributed commit.
- If some sites fail during the commit process then two phase commit protocol can be used.

5. Distributed deadlock

Distributed deadlocks may occur among several sites, due to

(a) Concurrency Problem : Multiple sites are accessing same system.

(b) Recovery problems.

4.3 Solutions for Concurrency Problems in Distributed Databases

Concurrency problem is very old and known problem which can be controlled with help of following two methods,

a) Maintaining a distinguished copy of a data item

b) Voting Method

4.3.1 Maintaining a Distinguished Copy of a Data Item

Introduction

- The idea is to designate a particular copy of each data item as a distinguished copy.
- The locks for data item are associated with this copy, and all locking and unlocking requests are sent to the site that contains that copy.
- There are three different ways of maintaining distinguishing copy of data :

1. Primary site method
2. Primary site with backup site
3. Primary copy method

1. Primary site method

Method

A single "Primary Site" is selected which contains distinguished copies for all database items. Hence, all locks are kept at this site and all requests are sent here.

Disadvantages

a) Primary site may be overloaded.

b) Failure of primary site brings down the entire system.

Advantages

a) Simple implementation.

b) No distributed deadlocks.

c) Once locks are accessed from the primary site, the data items can be accessed from any site.

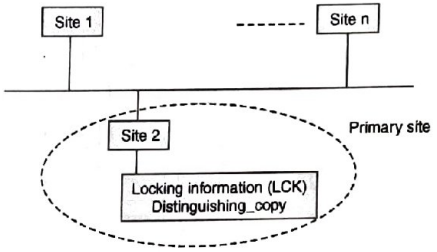


Fig. 4.3.1

2. Primary site with backup site

Method

- Apart from a primary site, a second site is selected as a backup site.
- All locking information is maintained at both the sites.

Advantages

a) If primary site fails, the backup site can take over and operation can be resumed after selecting another site as a backup site and copying all locking information at the new backup site.

b) Improved reliability.

c) More availability.

Disadvantages

It slows down the operation.

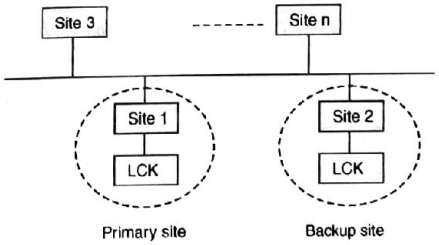


Fig. 4.3.2

3. Primary copy method

Method

a) The locking information is distributed, among various sites having distinguished copies.

b) Failure of one site will affect only transactions accessing locks on them whose primary or distinguishing copies reside at that site.

Advantages : It enhances reliability.

Disadvantages : But now in this case distributed dead lock may occur.

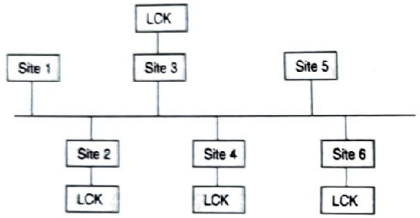


Fig. 4.3.3

4.3. 2 Voting Method

Introduction

There is no distinguishing copy, rather a lock request is sent to all sites that include a copy of the data item.

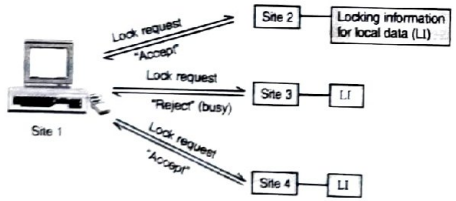


Fig. 4.3.4 : Voting method

Method

- In this method each site maintains its locking information for own copy of data items and this site can grant or deny the request.
- If a request R_1 from a transaction is granted by a majority (many sites) of the sites, it holds the lock and informs all sites that it has been granted the lock.
- Now onwards this site is locked for all other sites and no request will be accepted by this site.
- Site now starts replying to original request R_1.

Advantages

- No need of maintaining distinguishing copy.
- Any data request can be solved at that point of time only by taking reply (voting) from multiple sites.

Disadvantages

- It has higher network traffic among sites due to more replies from various sites.
- It is more complicated to implement as voting logic is also involved.

4.4 Concurrency Control Schemes

Q. Explain concurrency control in distributed database. **MU - May 19, 10 Marks**

- Isolation property of transaction ensures that each transaction must remain unaware of other concurrently executing transactions.
- In case of concurrent transactions, when several transactions are executing simultaneously on a database may not preserve isolation property for long time.
- To implement concurrent system there must be interaction among various concurrent transactions.
- This can be done by using one of the concurrency control schemes.
- All the following schemes are based on serizability of schedules.

4.5 Locking Based Concurrency Control

Q. Explain two concurrency control algorithms for a distributed database system. **MU - Dec. 15, 10 Marks**

Q. What are the various concurrency control techniques ? Compare lock based concurrency control strategies in detail. **MU - May 16 10 Marks**

- In concurrent environment, many users can access same data in a DBMS simultaneously; each has the feels that he is having exclusive access to the database.
- To achieve such system we must have interaction amongst those concurrent transactions which is also called as Mutual Exclusion which is required for concurrent executions of various transactions.
- Whenever one transaction is accessing data, second transaction should not change data otherwise there may be dirty read problem. This can be done with help of locking concept.
- Transaction can access data if it is locked by that transaction. Locking is necessary in a concurrent environment to assure that one process should not retrieve or update a record which another process is updating.

4.5.1 Types of Locks

a) Shared locks

- This type of locking is used by the DBMS when a transaction wants to only read data without performing modification to it from the database.
- Another concurrent transaction can also acquire a shared lock on the same data, allowing the other transaction to read the data.
- Shared locks are represented by S.
- SQL Implementation

LOCK TABLE customer IN

SHARED MODE;

b) **Exclusive locks**

- This type of locking is used by the DBMS when a transaction wants to read or write (i.e. performing update) data in the database.
- When a transaction has an exclusive lock on some data, other transactions cannot acquire any type of lock (shared or exclusive) on the data.
- Exclusive locks are represented by X.
- SQL Implementation

LOCK TABLE customer IN

EXCLUSIVE MODE;

- Lock Compatibility Matrix

		Lock by Transaction T_j	
		S	**X**
Lock by Transaction T_i	**S**	**No Conflict**	Conflict
	X	Conflict	Conflict

4.5.2 Two Phase Locking

Two-phase locking (2PL) synchronizes reads and writes by explicitly detecting and preventing conflicts between concurrent operations.

Before reading data item X, a transaction must "own" a read lock on X. Before writing into X, transaction must "own" a write lock on X.

The ownership of locks is governed by two rules :

a) Different transactions cannot simultaneously own conflicting locks (i.e. WR).

b) Once a transaction surrenders ownership of a lock, it may never obtain additional locks.

Phases of Locking

Growing Phase : In this phase, transaction may obtain locks but may not release any lock.

Shrinking Phase : Transaction may release locks but may not obtain any new one.

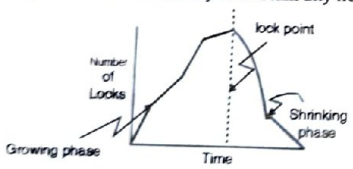


Fig. 4.5.1 : Locking Point

Working of 2P protocols :

- Initially transaction is in 'growing phase', it acquires locks as needed. Once it starts releasing locks it enters into the 'Shrinking Phase' and now it may not acquire any lock after shrinking phase starts.
- The point at which transaction obtains final (last) lock is called as lock point of transaction.

Modified Versions of Two-phase Locking Protocol :

1. **Strict two-phase locking protocol**

   It requires not only two-phase locking but also that all exclusive-locks held by transaction should be held until that transaction commits or abort.

   This property ensures that if data is being modified by one transaction (holding lock-X) then other transaction can't read it until first transaction commits.

2. **Rigorous two-phase locking protocol**

   It requires that all share and exclusive locks to be held until the transaction commits. So transactions can be serialized in the sequence they commit.

3. **Conservative 2-P locking protocol (Static 2PL)**

   This scheme requires locking all items needed to access before the transactions starts. It begins execution by declaration about Read set and Write Set of all data items needed in advance.

   Read Set : Set of all data transactions lock.

   Write Set : Set of all data than n-calls transactions lock.

   If any one item of above list is currently not available for locking then lock will not be granted it waits till all items are ready for locking.

4.5.3 Distributed Two Phase Locking

- In distributed databases, transaction may starts transaction in system at Site 1, but requests a lock from Site 3. At the same time, a conflicting transaction may enter the system at Site 2 and request a lock from Site 4.
- If there isn't any coordination in Site 4 and Site 3 then, conflict may not be detected, which results in inconsistency.
- In order to solve above issues distributed database needs some manager to handle this issue.

a. Centralized 2PL

- In this approach we select one site as lock manager (CLM). All other sites know where the CLM is located.
- Each site acquires LOCK using 2PL rules issued by centralized lock manager.
- To request a lock, local site sends a lock request to the CLM not to local lock manager. If that data item is available, then CLM locks the item and sends a "LOCK Grant" message to requested site.
- If the data item not available due to one or more conflicting transactions, the lock is not granted "LOCK Reject" is sent to respective site.
- After granting lock, Local transaction manager (TM) will make changes to database and inform about updated data items to LCM so that it will take care of making changes at all replicated databases in distributed environment.
- Once task is done Local TM issues the request to release locks.
- Once lock is released LCM will takes care that it no other Local TM will acquire lock again to maintain 2PL.

b. Primary copy 2PL

- In primary copy 2PL multiple sites are selected as control centers.
- Each site takes responsibility of managing a number of locks.

- Example, one site takes responsibility of locking rows in the STUDENT table, while another site manages the locks for rows in the DEPARTMENT table.

c. Distributed 2PL

- Distributed 2PL requires Local lock manager (LM) to manage the data items stored at its own site. Lock manager depends on the data distribution and/or replication.
- There are three alternatives :

i) No replication

- Distributed database maintains no replicated data items.
- Now a site where the only copy of the data item present will acts like lock manager for the item.TM will decide which copy to access or read and also makes sure that a write operation is done for all copies of data item.
- Once a transaction is done, a TM notifies the CLM to release lock and enters to second phase of locking.

ii) Full replication

- We select one site as the lock manager for each data item.
- Responsibility of managing the locks for all items is given to one site, then approach works like centralized 2PL.

iii) Partial replication

- Only few data items in the system may be replicated.
- For the non-replicated data items, the site where the data item located acts like lock manager.
- For the replicated data items, one site is selected as the lock manager for each item which is responsible to acquire the lock from each data item's lock manager.

4.6 Timestamp Based Concurrency Control

Q. Explain Timestamp-based concurrency control mechanisms in DDB. **MU - May 15, 10 Marks**

Q. Explain the basic timestamp ordering algorithm. **MU - May 15, 10 Marks**

- To achieve serializability order of transactions for execution can be decided by its time at which transaction entered in system.
- A general method to achieve this is using time stamp ordering protocol.
- A fixed timestamp is assigned at start of execution of the transaction. Every transaction T_j has been assigned a timestamp by database system denoted as TS (Tj). A transaction which has entered in system recently will have greater timestamp. If transaction Tj starts after T_i then,

$$TS(T_i) < TS(T_j)$$

- Every data item X is with two timestamp values :

(a) W-timestamp (X) : Timestamp of last transaction that executed WRITE (X) successfully on given data item That mean it is timestamp of recent WRITE(X) operation.

(b) R-timestamp (X) : Timestamp of last transaction that executed READ (X) successfully on data item (X).That mean it is timestamp of recent READ (X) operation.

4.6.1 Centralised Timestamp - Ordering Protocol (Basic TO Algorithm)

For transaction Ti to execute READ (X) Operation,

1. If TS (T_i) < W - timestamp (X) Then T_i is trying to read value of X that is overwritten by other transaction.

   So, this READ is rejected (as T_x is already performing write operation on data so no other operation can be performed by any other transaction) and T_i is rolled back.

W - timestamp (X) = 149 (Recent Write)			
TS	**T_i**	**T_x**	**Operations**
148	READ (X)	...	TS (T_i) < W - timestamp (X) i.e. 148 < 149 (W-Timestamp)
149	Unlock(X)	WRITE (X)	Record W - timestamp (X) = 149

2. If TS (T_i) ≥ W - timestamp (X) : Then READ executed and set

   R-timestamp(X) = max {TS (T_i), R-timestamp}

TS	**Ti**	**Tx**	**Operations**
148	....	WRITE (X)	Record W - timestamp (X) = 148 (recent write)
149	READ (X)	....	TS (T_i) ≥ W - timestamp (X) i.e. 149 ≥ 148 (W-Timestamp) Record R-timestamp(X) = 149
150	....	....	....
151	READ (X)	....	TS (T_1) ≥ W - timestamp (X) i.e. 151 ≥ 148 (W-Timestamp) Record R-timestamp(X) = max{149,151} = 151

If transaction T_i execute WRITE (X) Operation,

1. If TS (T_i) < R - timestamp (X) : Then Ti has produced value of X which is not needed now so rollback T_i.

TS	**T_i**	**T_x**	**Operations**
148	....	...	...
149	WRITE(X)	....	TS (T_1) < R - timestamp (X) i.e. 149 < 151 (R-Timestamp) Reject WRITE roll back T_i
150	....	....	....
151		READ (X)	Record R-timestamp(X) = 151 (Recent READ Operation)

2. If TS (T_i) < W - timestamp (X) : Then T_i is trying to write obsolete value of X, So rollback T_i

TS	T_i	T_x	Operations
148	....	...	...
149	WRITE (X)	...	TS (T_1) < W - timestamp (X) i.e. 149 < 151 (W-Timestamp) Cannot write as other transaction using data X for writing.
150	....	...	....
151		WRITE (X)	Record W-timestamp(X) = 151

3. Otherwise, system executes WRITE (X) and sets
W-timestamp (X) = TS (T_i)

TS	T_i	T_x	Operations
150	....	....	....
151	WRITE (X)		Record W-timestamp(X) = 151

4.6.2 Distributed Timestamp - Ordering Protocol

1. Introduction

- The timestamp concurrency control algorithms in centralized systems can be easily be extended for distributed database.
- This method totally rely on timestamp for resolving any types of conflicts, timestamp can be a good way to indicate relative age of each transaction in system.

2. Working

- In distributed database system, we can make use of logical clock (LC) which increments like transaction timestamp.
- But, this logical clock (LC) for local site may not be unique at global level. Hence, in case of distributed system, we use logical clock given by local site and also site ID as a timestamp of a transaction for uniqueness.
- Clock synchronization is required only if transactions at S_1 and S_2 are in conflict, then TMs at these sites will have to communicate with each other.
- LC at any local site is given as,

  LC = Max(Local LC, LC received with the message)
- Implementing Time Stamping algorithm in a distributed database is not cost effective due to the substantial number of messages required for read, pre-write, and also for commit phase of the algorithm.

3. Types

- Basic Timestamp Ordering Algorithm.
- Conservative Timestamp Ordering Algorithm.
- Multisession concurrency Control Algorithm.

4.7 Commit Protocol

4.7.1 Two Phase Commit Protocol

Q. Explain Two Phase Commit Protocol in detail. MU - May 19, 10 Marks

1) Introduction

- In database transaction processing two phase commit protocol (2PC) is a type of an atomic commit protocol.
- This is a distributed algorithm that coordinates in all the distributed transaction whether to *commit* or *rollback* the transaction.

2) Purpose of using two phase protocol

- Ensures commit or rollback
- The two-phase commit protocol ensures that all participating database servers receive and implement the same action (commit or to roll back a transaction), despite of local or network failure.
- Automatic recovery mechanism
- The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction.

3) Sites used in two phase protocol

- Coordinator site
- The transaction manger at the site where the transaction originated is called a coordinator site.
- Subordinator site (or cohorts)
- The transaction managers at sites where sub transactions execute are called subordinates.

4) Two phases of working

The two phases of the protocol are the *commit-request phase and commit phase.*

- Commit request phase
- In this phase a *coordinator* site attempts to prepare all the subordinate transaction's and take the necessary steps for either committing or aborting the transaction.
- Commit phase
- The commit phase is based on voting ("Yes" means commit and "No" means abort) of the subordinates (cohorts).
- The coordinator decides whether to commit (only if all vote "Yes") or abort the transaction (otherwise) from above voting and then notifies the result to all subordinates.
- The subordinate site then follows with the actions (commit or abort).

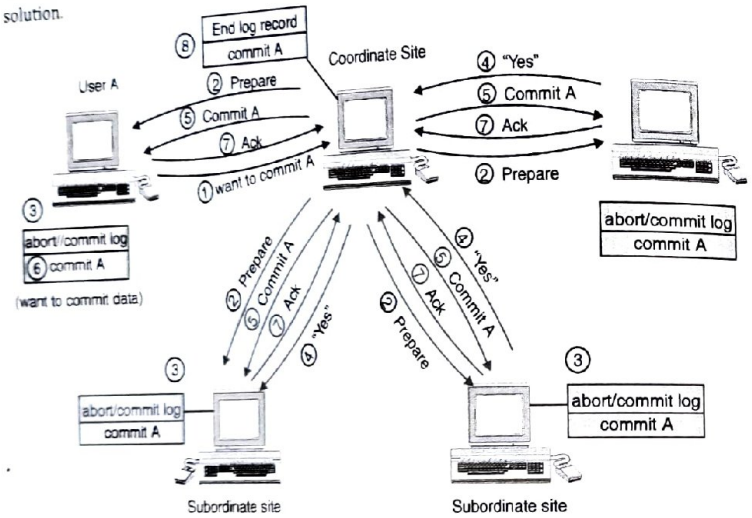
5) Example

- When user decides that he needs to commit the transaction, the commit command is sent to the coordinator and this will initiates two phase protocol.
- The coordinator site will sends a **prepare message** to all subordinates
- Upon receiving prepare message, the subordinate decides whether to abort or commit its sub transaction.
- It writes an abort or prepare log record and **sends no or yes message** to the coordinator.
- **Commit message :**If coordinator receives yes message from all subordinate, writes to commit log record and sends a commit message to all subordinates.
- **Abort message :** If it receives even one, **no** message from some subordinate for a specified time out interval, it writes abort log record and sends an abort message to all subordinates sites.
- Based on the message received by the subordinates it writes **commit/abort record** and Commit/abort sub transaction and sends acknowledge message to the coordinator.

- After receiving acknowledge message from all the subordinates, the coordinator site will writes an **end log record** for the transaction.
- A transaction is officially committed at the time the coordinator's commit log record is written.

