Introduction to Distributed Databases

DDB Technology :

Merger of two technologies:

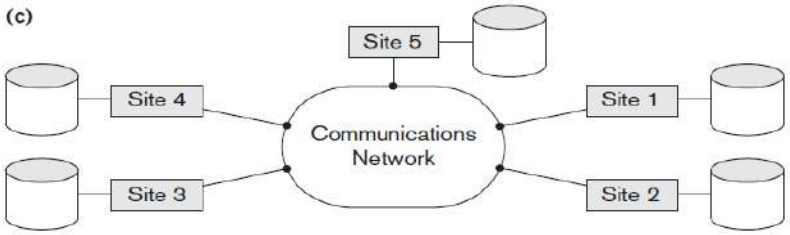
- database technology, and
- network and data communication technology.

Computer networks allow distributed processing of data.

Traditional databases, on the other hand, focus on providing centralized, controlled access to data.

Distributed databases allow an integration