6) Advantages

- The protocol performs well even in many cases of temporary system failure (failures like process, network node, communication etc.) and hence it is widely utilized.
- It is not flexible to all possible failure configurations, and in rare cases user intervention is needed for solution.



(1) User A wants to commit transaction
(2) Coordinator sends prepare message to all subordinates
(3) each subordinate site prepare abort/commit log
(4) Each site subordinate replyies to coordinate by "yes" or "no"
(5) If all of Reply "yes" transaction
(6) Write commit log to all subordinate site
(7) each subordinate give Acknowledgment to coordinator
(8) Coordinator writes end log record

Fig. 4.7.1 : Two phase commit protocol

4.7.2 Three Phase Commit Protocol

Q. Explain Three Phase Commit Protocol in detail. **MU - Dec. 19, 10 Marks**

1) Introduction

- Adjacent state means that this state can go other state with a single state transition.
- The Three Phase Commit Protocol (3PC) is designed for avoiding blocking state in distributed recovery algorithm which is possible with above two phase commit protocol.

2) Three phase commit concept

- Transaction is in non blocking state if its state transition diagram contains none of the following,
- Graph does not have any state that is adjacent to both a commit and an abort state.
- No non-committable state (adjacent to a commit state)
- If any process is in this state, we know that all the sites have voted "Yes" for committing the transaction. Such states are called committable.
- **READY state** is a non committable state, due to this state does not means that all the processes have voted "Yes" to commit the transaction.
- It is obvious that the WAIT state in the coordinator and the READY state in the participant 2PC protocol violate the non-blocking conditions we have stated above.
- Modification for 2PC protocol to satisfy the condition for non-blocking protocol.
- Add another state between the WAIT state-COMMIT states and READY state-COMMIT state which serves like buffer state where the process is ready to commit but has not yet committed.
- This is called the Three-Phase Commit Protocol (3PC) because there are three state transitions from the INITIAL state to a COMMIT state.

3) Sites used in two phase protocol

- Coordinator site
- The transaction manger at the site where the transaction originated is called a coordinator site.
- Subordinator site (or cohorts)
- The transaction managers at sites where sub transactions execute are called subordinates.

4) Three phases of working

The three phases of the protocol are the Pre commit request phase, commit-request phase and commit phase.

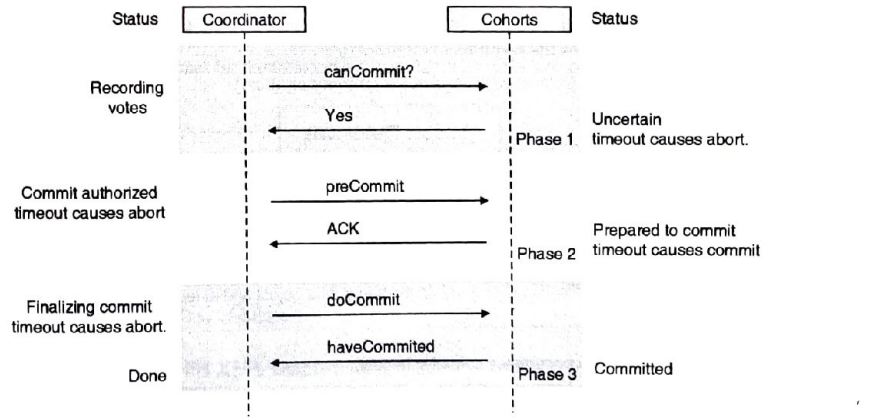


Fig. 4.7.2

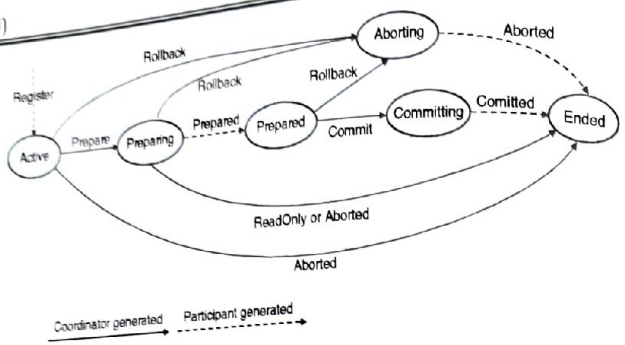


Fig. 4.7.3

(a) Pre commit request phase

- In this state the coordinator send message to all participants for pre-pairing to commit and coordinator site may not know if the non-responding participants have already moved to the PRECOMMIT state.
- However, it knows that they are at least in the READY state, which means that they must have voted to commit the transaction.
- The coordinator can therefore move all participants to PRECOMMIT state by sending a "prepare-to-commit" message in order to globally commit the transaction by writing a commit record in the log and sending a "global-commit" message to all the working participants.

(b) Commit request phase

In this phase a coordinator site attempts to prepare all the subordinate transaction's and take the necessary steps for either performing global commit or aborting the transaction.

(c) Commit phase

- The commit phase is based on voting ("Yes" means commit and "No" means abort) of the subordinates (cohorts).
- The coordinator decides to commit only if all votes "YES" or abort the transaction if any one participant reply "NO" and then notifies the result to all subordinates.
- The subordinate site then follows with the actions (commit or abort).

Review Questions

Q. 1 What is transaction ? Discuss transition model.

Q. 2 Define Serializability for distributed databases.

Q. 3 List the ACID properties. Explain the usefulness of each in distributed databases.

Q. 4 Comment on two actions conflict if they operate on the same data object and at least one of them is a write action.

Q. 5 Give various anomalies for concurrent executions.

Q. 6 Explain various problems for concurrency control schemes.

Q. 7 Describe various schemes to concurrency in distributed databases.

Q. 8 Explain locking based concurrency control in distributed databases.

Q. 9 Explain Timestamp based concurrency control in distributed databases.

□□□

5 XML Databases

Syllabus

Document Type Definition, XML Schema, Querying and Transformation: XPath and XQuery.

5.1 Introduction to XML Documents

Q. What is XML ? **MU - Dec. 19, 5 Marks**

1. Introduction

- XML is a mark-up language is very much like HTML but XML was designed to carry (transfer) data and not to display data.
- XML stands for Extensible Mark-up Language which gives a mechanism to define structures of a document which is to be transferred over internet.
- The XML defines a standard way of adding element to documents. Hence, XML is used for structured documentation.
- Unlike HTML, XML tags are not predefined one can define their own tags in XML.

2. Goals of XML

- XML should be directly used over the Internet. Users must be able to view XML documents easily as like HTML documents. This may be possible only when XML browsers are as robust and widely available as HTML browsers.
- XML should support a wide variety of applications.
- It should be Easy to write programs and process various XML documents.
- The minimum number of optional features in XML as it causes more confusion in programmers mind.
- XML documents should be logically clear.
- The design of XML shall be formal and concise and can be prepared very fast.
- XML documents shall be easy to create

5.2 Well Formed and Valid XML Documents

Q. What is well formed and valid XML document ? **MU - May 12, 10 Marks**

5.2.1 Well Formed Document

XML Documents which satisfy below given rules are called as well formed XML documents,

(a) XML document must start with an XML declaration to indicate the version number of XML being used all other relevant attributes.

```
<? xml version="1.0" standalone ="Yes"?>
```

(b) A well-formed XML document should be syntactically correct.

(c) Conditions for Tree Model / syntactically correct XML document

- There should be only one root element.
- Every element is included in an identical pair of start and end tags within the start and end tags of the parent element.
- Above conditions will ensure that the nested elements specify *a well-formed tree structure.*

(d) User defined Tags

An element within XML document can have any tag name as specified by user. There is no predefined set of tag names that a document knows.

5.2.2 Valid XML Documents

XML Documents which satisfy below given conditions are called as valid XML documents,

(a) Conditions

- The document must be well formed.
- Element used in the start and end tag pairs must follow the specified structure.
- This structure is specified in a separate XML DTD (Document Type Definition) file or XML schema file.

(b) DTD Syntax

- Start with a name is given to the root tag of the document.
- Then the elements and their nested structures are specified in top down fashion as below example.
- More details of this topic are given in section DTD set 5 of chapter

```
<! DOCTYPE company [
<! ELEMENT Employee (ID, Name, DeptNo proj) Company
<! ELEMENT ID (#PCDATA)
<! ELEMENT Name (#PCDATA)
<! ELEMENT DeptNo (#PCDATA)
<! ELEMENT project (Name, Number)
<! ELEMENT Name (#PCDATA)
<! ELEMENT Number (#PCDATA)
] >
```

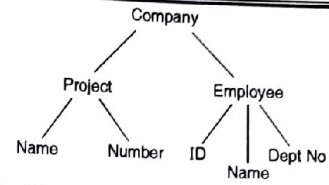



Fig. 5.2.1 : An XML DTD file called company

5.3 Structure of XML Data – Tree Data Model

1. Introduction

In the XML document the basic object is XML and it can be represented as hierarchical data model or tree data model.

2. Structuring concepts used to construct an XML document

(a) Element

- An element is a group of tags data values that can contain character data, child element or a mixture of both.
- Element can be of two type Simple and Complex.
- The elements are constructed from other elements by nesting them is called as complex element.
- The elements contain data values are called as Simple elements.

(b) Attributes

Additional information that describes elements.

(c) There are some additional concepts used in XML, such as entities, identifiers, and references.

3. A major difference between XML and HTML

- **Tag names :** In HTML tag names are used to describe how text is to be displayed and in XML tag names are defined to describe the meaning of the data elements in the document.
- **Processing :** With help of user defined tags it is possible to process the data elements in the XML document automatically using computer applications.

4. Tree Representation

XML Model is made of two elements :

(a) Complex elements are shown with help of internal nodes.

(b) Simple elements are shown by Leaf nodes.

Hence, XML Model is called as Tree Model or hierarchical model.

5. Example

(a) Tree Representation

Simple Elements : <Price>, <amount>

Complex elements : <Drink>, <Snack> etc.

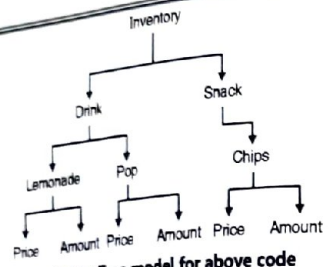


Fig. 5.3.1 : Tree model for above code

(b) Textual Representation

- Whenever value of the STANDALONE attribute in an XML document is set to "YES", such XML document is known as schema less XML documents.

E.g.

```
<inventory>
    <drink>
        <lemonade>
            <price>$2.50</price>
            <amount>20</amount>
        </lemonade>
        <pop>
            <price>$1.50</price>
            <amount>10</amount>
        </pop>
    </drink>
    <snack>
        <chips>
            <price>$4.50</price>
            <amount>60</amount>
        </chips>
    </snack>
</inventory>
```

5.3.1 Types of XML Documents

(a) Data-centric XML documents

- The document contains many small data items that follow a specific structure and hence it may be extracted from a structured database are called as data centric XML document.
- In order to exchange and display data over internet the document is formatted as XML documents.

(b) Document-centric XML documents

- The document contains large amounts of text, such as news articles or books are referred as document centric XML documents.

- There are only few structured data elements in these documents. Sometimes there are no data elements in such documents.

(c) Hybrid XML documents

- If both types of data are used simultaneously in document then it is referred as hybrid XML document.
- In such documents some parts that contain structured data and other parts that are predominantly unstructured.

5.3.2 XML – Structured Data

- Data centric XML documents sometimes considered as semi structured data or sometime considered as structured data.
- Document is considered as structured data if an XML document is written as per predefined XML schema or DTD.
- Document is considered as semi structure if an XML allows documents that do not conform to any particular schema.

5.4 XML Document Type Definition (DTD)

1. Introduction

- The way by which we describe a valid syntax of XML document by listing all the elements occur in the document which elements can occur in combination? How elements can be nested ? What attributes are available for each element type ? and so on.
- DTD describes the rules for analyzing an XML document.
- A DTD ensures that the contents of an XML document conform to expect rules that document should follow.

Example

```
<! DOCTYPE BOOK [
<! ELEMENT Book (Author, Title, Chapter>)
<! Element Author (# CDATA)>
<! Element Title (# PCDATA)>
<! Element Chapter (# PCDATA)>
<! ATTLIST Chapter # REQUIRED>]>
```

i) <! DOCTYPE Book >

The DTD starts with the above line.

It indicates that this DTD corresponds to an XML document has Book as root element.

ii) <! Element Book (Author, Title, Chapter)>

This line indicates first element in our XML document would be Book and Book element is the parent of sub-element namely, author, title and chapter.

iii) **<! Element Author (# CDATA)>**

This tag specifies that element Author is type of CDATA character data.

iv) **<! ELEMENT Title (# PCDATA)> and <! ELEMENT Chapter (# PCDATA)>**

This tag specifies that element Title and Chapter are of type PCDATA - passed character data.

v) **<! ATTLIST chapter # REQUIRED>**

This line specifies that id is an attribute of the element chapter. The #REQUIRED declarative, specifies that id is must.

2. Types of DTD as per location of DTD

A DTD is either contained in <!DOCTYPE> tag of same file as XML document or contained in an external file referenced from a <!DOCTYPE> tag or both.

(a) Internal DTD : DTD Embedded in XML Document

(b) External DTD : DTD as a separate external file

3. Types of DTD Declarations

(a) Element Type Declaration
(b) Attribute list declaration
(c) Entity Declaration
(d) Notation Declaration

(a) Element Type Declarations

- An element declaration defines one of the kinds of elements you can use, that is, one of the tag types.
- Example
- Suppose a <speech> element can contain any mixture of regular text, and text tagged with the elements <loud> and <soft>

```
<!ELEMENT speech ((#PCDATA|loud|soft)*)>
<!ELEMENT loud (#PCDATA)>
<!ELEMENT soft (#PCDATA)>
```

Notations

Let, A is element in XML document.

A * (optional multi valued (repeating) element)

Element name **A*** means that the element A can be repeated zero or more times in the XML document.

A + (required multi valued (repeating) element)

Element name **A+** means that the element A can be repeated one or more times in the XML document.

- **A ?** (optional single-valued (non repeating) element)
- Element name **A?** Means that the element A can be repeated zero or one times in the XML document.
- **A** (Required single-valued (no repeating) element)

- An element appearing without any of the preceding element must appear exactly once in the XML document.

A1 | A2 (Bar Symbol)

Only one out of *A1* and *A2* can appear in the XML document.

(A) ***Parentheses*** can be nested when specifying elements

(b) Attribute List Declaration

- Attribute list declaration identifies which elements have attributes they may have attributes, values and optional attributes.
  - Parts
    - (i) Name
    - (ii) Type
    - (iii) An optional default value
  - Example

```
<! ATTLIST vehicle_kind (car|truck|bike) #REQUIRED>
<!ATTLIST vehicle_kind (car|truck|bike) "car">
```

In above example first indicates vehicle_kind is not null or required attribute while second indicate vehicle_kind have 'car' as default value.

(c) Entity Declaration and Notation Declaration

Element declaration associates a name with same fragment of content, such as a piece of regular text, a piece of DTD or a reference to an external file containing text or binary data.

E.g. <!ENTITY DH "Xavier's Institute">

The process of unparsed entities is responsibility of application. Some information about the entities internal format must be declared after the identifier that indicates the entity location.

E.g. <!ENTITY Logo "XavierLogo.jpg" NDATA JPEG Format>

4. Advantages of DTD

(a) A DTD allow an XML parser to validate an XML document against its definition. This allows the online validation of an XML document before it is processed.

(b) A DTD can extend the capabilities of an XML document by allowing further processing possibilities.

(c) A DTD serves as an automotive documentation on XML document.

(d) A DTD can be used to validate data against the business rules. These business rules would be defined in the DTP and the incoming data can be validating against those.

5. Disadvantages of DTD

(a) It is written in a different (non XML) syntax.

(b) It has no support for Namespaces.

(c) It only offers extremely limited data typing.

5.5 XML Document Schema

Q. With example explain what is XML schema file ? MU - May 12, 10 Marks

Q. Explain XML schema document with example MU -Dec. 19, 5 Marks

5.5.1 Introduction

- The XML schema language is used as a standard language for specifying the structure of various XML documents.
- It works on same syntax rules as regular XML documents, so that only one processor can be used on both the programmes to compile.

5.5.2 Types of XML Documents

(a) XML instance document

A regular XML document is also called as instance document.

(b) XML schema document

A document that specifies a structure of tags present in XML document called as XML schema document.

XML schema can be shown by the *tree data model*, with elements and attributes as the main concepts of XML document structure. However there are some more concepts are also involved in XML.

5.5.3 An XML Schema Defines

(a) Elements present in a document, which elements are child elements, order of element and number of child elements and whether an element is empty or can include text.

(b) Attributes present in a document, define data types for elements and attributes and default and fixed values for elements and attributes.

5.5.4 XML Schemas as the Replacement to DTDs

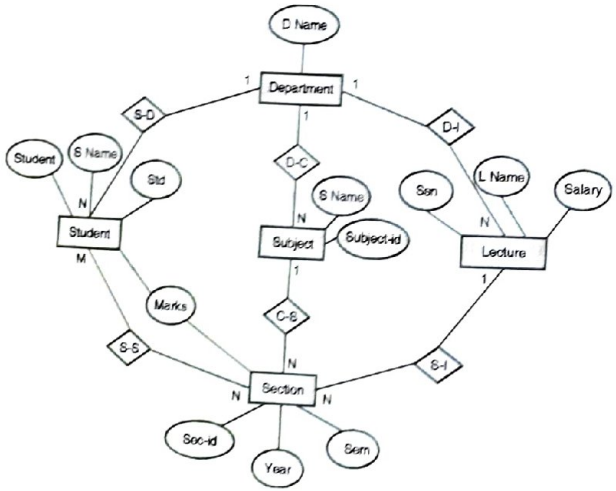
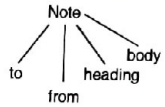


Fig. 5.5.1 : "Note" Tree structure

- XML Schemas are extensible and more powerful than DTDs.
- XML is ready for future developments and additions.
- It allows user-defined types to be created unlike HTML.
- XML Schemas are written in XML syntax and XML Schemas support xml data types.
- XML can allow complex data types to be extended with help of concept called as inheritance.
- XML also permits constraints like uniqueness constraint and foreign key constraint.



Schema for above "Note" tree structure

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.abc.org/2009/XMLSchema">
<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

5.5.5 XML Schema Features and Key Elements

(a) Schema descriptions and XML namespaces

It is necessary to identify the specific XML tags that being used to specify a file which is stored at a Web site location.

XML namespace : The above line in XML schema specifies commonly used standard such individual definitions are also called as XML namespace.

```
<xs:schema xmlns:xs="http://www.abc.org/2009/XMLSchema">
```

XML namespaces defines the set of commands or names that can be used.

XSD (XML Schema Description) will assign file name to the variable using the attribute called xml ns (XML namespace), now this variable is used as a prefix to all XML tag names.

(b) Annotations, documentation, and language used

- xsd : annotation, xs: documentation is used for providing comments in the XML document while writing xml documentation.

- In xml documentation attributes xml : lang element will specify the language being used, where "en" stands for the English language.

```
<xsd : annotation> <xsd : Documentation xml : lang = "en"> note
Schema </xsd : Documentation> </xsd : annotation> .
```

(c) Elements and types

- In XML schema there is root element present in every XML schema.
- The name of attribute xsd : element tag specifies the element name In our example note is the root element.
- The structure of the "note" root element can then be specified, which in our example is xsd : complex Type.

(d) First-level elements in the Note database

- The / character is used to indicate closing of tag.

```
<elementname = "body"/ >
```

- This indicates that the element title is not divisible further. The '/ ' at the end indicates that the 'body' is a single element that doesn't have any sub-element or attributes underneath.
- We specify the three first-level elements under the "note" root element. These elements are named to, from, and header and body are specified in an xs: element tag.

(e) Specifying element type and minimum and maximum occurrences

This indicates that the body tag expects string values must occur only once and cannot contain nulls. Also, the default value for the tag is No body. (Empty msg).

```
<element name = "body"
  type = "xsd : string"
  minOccurs = "1"
  maxOccurs= "2"
  default = "No title"
  nullable = "1"/>
```

(f) Specifying key

In XML we can specify constraints that correspond to primary key or unique constraints in RDBMS, as well as foreign keys constraints.

xsd : unique

- Unique attributes in a database specifies elements that correspond to unique values and may not be primary key.
- We can give name to each unique constraint.

Xsd : selector and xs:Field tags

These tags are used to select element type that contains the unique element and the element name which is unique using the xpath attribute.

Xsd : key ref tag

- Tags xs : selector and xsd : field specifies the referencing attribute in XML document while specifying foreign key.
- XSD : key ref specifies the referenced primary key.

(g) Specifying the structures of complex elements via complex types

- The last part of our example specifies the structures of the complex elements "note" using the tag xs: complex Type.
- Database attributes of each entity corresponds to sequence of sub elements by using the xsd : sequence and xsd : element tags of XML schema.
- Each element in schema is given a name and type with help of attributes name and type of xsd : element tag.

Ex. 5.5.1 : Consider a employee management database which maintains entries for employees in a company. Employees may be programmers, managers, designers and testers. Appropriate information is to be maintained for each employee along with their address, salary, etc. (You can make any other reasonable assumptions)

I. Give the DTD for the XML schema for the described system.

II. Write the following query in XQuery.

"Find programmers who have worked in projects coding at least two different languages in one year."

MU - Dec. 15, 10 Marks

Soln. :

```
<Company>
<Department>
        <Did>10</Did>
        <Dname>HR</Dname>
        <Project>

        <Employee>
            <Name>Bhavana</Name>
            <Salary>25000</Salary>
            <Designation>Programmer</ Designation>
            <Technology>C++</ Technology>
            <Technology>Java</ Technology>
            <Technology>C#</ Technology>
        </Employee>
        <Employee>
            <Name>Geeta</EmployeeName>
            <Salary>20000</EmployeeSalary>
            <Designation>Manager</ Designation >
        </Employee>
        .....
</Project>
</Department>
<Department>
        <Did>20</Did>
        <Dname>TIS</Dname>
        <Employee>
```

```
        ...
    </Employee>
    <Employee>
        ....
    </Employee>
</Department>
    ......
</Company>
<! DOCTYPE Company [
<! ELEMENT Department (Did,Dname,Project+)>
<! ELEMENT Did (#PCDATA) >
<! ELEMENT Dname(#PCDATA) >
<! ELEMENT Project (Pname, Employee+)>
<! ELEMENT Pname (#PCDATA) >
<! ELEMENT Employee(Ename, Designation, Technology*) >
<! ELEMENT Ename(#PCDATA) >
<! ELEMENT Technology (#PCDATA)>
<! ATTLIST Did CDATA #REQUIRED>
] >
```

Find programmers who have worked in projects coding at least two different languages in one year.

```
FOR     $x IN doc (www.company.com/info.xml)/company/Department/Project/employee
WHERE   $x/Designation EQ'Programmer' and count($x/Technology)>2
RETURN <res>
            $y/Name
            </res>
```

Ex. 5.5.2 : University database contains information about the course and the professors who teach the courses in each semester. Each course must also have information about the number of student enrolled, room no. data and time (when and where the course is conducted)

(i) Write DTD rules for above XML documents.

(ii) Create an XML schema for above XML documents.

MU - May 15, 10 Marks

Ans. :

1) Student Document structure

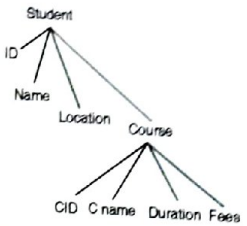


Fig. P.5.5.2 : Student document structure

2) Writing xml DTD for above ERD

```
<! DOCTYPE student [
<! ELEMENT students (ID, Name, Location, course+)>
<! ELEMENT ID (#PCDATA)>
<! ELEMENT Name (#PCDATA)>
<! ELEMENT Location (#PCDATA)>
<! ELEMENT course (CID, Cname, Duration, Fees)>
<! ELEMENT CID (#PCDATA)>
<! ELEMENT Cname (#PCDATA)>
<! ELEMENT Duration (#PCDATA)>
<! ELEMENT Fees (#PCDATA)>
]>
```

3) Writing XML Schema for above ERD

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd"http://www.abc.org/2009/XMLSchema">
<xs:element name="student">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ID" type="xs:string"/>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Location" type="xs:string"/>
            <xs:element name="Course" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="CID" type="xs:string"/>
                        <xs:element name="Cname" type="xs:string"/>
                        <xs:element name="Duration" type="xs:string"/>
                        <xs:element name="Fees" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

4) Writing xml Document based on above DTD or Schema

```
< Student >
    < ID > 100 < /ID >
    < Name > Mahesh < /Name >
```

```
< Location > Worli < /Location >
< Course >
    < CID > 10 < /CID >
    < CName > JAVA < /CName >
    < Duration > 100 < /Duration >
    < Fees > 10000 < /Fees >
< /Course >
....
< ID > 101 < /ID >
< Name > Harshad < /Name >
< Location > Worli < /Location >
< Course >
    < CID > 15 < /CID >
    < CName > ADBMS < /CName >
    < Duration > 200 < /Duration >
    < Fees > 15000 < /Fees >
< /Course >
....
< /Student >
```

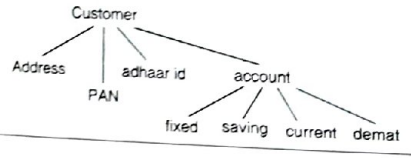
Ex. 5.5.3 : A banking database could contain the customers information along withthe types of accounts customer is maintaining. Customer information is its full profile information along with his current address, PAN ID, adhaar card no. included and account information should include type of account (Saving, fixed, demat, recurring, current), date and time of access and the transactions details.

(i) Write the DTD rules for the above XML documents.

(ii) Create an XML schema for the above XML document.

MU - May 16, 10 Marks

Soln. :




```
<! DOCTYPE Bank [
    <! ELEMENT Customer (Pan, Uid, Address,Project)>
    <! ELEMENT Pan (#PCDATA) >
    <! ELEMENT Uid (#PCDATA) >
    <! ELEMENT Address (City, State, country)>
        <! ELEMENT City (#PCDATA) >
        <! ELEMENT State (#PCDATA) >
        <! ELEMENT Country (#PCDATA) >
    <! ELEMENT Accounttype (Fixed|Saving|Demat|Current) "Saving">
```



```
    <! ELEMENT type (#PCDATA) >
    <! ATTLIST Uid CDATA #REQUIRED>
] >
```

3) Writing XML Schema for above ERD

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd"http://www.abc.org/2009/XMLSchema">
<xs:element name="customer">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Pan" type="xs:string"/>
            <xs:element name="UID" type="xs:string"/>
            <xs:element name="Address">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="City" type="xs:string"/>
                        <xs:element name="State" type="xs:string"/>
                        <xs:element name="pin" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="UID" type="xs:string"/>
            <xs:element  name="Account">
                <xs:simpleType>
                    <xs:restriction  base="xs:string">
                        <xs:enumeration  value="Current"/>
                        <xs:enumeration  value="Saving"/>
                        <xs:enumeration  value="Fixed"/>
                </xs:simpleType>
</xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
            </xs:restriction>

</xs:schema>
```

5.6 XML Transformation to Relational Model

1. Introduction

- Multiple relationships among the various entities can be used represent complex subset with one or more cycles.
- In presence of such cyclic graphs will complicates decision that how to create the document hierarchies.
- To indicate multiple relationships we need additional duplication of entities.

2. Example

- In below example STUDENT is a root element.
- Fig. 5.6.1 gives a possible hierarchical tree structure for STUDENT document.
- In Fig. 5.6.2 part one is not a tree structure due cycles are present in it.

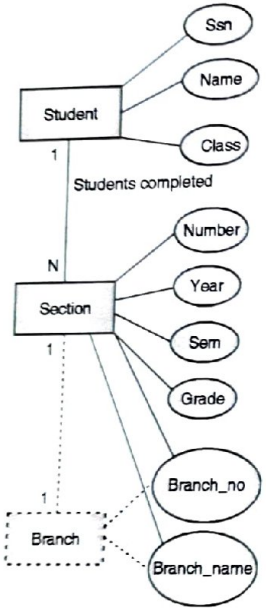
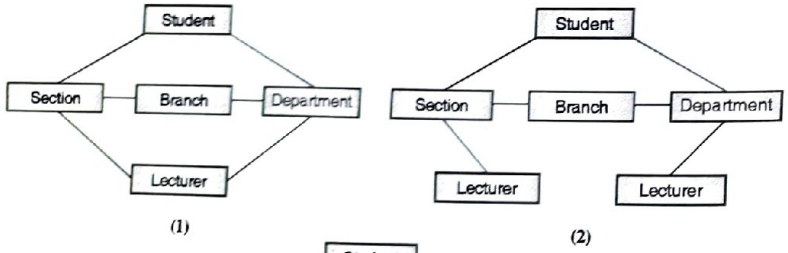


Fig. 5.6.1 : Hierarchical tree structure for STUDENT



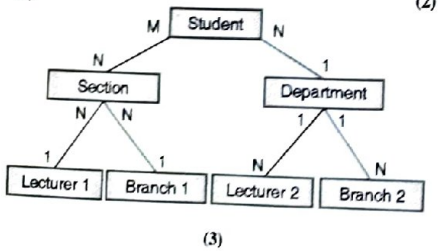


Fig. 5.6.2

Solution :

- Break the cycles by replicating the all entity types which contains cycle.
- Start replicating LECTURER entity first Fig. 5.6.2 part (2), calling the replica to the left of LECTURER 1 entity.
- The LECTURER replica on left will show the relationship between the sections they teach and himself,
- The LECTURER 2 replica on right will show the relationship between departments for which they works and themselves.
- Now see Graph structure is converted to tree structure.

```
<xs: element name="root">
<xs: sequence>
    <xs: element name="student" minOccurs="0" maxOccurs="unbounded">
    <xs: sequence>
    <xs: element name="ssn" type="xs: string" />
    <xs: element name="name" type="xs: string" />
    <xs: element name="class" type="xs: string" />
    <xs: element name="section" minOccurs="0" maxOccurs="unbounded">
    <xs: sequence>
    <xs: element name="Number" type="xs: unsignedInt" />
    <xs: element name="year" type="xs: string" />
    <xs: element name="sem" type="xs: string" />
    <xs: element name="Branch_no" type="xs: unsignedInt" />
    <xs: element name="Name" type="xs: string" />
    <xs: element name="Grade" type="xs: string" />
    </xs: sequence>
    </xs: element>
    </xs: sequence>
    </xs: element>
</xs: sequence>
</xs: element>
```

5.6.1 Other Steps for Extracting XML Documents from Databases

1. For creating XML schema document or XML hierarchy, below mentioned steps can be needed to extract a particular XML document from available database :
   (a) We can write accurate query in SQL which will extract the required information from the XML document.
   (b) It may be from XML tree structure or flat relational form the retrieved query result must be structured.
   (c) We can select either a single object or multiple objects into the document by customizing the query.

   E.g.

   We can create a simple document from single student with help of query that can select a single student entity then we can create document of that single student or it may select several or all the students and create a document with multiple students.

5.7 XML Querying

There have been several proposals for XML query languages. But two of them are widely accepted by user as follows :

(a) XPath

This construct used to provides the language constructs by writing path expressions in query to identify certain nodes (elements) within an XML document that satisfies some conditions.

(b) XQuery

This is a general query language used in XML. XQuery may use XPath expressions but with some additional constructs.

5.7.1 XPath : Specifying Path Expression

- An XPath may results in a collection of element that satisfies certain patterns as specified in the path expression.
- The names in the XPath are nothing but node names in the XML document. That can be either tag (element) names or attribute names.
- Two main separators are used when specifying a path :

(a) Single slash (/) :

Whenever there is single slash before any tag that tag must appear as a direct child of the parent tag of tag before it. e.g. /company/department - department is direct child of company.

(b) Double slash (//) :

Whenever a double slash is used before tag that can be appear as a descendant of tag before it at any level. e.g. //Employee - may be located at any level.

Examples

Consider following company xml document,

```
<Company>
    <Employee>
        <Name> Bhavana</EmployeeName>
        <Salary>25000</EmployeeSalary>
    </Employee>
    <Employee>
        <Name> Geeta</EmployeeName>
        <Salary>20000</EmployeeSalary>
    </Employee>
    ……
    <Department>
        <Did>10</Did>
        <Dname>HR</Dname>
    </Department>
    <Department>
        <Did>20</Did>
        <Dname>TIS</Dname>
    </Department>
    ……
</Company>
```


(a) /Company

The first XPath returns the company root node and all its descendant nodes, which means that it returns the whole XML document. We should note that it is customary to include the file name in the XPath query.

XML Document : Output for above path Expression

```
<Company>
    <Employee>
        <Name> Bhavana</EmployeeName>
        <Salary>95000</EmployeeSalary>
    </Employee>
    <Employee>
        <Name> Smita</EmployeeName>
        <Salary>85000</EmployeeSalary>
    </Employee>
    <Employee>
        <Name> Sagar</Name>
        <Salary>75000</Salary>
    </Employee>
    <Employee>
        <Name> Geeta</Name>
        <Salary>20000</Salary>
    </Employee>
    ……
    <Department>
        <Did>10</Did>
        <Dname>HR</Dname>
    </Department>
    <Department>
```

```
    <Did>20</Did>
    <Dname>TIS</Dname>
  </Department>
  .....
</Company>
```

E.g. COMPANY XML document is stored at the location ***www.company.com/info.xml*** then the above expression can be written as doc ***(www.company.com/info.xml)/company***

(b) /company/department

- This returns all department nodes (elements) and their descendant subtrees.
- Note that the nodes (elements) in an XML document are ordered, so the XPath result that returns multiple nodes will do so in the same order in which the nodes are ordered in the document tree.

Output :

```
<Department>
    <Did>10</Did>
    <Dname>HR</Dname>
</Department>
<Department>
    <Did>20</Did>
    <Dname>TIS</Dname>
</Department>
.....
```

(c) //company/department

- The expression uses //, which is convenient to use if we do not know the full path name we are searching for, but do know the name of some tags of interest within the XML document.
- This is particularly useful for schemaless XML documents.

Output :

```
<Department>
    <Did>10</Did>
    <Dname>HR</Dname>
</Department>
<Department>
    <Did>20</Did>
    <Dname>TIS</Dname>
</Department>
.....
```


(d) //employee [Salary gt 70000]/Name

- This query will return all emp1oyeeName nodes that are direct children of an emp1oyee node, such that the employee node has another child element employeeSalary whose value is greater than 70000.
- This expression should return the same result as the previous one, except that we specified the full path name in this example.
- This illustrates the use of qualifier conditions, which restrict the nodes selected by the XPath expression to those that satisfy the condition.

Output :

```
<Name> Bhavana</Name>
<Name> Geeta</Name>
<Name> Sagar</Name>
.....
```

(e) /company/employee [Salary ge 25000]

This expression returns all employees nodes and their descendant nodes that are children under a path /company/employee and have child node employeeSalary with a value greater than or equal to 25000.

Output :

```
<Employee>
    <Name> Bhavana</Name>
    <Salary>95000</Salary>
</Employee>
<Employee>
    <Name> Smita</Name>
    <Salary>85000</Salary>
</Employee>
<Employee>
    <Name> Sagar</Name>
    <Salary>75000</Salary>
</Employee>
```

5.7.2 XQuery : Specifying Queries

1. Introduction

(a) XPath allows us to select nodes from a tree-structured XML document with help of some expression.

(b) XQuery allows writing of more general queries on one or more XML documents.

(c) XQuery is also known as a FLWR expression, which is initials of four main clauses of XQuery and has the following form :

```
FOR     <variable bindings to individual nodes (elements)>
LET     <variable bindings to collections of nodes (elements)>
WHERE   <qualifier conditions>
RETURN  <query result specification>
```

2. XQuery Comparisons

In XQuery there are two ways of comparing values.

a) General comparisons: =, !=, <, <=, >, >=

b) Value comparisons: eq, ne, lt, le, gt, ge

The difference between the two comparisons methods are shown below.

Expression 1 : Expression returns true if any attributes have a value greater than 10

```
$bookstore//book/price > 10
```

Expression 2 : Expression returns true if there is only one attribute returned by the expression, and its value is greater than 10.

If more than one q is returned, an error occurs :

```
$bookstore//book/price gt 10
```

3. Examples

(a) Retrieves the employee employee names of employees who earn more than $70,000.

```
FOR     $X IN doc (www.company.com/info.xml)
        //employee [Salary gt 70000]/Name
RETURN  <res>
            $x/Name/firstName,
        $x/Name/lastName
        </res>
```

Output :

```
<Name> Bhavana</Name>
<Name> Smita</Name>
<Name> Sagar</Name>
```

(b) Alternative way to retrieve the same elements retrieved by first query.

```
FOR     $x IN doc (www.company.com/info.xml)/company/employee
WHERE   $x/Salary gt 70000
RETURN  <res>
        $x/Name
        </res>
```


(c) This query retrieves the employee Name and total hours worked by all employees working on project No. 5 (Join Operation in XQuery)

```
FOR     $x IN doc (www.company.com/info.xml)/company/
            Project [projectNumber = 5]/projectWorker,
            $y IN doc (www.company. com/info.xml)
            /company/employee
WHERE   $x/hours gt 20.0 AND $y.ssn = $x.ssn
RETURN  <res>
        $y/Name
        $x/hours
        </res>
```

4. Applications

(a) Fetching information from a database for any use in web service.

(b) Generating summary reports for analysis on data stored in an XML database.

(c) Searching textual documents for extracting relevant information and compiling the results.

(d) Selecting and transforming XML data to XHTML to be published on the Web.

(e) Pulling data from databases to be used for the application integration.

(f) Splitting up an XML document that represents multiple transactions into multiple XML documents.

5.8 XML Applications

MU - Dec. 15, 10 Marks

Q. Give two applications of XML.

1. Problem Statement for XML Application

The hotel provides menu which contains food types like Chinese, Punjabi, South Indian food and Drinks. Each item has Name, cost, calories and veg or nonveg type. Write XML schema for above menu card.

2. Designing Tree Model for XML

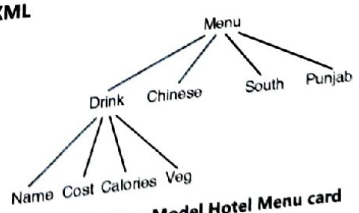


Fig. 5.8.1 : Tree Model Hotel Menu card

3. Designing Data Model For XML

```
<? xml version="1.0" encoding="UTF-8"?>
<Emps ID="1">
<ename> Ganesh </ename>
<Children>
    <name> Manesh </name>
    <Birthday>
        <day> 21 <day>
        <month> 05 </month>
        <year> 2000 </year>
    </Birthday>
</Children>
<Skill>
    <type> C </type>
    <exam>
        <year> 2007 </year>
        <city> Pune </city>
    </exam>
</Skill>
</Emps>
<Emps ID="2">
<ename> Mahesh </ename>
<Children>
    <name> Ganesh </name>
    <Birthday>
        <day> 1 <day>
        <month> 2 </month>
        <year> 2001 </year>
    </Birthday>
</Children>
<Children>
....
</Children>
<Skill>
```



```
    <type> JAVA </type>
    <exam>
        <year> 2004 </year>
        <city> Mumbai </city>
    </exam>
    <exam>
    ....
    </exam>
</Skill>
<Skill>
....
</Skill>
</Emps>
....
<Emps ID="10">
<ename> Manish </ename>
<Children>
    <name> Suresh </name>
    <Birthday>
        <day> 21 <day>
        <month> 04 </month>
        <year> 2007 </year>
    </Birthday>
</Children>
<Children>
    <name> Ash </name>
    <Birthday>
        <day> 21 <day>
        <month> 04 </month>
        <year> 2007 </year>
    </Birthday>
</Children>
<Skill>
    <type> HTML </type>
```

```
    <exam>
        <year> 2004 </year>
        <city> Goa </city>
    </exam>
<Skill>
</Emps>
```

4. Designing DTD (Document Type Definition)

```
<! DOCTYPE Employee [
<! ELEMENT Emps (ename,Children+, Skill+)>
<! ELEMENT ename (#PCDATA) >
<! ELEMENT Children (name, Birthday+)>
<! ELEMENT name (#PCDATA) >
<! ELEMENT Birthday (day, month, year)>
<! ELEMENT day (#PCDATA) >
<! ELEMENT month (#PCDATA) >
<! ELEMENT year (#PCDATA) >
<! ELEMENT Skill (type, exam+)>
<! ELEMENT type (#PCDATA) >
<! ELEMENT exam (year, city) >
<! ELEMENT year (#PCDATA) >
<! ELEMENT city (#PCDATA) >
<! ATTLIST EMPS ID CDATA #REQUIRED>
] >
```

5. Designing XSD (XML Schema Definition)

```
<? xml version="1.0" encoding="UTF-8" ?>
<xs: schema xmlns ="http:// www.2014Schema.com/xmldata">
<xs: Annotation">
      <xs: documentation XML: long = "en" > Menu
      <xs: documentation>
< /xs: Annotation>

<xs: element name = "Emps">
      <xs: complextype>
```



```
      <xs: sequence>
<xs: element name = "ename" type= "xs: string"/>

        <xs: element name="Children" >
            <xs: complextype>
                <xs: sequence >
                    <xs: element name = "name" type= "xs: string"/>
                    <xs: element name="Birthday" >
                        <xs: complextype>
                        <xs: sequence >
<xs: element name = "day" type= "xs:int"/>
                            <xs: element name="Month" type="xs:int"/>
                            <xs: element name="year" type = "xs:int"/>
                        </xs:sequence>
                        </xs:complextype"/>
                    </xs: element >
                </xs:sequence>
            </xs:complextype"/>
        </xs: element >
        <xs: element name="Skill" >
            <xs: complextype>
                <xs: sequence >
                    <xs: element name = "type" type= "xs: string"/>
                    <xs: element name="exam" >
                        <xs: complextype>
                        <xs: sequence >
<xs: element name = "year" type= "xs:int"/>
<xs: element name="city" type="xs:String"/>
                        </xs:sequence>
                        </xs:complextype"/>
                    </xs: element >
                </xs:sequence>
                </xs:complextype"/>
        </xs: element >
```

```
        </xs:sequence>
        <xs: attribute name = "ID" type= "xs: string" use="required" />
    </xs:complextype"/>
</xs:element >
```

Review Questions

Q. 1 Explain structured, semi-structured, and unstructured data ?

Q. 2 What do you mean by self-describing data ?

Q. 3 Give various conditions for wellformness of XML documents and valid XML documents.

Q. 4 Explain the XML hierarchical data model ?

Q. 5 What is DTD? Give Example of DTD for any XML Document ?

Q. 6 What is the difference between XML schema and XML DTD ?

Q. 7 What is the difference between data-centric and document-centric XML documents ?

Q. 8 Write a short note on :

a) XQuery b) Xpath

c) XML schema d) XML documents

□□□

6 JSON

Syllabus

Basic JSON syntax, (Java Script Object Notation),JSON data types,

Stringifying and parsing the JSON for sending & receiving, JSON Object retrieval using key-value pair and JQuery, XML Vs JSON.

6.1 Introduction to JSON (Java Script Object Notion)

- JSON stands for JavaScript Object Notation.
- JSON is basically used for data transmission in the web applications.
- JSON is the extension of the JavaScript language.
- JSON can be used for storing and exchanging data.
- JSON can be used as an alternative to XML.
- JSON is a lightweight, text based data-interchange format. Text can be read and used as a data format by any programming language.
- JSON is language independent and can be used with modern programming languages. All popular programming languages have parsing code and support for generating JSON data.
- JSON is self-describing and easy to understand.
- JSON has file name extension .json.
- JSON supports name-pair values, arrays, list, vector, sequences etc.

6.2 Comparison JSON and XML (XML vs JSON)

Q. Compare JSON & XML with example.

Q. Define Basic JSON syntax.

1. Similarity between XML and JSON

- Both languages are known as the **self-describing languages**
- They are hierarchical (values within values) and represented in tree form
- Both of them is used by lots of programming languages

2. **Main differences between XML and JSON**

- XML uses tags whereas JSON does not uses any tag
- JSON files are simple and small to read and code

3. **Basic Syntaxes of XML and JSON**

The lot of difference's observed in syntax of JSON and XML

Basic JSON Syntax

```
{"movies" : [{"Name" : "Tom and Jerry", "year" : "2000"},
{"Name" : "John Carter", "year" : "2014"}]}
```

XML Code for the same Program

```
<movies>
  <movie>
  <Name>Tom and Jerry</Name>
    <year>2000</year>
  </movie>
  <movie>
  <Name>John Carter</Name>
    <year>2014</year>
  </movie>
</movies>
```

From the above two codes it is clear that JSON requires less efforts to write the same code as compared to XML.

6.2.1 JSON Data Types

1. **Number :**

- Numbers are defined as a double precision floating-point format.
- Octal formats, hexadecimal formats, NaN, Infinity are not used.

Example : var jnum = {marks : 69}

Here jnum is the json object name given by user.

2. **Boolean :**

Boolean is used to indicate either true or false values.

Example : var jnum = {name : 'Sachin', marks : 69, Firstclass : true}

3. **null :**

Null means empty.

Example :

```
var c = null;
if(c==1)
{
    document.write("Sachin");
}
else
{
    document.write("not Sachin");
}
```

4. **JSON value :**

Json values may include string, boolean value, inerger, float,null etc.

Example :

var x=2;

var c = "Sachin";

var d = null;

5. **String :**

1. As in character only a single element can be accepted, the string length for character is 1.
2. String is a sequence of characters.

Example : var jnum = {name : 'Sachin'}

6. **Array :**

Array is a collection of similar data type elements. Array value generally starts with 0 and ends with n-1. Here n is the number of elements in the array.

Example : Array containing multiple objects in which movies is an array.

```
{
  "movies" : [
  { "movie" : "Hanuman" , "type" : "cartoon" },
  { "movie" : "JohnCarter" , "type" : "mystery" },
  ]
}
```

7. **Object :**

- Object uses name/value pairs.
- All the keys should be different from each other.
- For arbitrary strings Objects are generally used.

Example :

```
{
   "screen" : "05",
   "movie" : "John Carter",
   "ticket" : 200,
}
```

Here screen, movies, ticket atc. are different keys / names with different values.

8. **White space :**

Whitespaces are used between token pairs to make code more readable.

Example : var c= "Sachin";

6.3 JSON Schema

Q. What is the rule for writing JSON ? Differentiate between JSON and XML. **MU - May 19, 8 Marks**

With JSON Schema, structure of JSON data can be described. With JSON Schema it is very easy to validate something. It is suitable for automation testing. Clear and easy documentation can be generated for human as well as machine.

JSON schema validation libraries

Currently the most reliable and commonly used JSON Schema validator is JSV.

Languages	Libraries
Java	json-schema-validator (LGPLv3)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
C	WJElement (LGPLv3)
Python	Jsonschema
.NET	Json.NET (MIT)
PHP	php-json-schema (MIT), json-schema (Berkeley)

JSON schema example :

```
{
  "$schema" : "http : //json-schema.org/draft-04/schema#",
  "title" : "movies",
  "description" : "Famous movies",
  "type" : "object",
  "properties" : {
```



```
    "screen" : {
      "description" : "The screen number of the movie",
      "type" : "integer"
    },
    "name" : {
      "description" : "MovieName",
      "type" : "string"
    },
    "ticket" : {
      "type" : "number",
      "minimum" : 0,
      "exclusiveMinimum" : true
    }
  },
  "required" : "screen", "name", "ticket"]
}
```

Above schema can be used to test the validity of the below given JSON code :

```
[
  {
    "screen" : 01,
"name" : "Hanuman",
    "ticket" : 200,
  },
  {
"screen" : 02,
"name" : "John Carter",
    "ticket" : 300,
  }
]
```

Note : These are standard keywords hence we cannot change it more. Keywords and description are written according to flow of the program.

Important schema keywords :

Keywords	Description
$schema	Schema is written according to the draft v4 specification.
title	Schema Title

Keywords	Description
description	Schema short description
type	Defines the first constraint on JSON data : it has to be a JSON Object.
properties	Various keys and their value types, minimum and maximum values to be used in JSON file.
required	Required properties.
minimum	Minimum acceptable value.
exclusiveMinimum	The instance is valid if it is strictly greater than the value of "minimum".
maximum	Maximum acceptable value.
exclusiveMaximum	The instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	Maximum number of characters in a string
minLength	Minimum number of characters in a string
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

6.4 Stringify and Parsing JSON for sending and receiving

Q. How to Stringify JSON for sending and receiving. **MU - May 19, 8 Marks**

1. Introduction

- A common use of JSON is to data exchange to/from a web server.
- In order to send data to a web server, the data has to be a string.
- The JSON syntax is exactly like JavaScript's object syntax, but JSON object cannot be assigned to a variable.
- JSON file represents the data.
- JSON is nothing but a string of text, it needs to be converted to an object to use inside JavaScript.
- similarly, JavaScript objects need to be converted into string for using as JSON data.
- Example,

2. Two functions for working with JSON are built into JavaScript:

- *JSON.parse()* - This function is used toconvert a JSON string into a JavaScript object.
- *JSON.stringify()* - This function is used toconvert a JavaScript object into a JSON string.

3. Send JSON Data from the Client Side

- To send data from client we create a JavaScript object.
- Use JSON.stringify() to convert the JavaScript object into a JSON string.


```
const obj = {name: "Mahesh", age: 36, city: "Mumbai"};
const myJSON = JSON.stringify(obj);
```

- Send the encoded URL - JSON string to the server as part of the HTTP Request.
- The task can be performed by the HEAD, GET, or POST method by assigning the JSON string to a variable.
- It can also be sent as raw text using the POST method.

4. Receive JSON Data on the Server Side

1. To receive data we convert the incoming JSON string to an object using a JSON parser.
2. The methods available based upon which parser you are using for conversion.
3. We can perform any task of data received.

Sample JSON File

```
<!DOCTYPE html>
<html>
<body>
<h2> Create a JSON string</h2>
<p id="demo"></p>
<script>
const obj = {name: "Mahesh", age: 36, city: "Mumbai"};
const myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>
</body>
</html>
```

Output JSON File

```
{"name":"Mahesh","age":36,"city":"Mumbai"}
```

6.5 JSON Object Retrieval using key value pair and JQuery

1. Introduction to Key Value Pair

- The JSON object can be represented as key value pair.
- The sample JSON document and code to get values from the JSON document is given below.

```
{
"score": 97,
"subject": "Maths",
"mode": "Written"
}
```

2. Retrieval of JSON Object using Key Value Pair

- The JSON object can be read and retrieved using simple key value method.
- The JSON object can be represented as key value pair.

```
var score = object.get("score"); // score = 97
var mode = object.get("mode"); // mode = "Written"
var subject = object.get("subject"); // subject = "Maths"
```

3. Retrieval of JSON Object using JQuery

- The JSON object can be read and retrieved using JQuery from below document.

```
Var data =
{
"score": 97,
"subject": "Maths",
"mode": "Written"
}
{
"score": 67,
"subject": "Maths",
"mode": "Oral"
}
```

- The JQuery for retrieving data is written as given below.

```
var score = data[0].score;    // score = 97
var mode = data[0].mode; // mode = "Written"
var subject = data[0].subject; // subject = "Maths"
var mode = data[1].mode; // mode = "Oral"
```

Review Questions

Q. 1 Explain any five schema keywords.

Q. 2 Compare JSON & XML with example.

Q. 3 Define Basic JSON syntax.

Q. 4 What are the different data types in JSON? Discuss about JSON object and ARRAY in details.

Q. 5 What is the rule for writing JSON ? Differentiate between JSON and XML.

□□□

7 NoSQL Database

Syllabus

NoSQL database concepts : NoSQL data modeling, Benefits of NoSQL, comparison between SQL and NoSQL database system. Replication and sharding, Distribution Models Consistency in distributed data, CAP theorem, Notion of ACID Vs BASE, handling Transactions, consistency and eventual consistency . Types of NoSQL databases : Key-value data store, Document database and Column Family Data store, Comparison of NoSQL databases w.r.t CAP theorem and ACID properties.

7.1 Introduction to NoSQL Database

1. History

- The term NoSQL was first used by Carlo Strozzi in the year 1998.
- He mentioned this name for his Open Source Database system in which there was no provision of SQL Query interface.
- In the early 2009, at conference held in USA, NoSQL was comes into picture and actually comes in practice.

2. Overview

- NoSQL is a not a RDBMS (Relational Database Management System).
- NoSQL is specially designed for large amount of data stored in distributed environment.
- The important feature of NoSQL is, it is not bounded by table schema restrictions like RDBMS. It gives options to store some data even if there is no such column is present in table.
- NoSQL generally avoids join operations.

3. Need

- In real time, data requirements are changed a lot. Data is easily available with Facebook, Google+, Twitter and others.
- The data that includes user information, social graphs, geographic location data and other user-generated content.
- To make use of such abundant resources and data, it is necessary to work with a technology which can operate such data.
- SQL databases are not ideally designed to operate such data.
- NoSQL databases specially designed for operating huge amount of data.

4. Advantages

1. Good resource scalability.
2. Lower operational cost.
3. Supports semi-structure data.
4. No static schema.
5. Supports distributed computing.
6. Faster data processing.
7. No complicated relationships.
8. Relatively simple data models.

5. Disadvantages

1. Not a defined standard.
2. Limited query capabilities.

6. Companies working with NoSQL

1. Google 2. Facebook
3. LinkedIn 4. McGraw-Hill Education

7.2 NoSQL Data modeling

- Data model will give you idea how your final system or software will look like after development is completed.
- This concept is exactly like real world modeling in which before constructing any project (Bridges, Buildings, Towers etc.) engineers create a model for it, this model gives you idea about how your project will look like after construction.
- A data model is an overview of a software system which describes how data can be represented and accessed from software system after its complete implementation.
- Data models define data elements and relationships among various data elements for a specific system.
- If we need to design a system which keeps track of student and classes information we can use model given below,

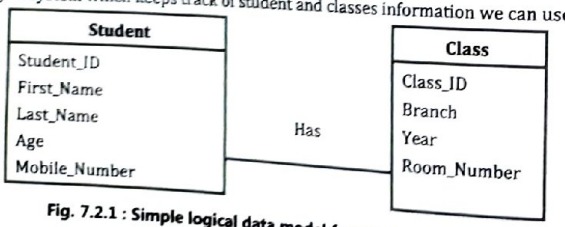


Fig. 7.2.1 : Simple logical data model for student and class

- **According to Hoberman (2009)** "A data model is a way finding tool for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment."
- Data model is a simple Abstraction of complex real-world data gathering environment.

7.2.1 Benefits of Data Modeling

- A data model is a set of concepts that can be used to **describe thestructure of data** in a database.
  In Fig. 7.2.1 (Simple Logical Data model) we have described structure of student data and class data.
- Data models are used to **support the development** of information systems by providing the definition and format of data to be involved in future system.
- Data model is acting like a guideline for development also gives idea about possible alternatives to achieve targeted solution.
- A data model can be sometimes referred to as a **data structure**, especially in the context of programming languages.
- In Fig. 7.2.1 (Simple Logical Data model) student and class are data structure of type class.

a. Reduced risk

- Data model prevents system from **future risk and failure** by defining structure of data in advance.
- As we got idea of final system in the beginning of development itself so if need to have any revision or improvement we can do it in system, as actual system is not yet developed.

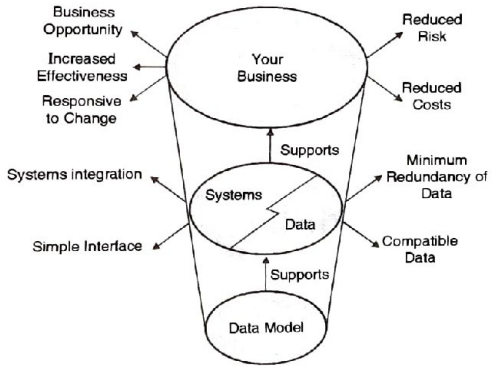


Fig. 7.2.2 : Important roles of Data Model in DBMS

b. Reduced cost

As we got an idea of final system at the beginning of development itself, so we can reduce cost of project by proper planning and cost estimations as actual system is not yet developed.

c. Minimizes redundancy and data compatibility

Data repetition and data type compatibility can be checked and removed with help of data model.

d. Improves effectiveness of system

We can improve Graphical User Interface (GUI) of system by making its model and get it approved by its future user (user of that system) so it will be simple for them to operate system and make entire system effective.

7.2.2 NoSQL of Data Modeling

- NoSQL data modelling often requires good understanding of data structures and algorithms than relational database modelling.

- Relational databases are not much suitable for hierarchical or graph-like data modelling and processing.
- Graph databases can be a good solution for this area, but actually most of NoSQL solutions are surprisingly are very good for such problems.
- There are multiple solutions are used for implementing NoSQL.

7.3 Types of NoSQL Databases

Types of NoSQL Databases

- Key-Value databases Eg.Riak,Redis,Memcached,BerkeleyDB, upscaledb,Amazon DynamoDB.
- Document databases Eg. MongoDB, CouchDB, Terrastore,OrientDB,RavenDB
- Column family stores Eg.Cassendra, HBase,HyperTable.
- Graph Databases Eg. Neo4j,InfiniteGraph, FlockDB.

7.3.1 Key-Value Data Store Databases

- This is very simple NoSQL database.
- It is specially designed for storing data as a schema free data.
- Such data is stored in a form of data along with indexed key.

Examples :

1. Cassandra
2. Azure Table Storage (ATS)
3. DyanmoDB

Fig. 7.3.1

Use Cses

- This type is generally used when you need quick performance for basic Create-Read-Update-Delete operations and data is not connected.

Example :

- Storing and retrieving session information for a Web pages.
- Storing user profiles and preferences
- Storing shopping cart data for ecommerce

Limitations

- It may not work well for complex queries attempting to connect multiple relations of data.
- If data contains lot of many-to-many relationships, a Key-Value store is likely to show poor performance.

7.3.2 Column Family Data Store Database

- Instead of storing data in relational tuples (table rows), it is stored in cells grouped in columns.
- It offers very high performance and a highly scalable architecture.

Fig. 7.3.2

Examples :

1. HBase 2. Big Table 3. Hyper Table

Use Cases

- Some common examples of Column-Family database include event logging and blogs like document databases, but the data would be stored in a different fashion.
- In logging, every application can write its own set of columns and have each row key formatted in such a way to promote easy lookup based on application and timestamp.
- Counters can be a unique use case. It is possible to design application that needs an easy way to count or increment as events occurs.

7.3.3 Document Database (Document Store)

Q. Give difference between document oriented database and traditional database. **MU – May 19, 10 Marks**

Q. Write short note on : Document Oriented Databse. **MU – May 19, 10 Marks**

- Document databases works on concept of key-value stores where "documents" contains a lot of complex data.
- Every document contains a unique key, used to retrieve the document.

- Key is used for storing, retrieving and managing document-oriented information also known as semi-structured data.

Fig. 7.3.3

Examples:

1. MongoDB 2. Couch DB

Use Cases

- The example of such system would be event logging system for an application or online blogging.
- In online blogging user acts like a document; each post a document; and each comment, like, or action would be a document.
- All documents would contain information about the type of data, username, post content, or timestamp of document creation.

Limitations

- It's challenging for document store to handle a transaction that on multiple documents.
- Document databases may not be good if data is required in aggregation.

7.3.4 Graph Database (Distributed Document Store)

Data is stored as a graph and their relationships are stored as a link between them whereas entity acts like a node.

Examples :

1. Neo4j 2. Polyglot

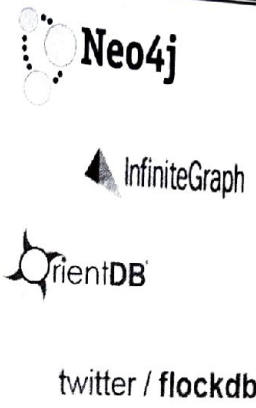


Fig. 7.3.4

Use Cases

- The very important and popular application would be social networking sites can benefit by quickly locating friends, friends of friends, likes, and so on.
- The Google Maps can help you to use graphs to easily model their data for finding close locations or building shortest routes for directions.
- Many recommendation systems makes effective use of this model.

Limitations

- Graph Databases may not be offering better choice over other NoSQL variations.
- If application needs to scale horizontally this may introduces poor performance.
- Not very efficient when it needs to update all nodes with a given parameter.

7.3.5 Comparison of NoSQL Databases w.r.t. CAP Theorem and ACID

Database model	Performance	Scalability	Flexibility
Key value store database	High	High	High
Column store database	High	High	Moderate
Document store database	High	Variable (High)	High
Graph database	Variable	Variable	High

7.3.6 Benefits of NoSQL

1. **Big Data Analytics**
   - Big data is one of main feature promotes growth and popularity of NoSQL.
   - NoSQL has good provision to handle such big data.
2. **Better Data Availability**
   - NoSQL database works with distributed environments.

- NoSQL database environments should provide good availability across multiple data servers.
- NoSQL databases supply high performance.

3. Location Independence

NoSQL data base can read and write database regardless of location of database operation.

7.4 Comparative between SQL and NoSQL Database systems (SQL vs NoSQL)

Q. Explain how NOSQL databases are different than relational databases? MU - Dec.17. 8 Marks

Q. Explain the difference SQLVsNoSQL. MU - May 18,Dec.18. 4 Marks

SQL databases are Relational Databases (RDBMS); whereas NoSQL database are non-relational database.

1. Data Storage

- SQL databases stores data in a table whereas NoSQL databases stores data as document based, key-value pairs, graph databases or wide-column stores.
- SQL data is stored in form of tables with some rows.
- NoSQL data is stored as collection of key-value pair or documents or graph based data with no standard schema definitions.

2. Database Schema

SQL databases have predefined schema which cannot be change very frequently, whereas NoSQL databases have dynamic schema which can be change any time for unstructured data.

3. Complex Queries

- SQL databases provides standard platform for running complex query.
- NoSQL does not provide any standard environment for running complex queries.
- NoSQL are not as powerful as SQL query language.

Table 7.4.1

Sr. No.	SQL	NoSQL
1.	Full form is Structured Query Language.	Full form is Not Only SQL or Non-relational database.
2.	SQL is a declarative query language.	This is Not a declarative query language.
3.	SQL databases works on ACID properties, Atomicity Consistency Isolation Durability	NoSQL database follows the Brewers CAP theorem, Consistency Availability Partition Tolerance
4.	Structured and organized data	Unstructured and unreplicable data
5.	Relational Database is table based.	Key-Value pair storage, Column Store, Document Store, Graph databases.
6.	Data and its relationships are stored in separate tables.	No pre-defined schema.
7.	Tight consistency.	Eventual consistency rather than ACID property.

Sr. No.	SQL	NoSQL
8.	**Examples :** MySQL Oracle MS SQL PostgreSQL SQLite DB2	**Examples :** MongoDB Big Table Neo4j Couch DB Cassandra HBase

7.4.1 Business Drivers of NoSQL

1. The Growth of Big Data

- Big Data is one of the main driving factor of NoSQL for business.
- The huge array of data collection act as driving force for data growth.

2. Continuous Availability of Data

- The competition age demands less downtime for better company reputation.
- Hardware failures are possible but NoSQL database environments are built with a distributed architecture so there are no single points of failure.
- If one or more database servers goes down, the other nodes in the system are able to continue with operations without any loss of data.

So, NoSQL database environments are able to provide continuous availability.

3. Location Independence

- It is ability to read and write to a database regardless of where that I/O operation is done.
- The master/slave architectures and database sharding can sometimes meet the need for location independent read operations.

4. Modern Transactional Capabilities

The transactions concept is changing and ACID transactions are no longer a requirement in database systems.

5. Flexible Data Models

- NoSQL has more flexible data model as compared to others.
- A NoSQL data model is schema-less data model not like RDBMS.

6. Better Architecture

- The NoSQL has more business oriented architecture for a particular application.
- So, Organizations adopt a NoSQL platform that allows them to keep their very high volume data.

7. Analytics and Business Intelligence

- A key driver of implementing a NoSQL database environment is the ability to mining data to derive insights that offers a competitive advantage.
- Extracting meaningful business information from very high volumes of data is a very difficult task for relational database systems.
- Modern NoSQL database systems deliver integrated data analytics and better understanding of complex data sets which facilitate flexible decision-making.

7.5 Distribution Models: Master-Slave versus Peer-to-Peer

Master Slave Vs. Peer to Peer Model

- The data bases like HBase follow the master slave model In Master slave model one node out of all participating nodes will acts as the master i.e. the master node will responsible for governing all the decisions related to the job assignment, task assignment, data storage, data retrieval, data manipulation etc. The Master will delegate the jobs and tasks to the other working nodes in a given cluster. The main advantage is the working nodes cannot manipulate the things without permission of the master node.
- Fig. 7.5.1 shows Master slave model.

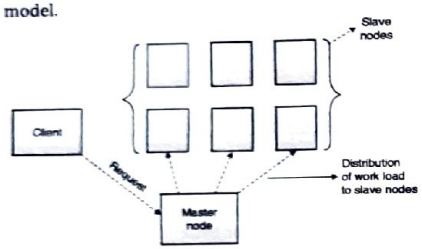


Fig. 7.5.1 : Master slave model

- The data bases like cassendra will follow the Peer- to-peer model. In peer to peer model all participating nodes has same rights. It means every node can perform all operations requested by the database user.
- These operations include Create, Read, Update, Delete as well as operations related to configuration of such databases.
- As all nodes has the same priority , so the requests from data base users will be received by any of the nodes irrespective of work load distribution.
- The peer to peer model will exhibit a data replication mechanism where data packets will be replicated for certain number of times to avoid the data loss if some part of the data base will get crashed.
- Fig. 7.5.2 shows Peer to Peer model.

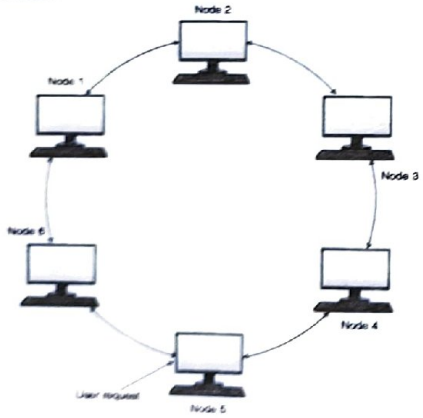


Fig. 7.5.2 : Peer to peer model

7.6 CAP Theorem (Brewer's Theorem)

Q. State and Explain CAP Theorem.

Q. Explain the CAP theorem referred during the development of any distributed application.

CAP theorem states three basic requirements of NoSQL databases to design a distributed architecture.

a) **Consistency**

Database must remain consistent state like before, even after the execution of an operation.

b) **Availability**

It indicates that NoSQL system is always available without any downtime.

c) **Partition Tolerance**

This means that the system continues to function even the communication failure happens between servers i.e. if one server fails, other server will take over.

Note : It is very difficult to fulfill all the above requirements.

There are many combinations of NoSQL rules :

1. **CA**
   - It is a single site cluster.
   - All nodes are always in contact.
   - Partitioning system can block the system.
2. **CP**

   Some data may not be accessible always still it may be consistent or accurate.
3. **AP**
   - System is available under partitioning.
   - Some part of the data may be inconsistent.

7.7 Database Replication and Sharding

- Database Replication means creating multiple copies of data which is also called as database shards.
- Database Sharding is related to horizontal partitioning in databases for purpose of improving availability of data.
- The portioning means separating rows of one table into multiple different tables.
- In such case each partition has the same schema and columns, but different set of rows.
- The data held in each partition is unique and independent of the data in other partition.
- The Vertical partitioning will separate all columns in table and insert into new table.
- The data held within one vertical partition is independent from the data in all the other vertical partitions.

Original **Table**

Customer ID	First Name	Last Name	Favorite Color
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAGCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1		
Customer ID	First Name	Last Name
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAGCAN
4	JIM	PEPPER

VP2	
Customer ID	Favorite Color
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1			
Customer ID	First Name	Last Name	Favorite Color
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2			
Customer ID	Favorite Color	Last Name	Favorite Color
3	SELDA	BAGCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Fig. 7.7.1

1. **Sharding Process**
   - Sharding process involves breaking up data on one server to multiple smaller chunks or parts, which is also called as *logical shards*.
   - The logical shards are distributed across separate database servers or nodes.
   - The physical server holding multiple logical shards together called as physical shards.
   - All shards together represent an entire logical dataset.
2. **Sharding Nothing Architecture**
   - This architecture doesn't share any data or computing resources.
   - The certain tables can be replicate into each shard to serve as reference tables.
   - The table containing the necessary data for computation of value must be kept in all shard, it would help to ensure that all of the data required for computation is held in every shard.
   - It is possible to implement sharding at the application level, So application includes code that defines which shard to transmit the data.
   - Few DBMS are having sharding capabilities by default, allowing you to implement sharding directly at the database level.
3. **Advantages**
   - **Scalability**: The advantage of sharding a database is that it can help for *horizontal scaling*, also known as *scaling out*. It means adding more machines to an existing architecture in order to distribute the load and allow for more traffic for faster processing.
   - **To Improve Speed and response time**: Sharded database architecture will increase the speed of query to reduce the response times. When you submit a query on a database that hasn't been sharded, it may have to search every row in the table you're querying before it can find the result set you're looking for. For an application with a large, monolithic database, queries can become prohibitively slow. By sharding one table into multiple, though, queries have to go over fewer rows and their result sets are returned much more quickly.

   - **Smaller Databases (shards) are Easier to Manage:** The databases must be managed regularly with backups, cleaning and other common tasks.
   - **Improve Reliability**: Sharding will make an application more reliable by reducing the impact of outages.
   - **Economic:** Many sharding implementations uses low-cost open source databases which reduces overall cost of implementation.
4. **Disadvantages**
   - **Increases Complexity:** It introduces the additional complexity of implementing a sharding database architecture. Even sometimes data loss is als possible due to incorrect implementation.
   - **Slowdown:** Database may face problem of slowdown due to more shards of same data.
   - Not supported in all database architectures.

7.8 Database Consistency Models

7.8.1 ACID vs Base Notion

- When database shards comes to NoSQL databases, data consistency models can sometimes be strikingly different than those used by relational databases (as well as quite different from other NoSQL stores).
- The two most common consistency models are known by the acronyms ACID and BASE.
- While they're often pitted against each other in a battle for ultimate victory (please someone make a video of that), both consistency models come with advantages – and disadvantages – and neither is always a perfect fit.
- Let's take a closer look at the trade-offs of both database consistency models.
- In this *Graph Databases for Beginners* blog series, I'll take you through the basics of graph technology assuming you have little (or no) background in the space. In past weeks, we've tackled why graph technology is the future, why connected data matters, the basics (and pitfalls) of data modeling, why a database query language matters, the differences between imperative and declarative query languages, predictive modeling using graph theory, the basics of graph search algorithms and why we need NoSQL databases.This week, we'll take a closer look at the key differences between ACID and BASE database consistency models and what their trade-offs mean for your data transactions.

7.8.2 ACID Consistency Model

- ACID transactions are very common in relational databases.
- The ACID properties guarantee is that it provides a safe environment to operate on your sensitive data.
- ACID properties make sure once a transaction is complete data submitted is consistent and stable on distributed or single disk.
- When it comes to NoSQL technologies, most graph databases uses an ACID consistency model to ensure data is safe and consistently stored.

1. **Atomicity**
   - Transaction must be treated as a single unit of operation.
   - That means when a sequence of operations is performed in a single transaction, they are treated as a single large operation.
   - In distributed environment there are many local transaction performing changes on same data items makes a single global transaction which should be executed completely or else all sites operation should be roll back.

Examples,

(a) Withdrawing money from your account.

(b) Making an airline reservation.

- Execution of a distributed transaction should be either complete or nothing should be executed at all.
- No partial transaction executions are allowed. (No half done transactions)

2. Consistency

- If there are multiple sites performing changes on distributed data but only valid and correct changes should be reflected on distributed database.
- Consistent state is a state in which only correct data will be written to the database.
- If due to some reason, a transaction violates the database's data consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules.
- On the other hand, if a transaction is executed successfully than it will take the database from one consistent state to another consistent state.
- DBMS should handle an inconsistency and also ensure that the database is clean at the end of each transaction.
- Consistency guarantees that a transaction will never leave your database in a half finished (inconsistent) state.
- Consistency means if one part of the transaction fails, all of the pending changes made by that transaction are rolled back, leaving the database as it was before you initiated the transaction.

3. Isolation

- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- Isolation property keeps multiple transactions separated from each other until they are completed.
- Operations occurring in a transaction (example, insert or update statements) are invisible to other transactions until the transaction commits (on transaction success) or rolls back (on transaction fails).
- For example, when a transaction changes a bank account balance other transactions cannot see the new balance until the transaction commits.
- Even though many transactions may execute concurrently in the system. System must guarantees that, for every transactions (T_i)other all transactions has finished before transactions (T_i) started, or other transactions are started execution after transactions (T_i) finished.
- That means each transaction is unaware of other transactions executing in the system simultaneously.
- A way to avoid the problem of concurrently executing transactions is to execute one transaction at a time.

4. Durability

- The results of a transaction that has been successfully committed to the database will remain unchanged even after database fails.
- Changes made during a transaction are permanent once the transaction commits.
- The distributed database handles durability by storing uncommitted transactions in a global transaction log. So that, a partially completed transaction won't be written to the database in the event of an abnormal termination.

7.8.3 BASE Consistency Model

Q. State and Explain: BASE properties.

Q. BASE Transactions ensures the properties like Basically Available, Soft State, Eventual Consistency. What is soft state of any system, how it is depend on Eventual consistency property? MU - May 18.6 Marks MU - Dec.17 May 18,Dec.18.6 Marks

1. BASE Introduction

- Relational databases have some rules to decide behavior of database transactions.
- ACID model maintains the atomicity, consistency, isolation and durability of database transactions.
- NoSQL turns the ACID model to the BASE model.

2. Data storage

- NoSQL databases use the concept of a key / value store.
- There are no schema restrictions for NoSQL database.
- It simply stores values for each key and distributes them across the database, it offers efficient retrieval.

3. BASE Guidelines

Basic availability

The database isaccessible to all users most of the time.

Soft state

- The Data Stores No need to have be write-consistent
- Different replicas not be mutually consistent all the time.

Eventual consistency

- Data Stores and exhibit consistency at some later point also called as Lazy Read or Write.

4. Redundancy and Scalability

- To add redundancy to a database, we can add duplicate nodes and configure replication.
- Scalability is simply a matter of adding additional nodes. There can be hash function designed to allocate data to server.

5. Redundancy and Scalability

- BASE properties are much soft limitation than ACID Properties.
- There is no direct mapping between the two consistency models.
- A BASE data store values availability, but it cannot guarantee consistency of replicated data at write time.

Review Question

Q. 1 Compare SQL and NoSQL databases.

Q. 2 Explain the concept of NoSQL Database and state its advantages over RDBMS.

Q. 3 Explain concept of Sharding.

Q. 4 What is data model?

□□□

8 NoSQL using MongoDB

Syllabus

NoSQL using MongoDB : Introduction to MongoDB Shell, Running the MongoDB shell, MongoDB client, Basic operations with MongoDB shell,

Basic Data Types, Arrays, Embedded Documents

Querying MongoDB using find() functions, advanced queries using logical operators and sorting, simple aggregate functions, saving and updating document.

MongoDB Distributed environment: Concepts of replication and horizonal scaling through sharding in MongoDB

8.1 Introduction to MongoDB Shell

"MongoDB is one of the leading NoSQL document store databases. It enables organizations to handle and gain meaningful insights from Big Data."

- A database is defined in large part by its data model. MongoDB's data model is document-oriented. MongoDB stores documents in a format called Binary JSON or BSON. BSON has a similar structure but is intended for storing many documents.
- When you query MongoDB and get results back, these will be translated into an easy-to-read data structure. The MongoDB shell uses JavaScript and gets documents in JSON.
- We have collection on MongoDB and tables in RDBMS. Collection consists of documents. The collection in MongoDB stores data on disk to retrieve we have to specify target collection.
- A normalized data set, among other things, ensures that each unit of data is represented in one place only. Whereas MongoDB avoids this entire hurdle i.e. it does not require normalization to perform because of its structure.
- Most databases give each document or row a primary key, a unique identifier for that datum. The primary key is generally indexed automatically so that each datum can be efficiently accessed using its unique key, and MongoDB is no different.
- But not every database allows you to also index the data inside that row or document. These are called secondary indexes. Many NoSQL databases, such as HBase, are considered key value stores because they don't allow any secondary indexes.
- This is a significant feature in MongoDB; by permitting multiple secondary indexes MongoDB allows users to optimize for a wide variety of queries.
- With MongoDB, you can create up to 64 indexes per collection.
- Replica set is used in MongoDB for database replication. Replica sets divide data across multiple machines (two or more) for redundancy and failure due to any outage of network or server failure. Also useful for scaling the data read.

- The easiest way to scale most databases is to upgrade the hardware. If your application is running on a single node, it's usually possible to add some combination of faster disks, more memory, and a beefier CPU to ease any database bottlenecks.
- The technique of augmenting a single node's hardware for scale is known as vertical scaling, or scaling up. Vertical scaling has the advantages of being simple, reliable, and cost-effective up to a certain point, but eventually you reach a point where it's no longer feasible to move to a better machine.
- It then makes sense to consider scaling horizontally, or scaling out. Instead of beefing up a single node, scaling horizontally means distributing the database across multiple machines.
- A horizontally scaled architecture can run on many smaller, less expensive machines, often reducing your hosting costs. What's more, the distribution of data across machines mitigates the consequences of failure.
- Machines will unavoidably fail from time to time. If you've scaled vertically and the machine fails, then you need to deal with the failure of a machine on which most of your system depends.
- This may not be an issue if a copy of the data exists on a replicated slave, but it's still the case that only a single server need fail to bring down the entire system. Contrast that with failure inside a horizontally scaled architecture. This may be less catastrophic because a single machine represents a much smaller percentage of the system as a whole. MongoDB was designed to make horizontal scaling manageable.
- It does so via a range-based partitioning mechanism, known as sharding, which automatically manages the distribution of data across nodes.
- There's also a hash- and tag-based sharding mechanism, but it's just another form of the range-based sharding mechanism.
- The sharding system able to handle addional nodes of shard and helps in automatic failure of system. Each shard is made up of replica set, it has atleast two nodes which ensures automatic recovery.
- A MongoDB deployment can have many databases. Each database is a set of collections. Collections are similar to the concept of tables in SQL; however, they are schemaless. Each collection can have multiple documents. Think of a document as a row in SQL. Fig. 8.1.1 depicts the MongoDB database model.

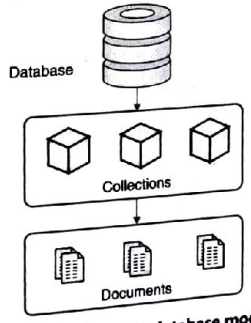


Fig. 8.1.1 : MongoDB database model

8.1.1 JSON and BSON

- MongoDB is a document-based database. It uses Binary JSON for storing its data. JSON stands for JavaScript ObjectNotation. It's a standard used for data interchange in today's modern Web (along with XML).

- The format is human and machine readable. It is not only a great way to exchange data but also a nice way to store data. All the basic data types (such as strings, numbers, Boolean values, and arrays) are supported by JSON.
- The following code shows what a JSON document looks like:

```
{
"_id" : 1,
"name" : { "first" : "vaibhav", "last" : "vasani" },
"publications" : [
{
"title" : "ADBMS",
"year" : 2021,
"publisher" : "Tech"
},
{ "title" : "Deep Learning",
"year" : 2021,
"publisher" : "Tech"
}
]
}
```

- JSON lets you keep all the related pieces of information together in one place, which provides excellent performance. It also enables the updating of a document to be independent. It is schemaless.

8.1.1(A) Binary JSON (BSON)

- MongoDB stores the JSON document in a binary-encoded format. This is termed as BSON. The BSON data model is an extended form of the JSON data model.
- MongoDB's implementation of a BSON document is fast, highly traversable, and lightweight.
- It supports embedding of arrays and objects within other arrays, and also enables MongoDB to reach inside the objects to build indexes and match objects against queried expressions, both on top-level and nested BSON key

8.1.1(B) The Identifier (_id)

- You have seen that MongoDB stores data in documents. Documents are made up of key-value pairs. Although a document can be compared to a row in RDBMS, unlike a row, documents have flexible schema.
- A key, which is nothing but a label, can be roughly compared to the column name in RDBMS. A key is used for querying data from the documents. Hence, like a RDBMS primary key (used to uniquely identify each row), you need to have a key that uniquely identifies each document within a collection.
- This is referred to as _id in MongoDB. If you have not explicitly specified any value for a key, a unique value will be automatically generated and assigned to it by MongoDB. This key value is immutable and can be of any data type except arrays.

8.1.1(C) Capped Collection

- MongoDB has a concept of capping the collection. This means it stores the documents in the collection in the inserted order.

- As the collection reaches its limit, the documents will be removed from the collection in FIFO (first in, first out) order.
- This means that the least recently inserted documents will be removed first.
- This is good for use cases where the order of insertion is required to be maintained automatically, and deletion of records after a fixed size is required.
- One such use cases is log files that get automatically truncated after a certain size

8.1.1(D) Polymorphic Schemas

- A polymorphic schema is a schema where a collection has documents of different types or schemas. A good example of this schema is a collection named Users.
- Some user documents might have an extra fax number or email address, while others might have only phone numbers, yet all these documents coexist within the same Users collection.
- This schema is generally referred to as a polymorphic schema.
- The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

Following points needs to be consider while designing Schema in MongoDB

- Design the schema according to user requirements.
- Combine the objects into one document if it's going to be using them together. Otherwise separate them (but make sure there should not be need of joins).
- Duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in the schema.

8.2 Running the MongoDB shell

8.2.1 Installing MongoDB on Ubuntu

1. Open Terminal on Ubuntu
2. Type the following command on terminal to import the public.GPG key for MongoDB:
   sudo apt-key adv –keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
3. Type the following command on terminal to create the /etc/apt/sources.list.d/ mongodb-org-3.0.list file:
   echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/ mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/ mongodb-org-3.0.list

4. Reload the repository by typing following command on terminal
   sudo apt-get update
5. To install the software type following script on terminal
   sudo apt-get install -y mongodb-org
6. Next, You can start Client and Server of MongoDB
   Start MongoDB
   sudo service mongodb start
   Stop MongoDB
   sudo service mongodb stop
   Restart MongoDB
   sudo service mongodb restart
7. To use MongoDB run the following command.
   mongo

8.2.2 Installing MongoDB on Windows

- Installing MongoDB on Windows is a simple matter of downloading the msi file for the selected build of Windows and running the installer.
  https://www.mongodb.com/download-cente
- Double click on the installer and wizard will guide you through installation of MongoDB. By default, it will be installed in the folder
  C:\Program Files\.
- MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is c:\data\db. So you need to create this folder using the Command Prompt. Execute the following command sequence.
  C:\>md data
  C:\md data\db
- Then you need to specify set the dbpath to the created directory in mongod.exe. For the same, issue the following commands.
- In the command prompt, navigate to the bin directory current in the MongoDB installation folder. Suppose my installation folder is
  C:\Program Files\MongoDB
  C:\Users\XYZ>d:cd
  C:\Program Files\MongoDB\Server\4.2\bin
  C:\Program Files\MongoDB\Server\4.2\bin>mongod.exe--dbpath "C:\data"
- This will show waiting for connections message on the console output, which indicates that the mongod.exe process is running successfully.
- Now to run the MongoDB, you need to open another command prompt and issue the following command.

```
C:\Program Files\MongoDB\Server\4.2\bin>mongo.exe
MongoDB shell version v4.2.1
```



```
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4260beda-f662-4cbe-9bc7-5c1f2242663c") }
MongoDB server version: 4.2.1
```

8.2.3 How to Verify the Installation is successful

The executable is present inside the subdirectory bin.

- Mongod is the core database server
- Mongo is the database shell
- Mongos is the auto-sharding process
- Mongoexport is the export utility
- Mongoimport is the import utility

All above is present inside the directory of bin.

Database Server and Client
Mongod as Server
Mongo as Client

Fig. 8.2.1

- The mongo application is used for launching the mongo shell, which access to the database contents and we can fire queries or execute aggregation on the data in MongoDB.
- The mongod application is used to start the database service.

8.3 MongoDB client

- The mongo shell is the standard distribution of MongoDB. It provides interface for MongoDB, which enables to interact through a JavaScript environment.
- The exact location of the executable is as follows
  C:\practicalmongodb\bin\ (Windows environment)
  1. Open Command Prompt (Run as administrator)
  2. Type monngo.exe to run
  3. Press enter Key
  4. mongo shell Will start

Sample Document

- Following example shows the document structure of a blog site

```
{
  _id: ObjectId(9va11ibh1987av)
title: 'MongoDB Course',
description: 'Course by Prof. Vaibhav Vasani',
by: 'vaibhavs blog',
```

```
url: 'http://www.krishna.com',
tags: ['mongodb', 'database course', 'NoSQL','Onetimeopportunity'],
likes: 1008,
comments: [
    {
user:'keshav',
message: 'best course',
dateCreated: new Date(2021,1,20,2,15),
like: 0
    },
    {
user:'radha',
message: 'enjoyed the course',
dateCreated: new Date(2021,1,25,7,45),
like: 51
    }
  ]
}
```

Note: _id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

8.4 Basic operations with MongoDB shell

8.4.1 MongoDB Help

To get a list of commands, type db.help() in MongoDB client. This will give you a list of commands

8.4.2 MongoDB Statistics

To get stats about MongoDB server, type the command db.stats() in MongoDB client. This will show the database name, number of collection and documents in the database.

8.4.3 The use Command

- The use command is used to create database.
- The use command will create a new database if it doesn't exist, otherwise it will return the existing database.

  Syntax:

  use Database_Name

  Example :

```
>use myVPVblogdb
switched to myVPVblogdb
```

- To check current database, use the command "db"

```
>db
myVPVblogdb
```


- List the databases present, use the command "show dbs".

```
>show dbs
local   0.78335GB
test    0.23401GB
```

- Our created database (myVPVblogdb) is not present in list. To display our database in the list, we need to insert at least one document into it then it will list our database.

```
>db.series.insert({"name":"radhakrishna"})
>show dbs
Local 0.78125GB
myVPVblogdb 0.23012GB
test 0.23012GB
```

- Test is default database. If we didn't create any database, then collections will be stored in default database "Test".

8.4.4 The dropDatabase() Method

- Command is used to drop a existing database.

  Syntax :

  db.dropDatabase()

- It will delete the selected database. If we do not select any database, then it will delete default 'test' database from the system.

Example

```
>show dbs
Local  0.78135GB
Myvpvblogdb 0.25012GB
test  0.230133GB
>
>use VPVmyblogdb
switched to db myVPVblogdb
>db.dropDatabase()
>{"dropped":"myvpvblogdb","ok":1}
>
>show dbs
Local  0.78135GB
test  0.230133GB
>
```

8.4.5 The createCollection() Method

- db.createCollection(name, options) command is used to create collection in MongoDB.

  Syntax:

  db.createCollection(name, options)

- Name is name of collection we want to create. Its type will be string.
- Options are a document and are used to specify configuration of collection. It is optional field; we can specify options about memory size and indexing.
- Options are as follows :

1. **Capped**

   its data type will be Boolean. If true, enables a capped collection. It is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If we specify true, the we need to specify size parameter also.

2. **AutoIndexId**

   It data type will be Boolean. If true, automatically create index on _id field.s Default value is false.

3. **Size number**

   Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.

4. **Max number**

   Specifies the maximum number of documents allowed in the capped collection.

   If you're inserting the document, MongoDB will first checks size field of capped collection, then it checks max field.

```
>use test
switched to db test
>db.createCollection("myVPVollection")
{"ok":1}
>
>show collections
myVPVcollection
system.indexes
```

Example

```
>db.createCollection("myvpvcol",{ capped :true,autoIndexID:true, size :7142870, max :10000}){
"ok":0,
"msg":"BSON field 'create.autoIndexID' is not known field.",
"code":421301,
"codeName":"Location421301"
}
>
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.tutorialspoint.insert({"name":"RadhaKrishna"}),
WriteResult({"nInserted":1})
>show collections
```



```
mycol
myStudcollection
system.indexes
technical_publication
>
```

8.4.6 The drop() Method

The db.collection.drop() is used to drop a collection from the MongoDB.

Syntax

db.COLLECTION_NAME.drop()

Example

```
>show collections
mycol
myVPVcollection
system.indexes
technical_publication
>
>db.myVPVcollection.drop()
true
>
>show collections
mycol
system.indexes
technical_publication
>
```

8.5 Basic Data Types

Data Type	Meaning
Symbol	This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
Date	This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it
Object ID	This datatype is used to store the document's ID.
Binary data	This datatype is used to store binary data.
Code	This datatype is used to store JavaScript code into the document.
Regular expression	This datatype is used to store regular expression.

Data Type	Meaning
String	This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
Integer	This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
Boolean	This type is used to store a boolean (true/ false) value.
Double	This type is used to store floating point values.
Min/ Max keys	This type is used to compare a value against the lowest and highest BSON elements.
Arrays	This type is used to store arrays or list or multiple values into one key.
Timestamp, ctimestamp	This can be handy for recording when a document has been modified or added.
Object	This datatype is used for embedded documents.
Null	This type is used to store a Null value.

8.6 Arrays

- MongoDB Arrays are a flexible document structure. It is a simple list of values.
- We can have an array of string, integer, embedded documents, Jason and BSON data types.

  Syntax

  We can define arrays are as follows.

  {< array field >: {< operator1> : <value1>, < operator2> : <value2>, < operator3> : <value3>, ... }}

  Parameters to MongoDB Array

  1. Array field : The field name of the collection on which we create or define array values.
  2. Operator : The operator name is specific to the value name in the array.
  3. Value : The actual values of an array.

Example

- In the below example, we have to define an array of stud_skills after defining we have inserted an array of documents in stud_count collection:

```
db.stud_count.insertOne({"stud_name":"Keshav", "stud_skills": [["PostgreSQL", "MongoDB", "MySQL", "Perl","ORACLE"]]});
```

- In the above example, we have to define an array of stud_skills of employees. At the time of defining an array, we have inserted documents and created stud_count collection.
- In the above example, we have created an array of stud_skills in stud_count collection. At the time of defining the array, we have created a collection name as stud_array.
- We can define an array at the time of insertion. We can define multiple array fields in a single collection.
- Below example shows that define a multiple array field in the single collection are as follows:

```
db.stud_countmultiple.insertOne({"stud_name":"Arjun", "stud_skills":[["PostgreSQL", "MongoDB", "MySQL", "Perl",
"ORACLE"]], "stud_address":[["Mathura", "Mumbai", "Gokul"]]});
```

- In the above example, we have defined multiple array fields in one collection. We have to define stud_skills and stud_address array in a single collection.

8.6.1 Various Array Operators in MongoDB

1. $all
2. $elemMatch
3. $size
4. $
5. $pull
6. $push
7. $pop

$all

It is used to display all the value from the array field.

Ex.

```
db.stud_finaltest.find ({results: {$all: [75]}})
```

$elemMatch

It is used to match the document which contains the array field and contains only one filed to match our given criteria :

Ex.

```
db.stud_finaltest.find({results: {$elemMatch: {$gte: 75, $lt: 80}}})
```

$size

- It will match any array with the number of elements specified by the argument.

  Ex.

  ```
  db.stud_finaltest.find ({"results": {$size: 2}});
  ```

$

- It is used to identify array element and update the same into the collection:

  ```
  db.stud_test.updateOne ({stud_id: 1, results: 75}, {$set: {"results.$" : 80 }})
  ```

$pop

- $pop array operator is used to remove the first and last element from an array.

  Ex.

  ```
  db.stud_finaltest.updateOne( { stud_id: 100 }, { $pop: { results: -1 } } )
  ```

$pull

- A pull array operator is used to remove elements from an existing array.

  Ex.

  ```
  db.stud_finaltest.update( { stud_id: 101 }, { $pull: { results: { $gte: 95 } } } )
  ```

$push

- A push array operator is used to append the value into the existing collection.

  ```
  db.stud_finaltest.update ({stud_id: 1}, { $push: { results: 101 }})
  ```

8.7 Embedded Documents

- Embedded document also known as nested documents. Embedded document means documents which contain a document inside another document and another document contains another sub-document, and so on, then such types of documents are known as embedded/nested documents.
- Such nesting is possible up to 100 levels and the document size should not exceed 16 MB.

8.7.1 Creating Embedded Documents

- In MongoDB, you can easily embed a document inside another document. As we know that in the mongo shell, documents are represented using curly braces ({}) and inside these curly braces we have field-value pairs.
- Now inside these fields, we can embed another document using curly braces {} and this document may contain field-value pairs or another sub-document.

Synatx:

```
{
—
field: {field1: value1, field2: value2}
—
}
```

```
db.Courses.insertOne(
{
      name: {first: "Manan", middle: "Vaibhav", last: "Vasani"},
      branch: "CSE",
      courseName: "Deep Learning",
      courseType: "Free"
})
{
"acknowledged" : true,
"insertedId" : ObjectId("5e380cs33d792e6dfdda3fc48ddss65")
}
```

8.8 MongoDB CRUD Operations

CRUD operations create, read, update, and delete documents.

8.8.1 Create Operations

- Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.MongoDB provides the following methods to insert documents into a collection :

  db.collection.insertOne()

  db.collection.insertMany()

```
>db.createCollection("krishnapost")
>db.post.insert([
      {
          title: "AdvaitGyan",
```



```
          description: "AdvaitGyan by Krishna",
          by: "Krishna",
          url: "http://www.krishna.com",
          tags: ["Advait", "Knowledge", "krishna"],
          likes: 100
      },
      {
      title: "life management",
      description: "life management from Bhagwadgita",
      by: "Vaibhav Vasani",
      url: "http://www.vaibhav.vasani.com",
      tags: ["life", "management", "Bhagwadgita],
      likes: 20,
      comments: [
          {
              user:"Manan",
              message: "Awesome",
              dateCreated: new Date(2017,09,12,1,13),
              like: 11
          }
      ]
}
])
BulkWriteResult({
      "writeErrors" : [ ],
      "writeConcernErrors" : [ ],
      "nInserted" : 2,
      "nUpserted" : 0,
      "nMatched" : 0,
      "nModified" : 0,
      "nRemoved" : 0,
      "upserted" : [ ]
})
>
```

8.8.2 Insert Operation

8.8.2(A) 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("facultyDetails")
{"ok":1}
>db.empDetails.insertOne(
    {
        First_Name:"Vaibhav",
        Last_Name:"Vasani",
        Date_Of_Birth:"1987-11-08",
        e_mail:"vaibhav.vasani@gmail.com",
    })
{
    "acknowledged":true,
    "insertedId":ObjectId("5ed62c4050fb13ffc3963baa")
}
>
```

8.8.2(B) 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 facultyDetails collection using the insertMany() method.

```
>db.facultyDetails.insertMany(
    [
        {
            First_Name:"Manan",
            Last_Name:"Vasani",
            Date_Of_Birth:"1995-02-16",
            e_mail:"mananvasani@gmail.com",
            phone:"9000012345"
        },
        {
            First_Name:"Laksh",
```



```
            Last_Name:"Vasani",
            Date_Of_Birth:"1997-02-16",
            e_mail:"laksh.vasani@gmail.com",
            phone:"9000054321"
        },
        {
            First_Name:"Aryan",
            Last_Name:"Vasani",
            Date_Of_Birth:"1998-02-16",
            e_mail:"aryanvasani@gmail.com",
            phone:"9000064321"
        }
    ]
)
{
    "acknowledged":true,
    "insertedIds":[
        ObjectId("5dd631f270fb13eec3963bee"),
        ObjectId("5dd631f270fb13eec3963bef"),
        ObjectId("5dd631f270fb13eec3963beg")
    ]
}
>
```

8.9 Querying MongoDBusing find () functions

8.9.1 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 -

```
>useVPVDB
switched to dbVPVDB
>db.createCollection("myVPVcol")
{"ok":1}
>
```

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

```
>db.myVPVcol.insert([
    {
        title:"Naruto",
        description:"Japanese Manga ",
        by:"Masashi Kishimoto",
        url:"http://www.anime.com",
        tags:["naruto","anime","manga"],
        likes:100
    },
    {
        title:"Harry Potter and half-blood prince",
        description:"Harry potter book series",
        by:"J K Rowling",
        url:"http://www.jkrowling.com",
        tags:["Harrypotter","HPbook","book"],
        likes:20,
        comments:[
            {
                user:"Kalindi ",
                message:"My favourite book",
                dateCreated:newDate(2020,01,11,2,35),
                like:10
            }
        ]
    }
])
```

Following method retrieves all the documents in the collection -

```
>db.myVPVcol.find()
{"_id":ObjectId("5de4e2cc0821d3b44607534c"),"title":" Japanese Manga ","description":" Japanese Manga ","by":" Masashi Kishimoto ","url":" http://www.anime.com","tags":[" naruto","anime","manga"],"likes":100}
{"_id":ObjectId("5dd4e2cc0821d3b44607534d"),"title":" Harry Potter and half-blood prince ","description":" Harry potter book series ","by":" J K Rowling ","url":" http://www.jkrowling.com ","tags":[" Harrypotter","HPbook","book
"],"likes":20,"comments":[{"user":" Kalindi","message":" My favourite book ","dateCreated":ISODate("2020-01-11T02:35:00Z"),"like":10}]}
>
```


8.9.2 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 myVPVcol and arranges them in an easy-to-read format.

```
>db.myVPVcol.find().pretty()
{
    "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
    "title":"Naruto",
    "description":"Japanese Manga",
    "by":"Masashi Kishimoto",
    "url":"http://www.anime.com",
    "tags":[
        "naruto",
        "anime",
        "manga"
    ],
    "likes":100
}
{
    "_id":ObjectId("5dd4e2cc0821d3b44607534d"),
    "title":"Harry Potter and half-blood prince",
    "description":"Harry potter book series",
    "by":"J K Rowling",
    "url":"http://www.jkrowling.com",
    "tags":[
        "Harrypotter",
        "HPbook",
        "book"
    ],
    "likes":20,
    "comments":[
        {
            "user":"Kalindi",
```

```
            "message":"My favourite book",
            "dateCreated":ISODate("2020-01-11T02:35:00Z "),
            "like":10
        }
    ]
}
```

8.9.3 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.myVPVcol.find().pretty()
{
    "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
    "title":"Naruto",
    "description":"Japanese Manga",
    "by":"Masashi Kishimoto",
    "url":"http://www.anime.com",
    "tags":[
        "naruto",
        "anime",
        "manga"
    ],
    "likes":100
}
```

8.9.4 MongoDB and RDBMS where clause Equivalents

Operation	Syntax	RDBMS Equivalent	Example
Values in an array	{<key>:{$in:[<value1>, <value2>,.....<valueN>]}}	Where name matches any of the value in :["krishna", "Madhav", "keshav"]	db.mycol.find({"name":{$in:["krishna", "Madhav", "keshav"]}}).pretty()
Values not in an array	{<key>:{$nin:<value>}}	Where name values is not in the array :["Ramesh", "Suresh"] or, doesn't exist at all	db.mycol.find({"name":{$nin:["Ramesh", "Suresh"]}}).pretty()

Operation	Syntax	RDBMS Equivalent	Example
Equality	{<key>:{$eg;<value>}}	where by = 'techknowledgebooks'	db.mycol.find({"by":"techknowledgebooks"}).pretty()
Less Than	{<key>:{$lt:<value>}}	where likes < 66	db.mycol.find({"likes":{$lt:66}}).pretty()
Less Than Equals	{<key>:{$lte:<value>}}	where likes <= 55	db.mycol.find({"likes":{$lte:55}}).pretty()
Greater Than	{<key>:{$gt:<value>}}	where likes > 45	db.mycol.find({"likes":{$gt:45}}).pretty()
Greater Than Equals	{<key>:{$gte:<value>}}	where likes >= 18	db.mycol.find({"likes":{$gte:18}}).pretty()
Not Equals	{<key>:{$ne:<value>}}	where likes != 17	db.mycol.find({"likes":{$ne:17}}).pretty()

8.10 Advanced queries using logical operators

8.10.1 AND in MongoDB

Syntax

- To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND –

  >db.mycol.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })

Example

```
>db.myVPVcol.find({$and:[{"by":"MasashiKishimoto"},{"title":"Naruto"}]}).pretty()
{
    "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
    "title":"Naruto",
    "description":"Japanese Manga",
    "by":"Masashi Kishimoto",
    "url":"http://www.anime.com",
    "tags":[
        "naruto",
        "anime",
        "manga"
    ],
    "likes":100
}
```

8.10.2 OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR –

```
>db.mycol.find(
  {
    $or: [
      {key1: value1}, {key2:value2}
    ]
  }
).pretty()
>db.myVPVcol.find({$or:[{"by":"MasashiKishimoto"},{"title":"Naruto"}]}).pretty()
{
      "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
      "title":"Naruto",
      "description":"Japanese Manga",
      "by":"Masashi Kishimoto",
      "url":"http://www.anime.com",
      "tags":[
            "naruto",
            "anime",
            "manga"
      ],
      "likes":100
}
```

8.10.3 Using AND and OR Together

Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>100 AND (by = 'Masashi Kishimoto' OR title = 'Naruto')'

```
db.mycol.find({"likes":{$gt:10}, $or:[{"by":" Masashi Kishimoto "},
{"title":"Naruto"}]}).pretty()
{
      "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
      "title":"Naruto",
      "description":"Japanese Manga",
      "by":"Masashi Kishimoto",
      "url":"http://www.anime.com",
      "tags":[
            "naruto",
            "anime",
            "manga"
      ],
      "likes":100
}
```


8.10.4 NOT in MongoDB

Syntax

```
>db.Collection_Name.find(
    {
        $NOT: [
            {key1: value1}, {key2:value2}
        ]
    }
).pretty()
```

8.11 Saving and Updating Document.

8.11.1 Update Document

MongoDB's save () as well as update () methods are used to update the documents into a collection/s. The save () method replaces the existing document with the document passed in save () method. The update () method updates the values in the existing document while the

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(1234548781331pada45ec5),"sname":"Manan Vaibhav Vasani"}
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
```

Following example will set the new title 'Manan P Vasani' of the documents whose title is 'Manan Vaibhav Vasani'.

```
>db.myVPVcol.update({'fname':'Manan Vaibhav Vasani'},{$set:{'fname':'Manan P Vasani'}})
WriteResult({"nMatched":1,"nUpserted":0,"nModified":1})
>db.myVPVcol.find()
{"_id":ObjectId(1234548781331pada45ec5),"sname":"Manan V Vasani"}
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
>
```

By default, MongoDB will update only a single document in a collection. But if you want to update multiple documents, you need to set a parameter 'multi' to true so it will update multiple documents.

```
>db.myVPVcol.update({'sname':'Manan V Vasani'},
{$set:{'sname':'Manan V Vasani'}},{multi:true})
```

8.11.2 MongoDB Save() Method

The save() method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB save() method is shown below –

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Example

Following example will replace the document with the _id '1234548781331pada45ec7'.

```
>db.myVPVcol.save(
{
"_id":ObjectId("1234548781331pada45ec7"),
        "sname":"Jia Chirag Vasani",
"Age": 02
}
)
WriteResult({
    "nMatched":0,
    "nUpserted":1,
    "nModified":0,
    "_id":ObjectId("1234548781331pada45ec7")
})
>db.myVPVcol.find()
{"_id":ObjectId(1234548781331pada45ec5),"sname":"Manan V Vasani", "Age": 02}
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
>
```

8.11.3 Delete Document

For deletion of document the remove () Method is use.

MongoDB's **remove**() method is used to remove a document from the collection. The remove () method accepts two parameters are as follows:

- **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

```
>db.Collection_Name.remove(DELLETION_CRITTERIA)
```

Example

Consider the myVPVcol collection has the following data.

```
{"_id":ObjectId(1234548781331pada45ec5),"sname":"Manan V Vasani"}
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
```



```
>db.myVPVcol.remove({'sname':'Manan V Vasani '})
WriteResult({"nRemoved" : 1})
>db.myVPVcol.find()
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
```

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

8.11.4 The Limit () Method

To limit the records in MongoDB, you need to use limit () method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

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

```
>db.COLLECTION_NAME.find().limit(NUMBER)
>db.myVPVcol.find({},{"sname":1,_id:0}).limit(2)
{"sname":"Manan"}
{"sname":"Jia"}
>
```

If you don't specify the number argument in limit() method then it will display all documents from the collection.

8.11.5 MongoDB Skip() Method

The skip() method accepts number type argument and is used to skip the number of documents.

Syntax

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Example

Following example will display only the second document.

```
>db.myVPVcol.find({},{"sname":1,_id:0}).limit(1).skip(1)
{"sname":"Jia"}
>
```

Please note, the default value in skip() method is 0.

8.12 Sorting

The sort() Method

The method accepts a document containing a list of fields along with their sorting order. 1 is used for ascending order while -1 is used for descending order.

Syntax:

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

Example

Consider the collection myVPVycol has the following data.

```
{"_id":ObjectId(1234548781331pada45ec5),"sname":"Manan V Vasani"}
{"_id":ObjectId(1234548781331pada45ec6),"sname":"Laksh A Vasani"}
{"_id":ObjectId(1234548781331pada45ec7),"sname":"Aryan V Vasani"}
{"_id":ObjectId(1234548781331pada45ec8),"sname":"Jia C Vasani"}
```

Following example will display the documents sorted by sname in the descending order.

```
>db.myVPVcol.find({},{"sname":1,_id:0}).sort({"sname":-1})
{"sname":"Manan V Vasani"}
{"sname":"Laksh A Vasani"}
{"sname":"Jia C Vasani"}
{"sname":"Aryan V Vasani"}
>
```

Please note, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

8.13 Simple Aggregate Functions

- Aggregation operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- The Simpleaggregate() Method
- For the aggregation in MongoDB, you should use aggregate() method.

Syntax

Basic syntax of aggregate() method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection you have the following data –

```
{
  _id:ObjectId(7df78ad8902c)
name:'Naruto',
  Bloodline:'Uzumaki',
likes:100
},
{
  _id:ObjectId(7df78ad8902d)
name:'Sasuke',
  Bloodline:'Uchiha',
likes:10
},
{
  _id:ObjectId(7df78ad8902e)
name:'Itachi',
  Bloodline:'Uchiha',
likes:750
},
```


Now from the above collection, if you want to display a list stating how many user from same bloodline are there, then you will use the following aggregate() method –

```
>db.myVPVcol.aggregate([{$group :{_id :"$Bloodline",num_people:{$sum :1}}}])
{"_id":"Uzumaki","num_people":1}
{"_id":"Uchiha ","num_people":2}
>
```

Following is a list of available aggregation expressions available with MongoDB.

Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from all documents in the collection.
$min	Gets the minimum of the corresponding values from all documents in the collection.
$max	Gets the maximum of the corresponding values from all documents in the collection.
$push	Inserts the value to an array in the resulting document.
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.

8.14 MongoDB Distributed environment

8.14.1 Replication

A replica set in MongoDB is a group of mongod methods that preserve the equal statistics set. Replica units offer redundancy and excessive availability, and are the premise for all production deployments.

Redundancy and Data Availability

- Replication gives redundancy as well as increases data availability. With more than one copies of information on distinct database servers, replication affords a level of fault tolerance towards the loss of a single database server.
- In a few instances, replication can offer elevated study capacity as clients can send read operations to different servers. Maintaining copies of statistics in distinct datacentre can increase statistics locality and availability for distributed programs. It maintains additional copies for dedicated functions, consisting of disaster recovery, reporting, or backup.a

8.14.2 Replication in MongoDB

- A replica set has many data bearing nodes and optional one arbiter node. Out of the data bearing nodes, one and only one member is considered the primary node, while the other nodes are considered secondary nodes.
- The primary node receives all write operations. The primary records all changes to its data sets in its operation log, i.e. oplog. See following figure.

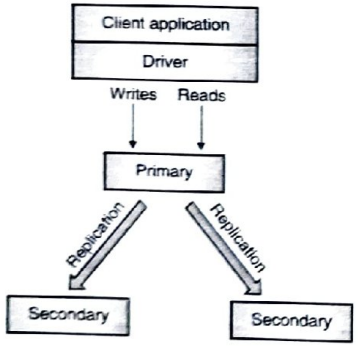


Fig. 8.14.1

- The secondary's replicate the primary's oplog and apply the operations to their data sets such that the secondary's data sets reflect the primary's data set. If the primary is unavailable, then an eligible secondary will hold an election to elect itself the new primary. See following figure.

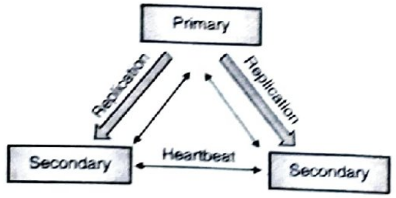


Fig. 8.14.2

- In sometime we may choose to add a mongod instance to a replica set as an arbiter. An arbiter participates in elections but does not hold data (i.e. does not provide data redundancy). See following figure.

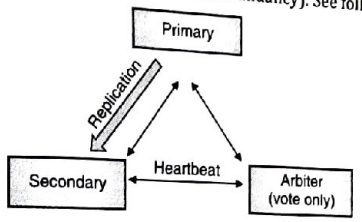


Fig. 8.14.3

- An arbiter will always be an arbiter whereas a primary may additionally step down and emerge as a secondary and a secondary might also emerge as the number one throughout an election.

8.14.3 Asynchronous Replication

Secondaries replicate the primary's oplog and apply the operations asynchronously. By having the secondaries' dataset reflect the primary's data set..

8.15 Sharding in MongoDB

- Sharding is a method for distributing data across many machines. Sharding is used for large deployments with very large data sets. For example, high read/write rates can exhaust the CPU capacity, stress on RAM and the I/O capacity of disk drives. System growth can be done using vertical and horizontal scaling.
- Vertical Scaling encompasses increasing the capacity of a single server, with more powerful CPU, to add additional RAM, or storage space.
- **Horizontal Scaling** encompasses dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade-off is increased complexity in infrastructure and maintenance for the deployment.

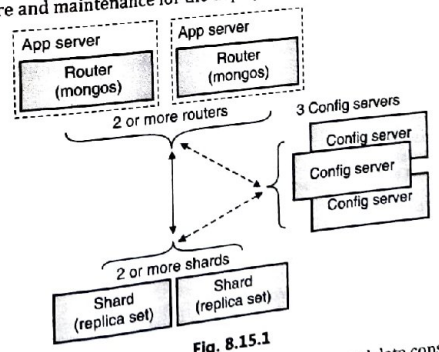


Fig. 8.15.1

- **Shards** – Shards are useful in storing data. It provides high availability and data consistency.

- **Query Routers** – Query routers are 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.
- **Config Servers** – It store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards.
- A sharded cluster has many query routers.
- MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Shard Keys

The shard key consists of a field/s in the documents. MongoDB uses the shard key to distribute the collection's documents across all the shards.

Chunks

- MongoDB partitions sharded data into chunks. Each chunk has an inclusive lower and exclusive upper range based on the shard key.
- Once a collection has been sharded, MongoDB provides no method to unshard a sharded collection. Careful consideration in choosing the shard key is necessary for ensuring cluster performance and efficiency.

Sharded and Non-Sharded Collections

A database may have a combination of sharded and unsharded collections. Sharded collections are partitioned and distributed across the shards in the cluster. Unsharded collections are stored on a primary shard. Each database has its own primary shard.

Model Questions

Q. 1 What are the key features of MongoDB?

Q. 2 Explain Key features of MongoDB?

Q. 3 Explain working of Sharding in MongoDB?

Q. 4 Explain working of replication in MongoDB?

Q. 5 Explainvarious datatypes used in MongoDB?

Q. 6 Explain CRUD Operation in MongoDB?

Q. 7 Write sample Document for facebook Post.

Q. 8 Explain MongoDB Database model

Q. 9 Write script for Tweeter Post to Create, Insert, Update and Drop.

Q. 10 What is MongoDB array? Explain its use.

Q. 11 Explain find() with its various options.

Q. 12 Enlist and explain the limitation of Sharding in MongoDB.

Q. 13 Enlist and explain Limitation of MongoDB.

□□□

9 Trends in Advance Database

Syllabus

Temporal database : Concepts, time representation, time dimension, incorporating time in relational databases.

Graph Database: Introduction, Features, Transactions, consistency, Availability, Querying, Case Study Neo4J, Spatial database: Introduction, data types, models, operators and queries

9.1 Temporal Database Concept

Q. Write a short note on : Temporal Database Concept **MU - Dec. 19, 10 Marks**

(i) Introduction

- Temporal database concept combines all database applications that make use of time aspect while arranging information.
- Generally database models maintain some aspect of the real world having the current state of data and do not store information about past states of the database.
- When the database state changes, the database gets updated and past information is automatically deleted but in many applications it is necessary to maintain past information.
- E.g. In medical database we need to keep patients' medical history for treatment of patients.
- Temporal database applications have been developed in very early age of database design, but the main design and development of such database are in the hands of designers and developers.

(ii) Examples

Applications that use temporal databases are as follows.

(a) Healthcare database

Keeps track of patients' medical history for further treatment.

(b) Airline reservation system

In general all reservation systems require all time- related information about reservation time, valid arrival time, departure time etc.

(c) Insurance database

History of all accidents and claims is stored along with corresponding time and date for further processing.

(d) Sales database

Sales database needs to maintain sales information along with data and time which may be helpful for further marketing decisions.

9.2 Temporal (Time)Representation

Time Specification/Temporal Data Types

(a) DATE : 'Date' data type stores year (yyyy) information, month (mm) information and day (dd) information. There are many formats available.

YYYY : MM : DD

DD : MM : YYYY

DD : MM : YYYY

Day : Mon : Year

(b) TIME : 'Time' data type stores time in the form of two digits for hours (HH) information, two digits for minutes (MM) information and two digits for seconds (SS) information.

Formats : HH : MM : SS

HH : MM – SS

(c) TIMSTAMP : 'Timestamp' combines above two data types together to store complete time information.

Format : DD : MM : YY HH : MM : SS

(d) INTERVAL : 'Interval' refers to a period of time. It may span of a few days, months or years.

9.3 Time Dimensions in Relational Data

9.3.1 Valid Time Temporal Database

- Valid time is defined as time at which particular event occurred or duration during which particular event is considered to be true.
- A temporal database that uses valid time is called as valid time temporal database.

Emp_validTime (Emp_VT)

Eid	Ename	Salary	DNo	VST	VET

VST – valid start time

VET – valid end time

Valid time database schema

- In the above relation (EMP_VT) the non temporal key (Eid) and valid start time (VST) is treated as new primary key

Eid	Ename	Salary	DNO	VST	VET
1	Akshaya	50000	10	01-01-99	15-12-99
2	Amrata	20000	40	01-01-01	01-01-02
3	Pallavi	15000	50	01-01-98	01-01-99
4	Bhavana	75000	30	31-12-98	01-01-01
5	Shubhra	25000	20	02-02-01	02-02-10

- Whenever one or more attributes of above employee table is updated

(1) System can overwrite the old values as in case of simple (non temporal) databases

OR

(2) System can cart new version and close current version by changing its valid end time (VET) to end time.

- In these cases we generally go for second approach i.e. creating new version instead of overwriting.

9.3.1(A) Transaction Time Temporal Database

- Transaction time is defined as time at which a particular event is actually recorded/stored in database.
- A temporal database that uses transaction time called as transaction time temporal database.

Emp_TransTime (Emp_TT)

Eid	Ename	Salary	DNo	TST	TET

TST – transaction start time

TET – transaction start time

Transaction time database schema

- In above relation the non temporal key (Eid) and transaction start time (TST) is treated as new primary key.

Example :

Emp_TT

Eid	Ename	Salary	DNo	TST	TET
1	Akshaya	50000	10	01-01-99	15-12-99
2	Amrata	20000	40	01-01-01	01-01-02
3	Pallavi	15000	50	01-01-98	01-01-99
4	Bhavana	75000	30	31-12-98	01-01-01
5	Shubhra	25000	20	02-02-01	02-02-10

- A transaction time database has also been called as rollback database, because user can logically rollback to original database state at any past point in time T by deriving all tuple version U whose transaction turn [U.TST, U.TET] include point T.

9.3.1(B) Bi-temporal Database Schema

- In some cases one of above time is required but in many cases we require both of above time dimensions. Such time is called as bitemporal time.
- The database that uses above time is called as Bitemporal time database schema.

Emp_BT

Eid	Ename	salary	DNo	VST	VET	TST	TET

Bitmporal time temporal database schema

- In above relation (Emp_BT) the non temporal key (EID) and transaction start time (TST) together act as primary key

Eid	Ename	Salary	DNo	VST	VET	TST	TET
1	Akshaya	50000	16	1-1-98	Now	30-10-98	15-10-99

Eid	Ename	Salary	DNo	VST	VET	TST	TET
2	Armuta	20000	40	1-1-99	15-12-99	15-10-99	UC
3	Pallavi	15000	50	16-12-01	15-10-01	15-10-01	UC
4	Bhavana	75000	30	15-01-01	15-02-01	15-02-01	UC
5	Shubhra	25000	20	10-02-09	17-03-09	17-03-09	UC

- As shown in above table, tuple whose transaction end time (TET) is UC are representing currently valid information

(d) User defined time temporal databases

- The user can define own semantics and program for application appropriately and it is called as user defined time.
- The temporal databases using user defined time are known as user defined time temporal databases.

Emp-UT

Eid	Ename	salary	DNo	TST	TET

9.4 Incorporating Time in relational Databases

(a) Storing data

There are two options for storing tuples in temporal databases

(i) Save all data in same table

(ii) Save all valid information in one table and other in second table

(iii) Vertical partition – The attribute of temporal relation can be sub divided in vertical columns, but for above partition for synchronizing data we need temporal intersection join which is expensive to implement.

(b) Append only database

(i) Complete record of changes

It is important that bitemporal databases allow complete record of changes

(ii) Correction

If more than one tuple with same valid time but different attributes value and transaction times are disjoint, in this way we can correct the incorrectly entered value data.

(iii) A database that keeps such a complete record of changes and corrections has been called an append only database.

9.4.1 Temporal Query Language

- **Temporal selection :** we can select temporal data which involves time attributes.
- **Temporal projection :** we can project temporal data in the projection and inherit those time intervals from tuple in original relation.
- **Temporal join :** Intersection of time interval of original tuples is derived. The empty tuples are discarded from join.
- **Temporal functions dependency :** Adding time dimension may invalidate functional dependency.

  $X \rightarrow Y$ on relation R

  For all instances i for R all snapshot of i satisfy functional dependency $X \rightarrow Y$

9.5 Graph Database Concept

(i) Introduction

- Graph Databases, arranges the data in the form of a graph, based on the principle of graph theory.
- Graph is a collection of nodes and edges as show in below diagram.
- Data is stored as a graph and their relationships are stored as a link between them whereas entity acts like a node.
- The main point of graph databases are the relationships between data is important as that of data itself
- Graph data is schema free it means there is no limitation of database schema on the data stored.

(a) Node

- Nodes are the entities in the scenario.
- Graphs can hold many attributes (key-value pairs) also called properties.
- Nodes are tagged with labels, representing their different roles in domain.
- Node labels serve as metadata to certain nodes.

(b) Relationship

- Relationships provide direction and name to connections between two node entities
- e.g. Teacher TEACHES Student
- A relationship has a start node and an end node.
- Relationships have properties associated with it.
- Two nodes can have any number of relationships without sacrificing performance.

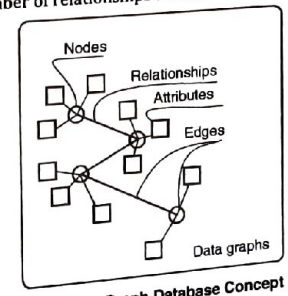


Fig. 9.5.1 : Graph Database Concept

(ii) Features

Naive Graph Storage

- This generation of Graphs also offer storage of data within the nodes and relationships
- It makes it ready for future use to save huge data store and faster access.

Parallel Loading Operation

- The loading of data will take large amount of time for terabytes of data.

- Nowadays, graph database is enough speed to work very faster.
- The data that takes few days to upload now takes just few hours.

Performance

- Data is going to increase a lot in coming future, so it automatically increases relationships between data.
- Big data is getting bigger day by day, but connected data will grow exponentially in near future.
- Unlike in traditional database the performance of graph database stays constant even as your data grows year over year.

Good Speed and scale up

- Sharded graphs and other features improves speed for processing of data.
- Distribution and analytics has provided good scale up for handling datasets in Terabytes.

Flexibility

- Graph databases can move at the speed of business because the structure and schema of a graph data model very flexible as your industry changes.
- The developer efforts can be reduced by reducing rework due to flexibility in schema.

Agility

Graph database technology is perfect match for today's agile methodology, test-driven development practices, allowing your graph-database to evolve with your changing business requirements.

9.5.1 Performance comparison of Data Models vs graph database

Database model	Performance	Scalability	Flexibility
Key value store database	High	High	High
Column store database	High	High	Moderate
Document store database	High	Variable (High)	High
Graph database	Variable	Variable	High

9.5.2 Graph Mining

- Graphs have become increasingly important in modeling various application and their interactions like chemical informatics, bioinformatics, computer vision, web analysis, text retrival etc. frequent subgraph patterns can be mined for further classification and clustering of important tasks.
- Graphs can be used to represent different kind of networks like biological network computer network, web and social community network etc. Many graph search algorithms have been developed in chemical informatics, text retrieval, computer vision and video indexing.
- With increasing need on analysis of large structured data, graph mining has become an active and important theme in data mining.
- Frequent substructures are the most basic pattern to be discovered in a collection of graph. Substructures are useful for characterizing and finding different graphs of graphs.
- Classifying and clustering graphs, building graph indices and searching graph database for similarity.
- Graph mining methods and algorithms are widely studied now a days in various applications to discover interesting patterns.

- **Example :** Discovery of active chemical structures in HIV screening datasets is possible using graph mining of frequent graphs between different classes.
- Graph mining can also be performed on large frequent sub pathways in metabolic networks, to classify chemical compounds, on the frequent graph mining technique to study protein structural families etc.

9.5.3 Neo4j Graph Databases

(i) Introduction

- Neo4j is a graph database management system
- This system offers us the ACID-compliant transactional database.
- In Neo4j, stores all data in the form of an edge, node, or attribute.
- Every node and edge can have any number of attributes.
- In system nodes and edges can be labelled, Labels can be used to narrow searches and make it faster.
- **Cypher** query language is used to manipulate data in Neo4J the graph database.
- Neo4j has naive data storage systems.
- Indexing was added also added to Cypher with the introduction of schemas.

(ii) Cypher : Querying in Graph Databases

- Cypher is a query language depending on patterns and is designed to find some data from available dataset
- Cypher is very simple and logical language for learners.
- Cypher contains a variety of keywords for specifying patterns, filtering patterns, and returning results.
- Most common patterns used are: MATCH, WHERE, and RETURN
- These operate slightly differently than the SELECT and WHERE in SQL.

(iii) Cypher : Transactions in Graph Databases

- A database transaction is a series of database operation that are treated as a single unit of action.
- These actions either should be completed successfully completely or should not take any effect at all.
- Transaction management can be broken down into four properties known as ACID properties.
- To enforce data integrity and make sure good transactional behavior, Neo4j supports the 4 ACID properties:
- Atomicity: If a part of a transaction fails, the data in the database is left unchanged.
- Consistency: Any transaction will leave the database in a consistent state.
- Isolation: When a transaction is happening, modified data cannot be accessed by other operations.
- Durability: The DBMS can always recover the results of a committed transaction.
- All database operations that access the graph or schema must be performed in a transaction.
- Data accessed by traversals is not protected from alterations by other transactions.
- Non-repeatable reads are possible in Neo4j database.
- One can manually acquire write locks on nodes and relationships to achieve higher level of isolation.
- Locks are acquired by transaction at transaction level on the Node and Relationship.
- Neo4J has built in deadlock detection.

9.5.4 Consistency and Availability

Please refer section 7.8 in chapter 7 of unit 4.

9.6 Spatial Databases Introduction

1. Introduction

- The spatial DBMS makes spatial data management simple for user and application.
- A spatial database supports various concepts for databases which keep information of objects in a multidimensional region.
- There are limited set of data types and operation are available for spatial applications which makes the modeling of real world spatial applications extremely difficult.
- Common example of spatial data is map of railway tracks. This map is two dimensional objects that contain points and lines that can represents route of railway and cities.

2. Spatial Database Model

(a) Cartographic Databases

This databases store maps include two dimensional spatial descriptions of their objects from countries and states to rivers, cities, roads, seas, and so on.

(b) Meteorological Databases

Weather information is 3D, since temperatures and other meteorological information are related to three dimensional spatial points.

3. Spatial Database Management

- The spatial relationships among the objects are important, and they are often needed when querying the database.
- Some extensions that are needed for spatial databases are models that can interpret spatial characteristics.
- In addition, special indexing and storage structures are often used for improving performance.
- The basic types of extensions required to be include 2 dimensional geometric concepts, Like points, lines, circles and polygons in order to specify the spatial characteristics.

Performance factors

- For better performance special techniques for spatial indexing is needed.
- One of the best known techniques for it is use of RV trees and their variations.
- RV trees group together objects that are in close spatial physical proximity on the same leaf nodes of a tree-structured index.
- Typical criteria for dividing the space include minimizing the rectangle areas, since this would lead to a quicker narrowing of the search space.

9.6.1 Spatial Data types

Q. Explain different types of Spatial Data models **MU – Dec. 19, 10 Marks**

(a) Point Data

- A point has a spatial extent characterized completely by its position or location.
- Point data can be collection of points in multidimensional space.

- Point data stored in a database can be based on direct measurement or data obtained through measurements.
- **Raster data** is example of directly measured point data.

(b) Region Data

- Region data has a spatial extent represented by location and boundaries.
- The location can be shown as the position of fixed points for the region.
- The boundary can be represented as a line in 2D space and as surface in 3D space.
- Region data consist of collection of regions in multidimensional space.
- Region data is nothing but a simple geometric approximation to an actual data object.
- **Vector data** is example of region data.

9.6.2 Spatial Database Queries and Operators

(a) Spatial Range query

- Spatial range queries having an associated region for it.
- These queries search for the objects of a particular type that are within a given area.

Example

(i) Find all hotels within 500 Km of Mumbai.
(ii) Find all pubs in Mumbai.

(b) Nearest neighbour query

Finds an object of a particular type that is closest to a given point or location.
Answer can be ordered by the distance from object.
Such Queries are very important in context of multimedia databases.
Example : find 10 water parks near to Mumbai.

(c) Spatial joins queries

Typically joins the objects of two types based on some spatial condition, such as the objects intersecting or being within a certain distance of one another.
If more detailed information is given about record then query may becomes more complex.
Example : find pair of cities that are within two miles of each other.
In above example each record is point representing a city, Query can be answered by self join

9.6.3 Spatial Database Applications

(a) GIS

- These applications are also known as Geographical Information Systems (GIS), and are used in areas such as environmental, emergency, and battle management.
- Point data and region data must handle properly.
- Arc Info is widely used GIS Software.

(b) CAD/CAM

- Spatial Objects such as surface of design objects.

- Point data and region data used extensively.
- Range and spatial queries are commonly used.

(c) Multimedia Database system

- They contain objects like images, audio and video and various types of time series data.
- Multimedia data is mapped to collection of points in which distance between them is very important.

Review Questions

Q. 1 Give various applications of spatial databases.

Q. 2 Write short notes on : Mobile databases

Q. 3 Write a note on architecture of Mobile databases.

Q. 4 Write short notes on : Temporal database and Mobile database.

Q. 5 Explain design and implementation issues in mobile databases.

Q. 6 What are types of Temporal databases.Q.What is Temporal Database? What are its characteristics?

Q. 7 Explain Temporal databases with suitable example.

Q. 8 What is Temporal Database? What are its characteristics?

Q. 9 Explain Temporal databases with suitable example.

Q. 10 Give various applications of graph databases.

Q. 11 Write short notes on : Neo4J

Q. 12 What is Graph Database? What are its characteristics?

❑❑❑

Note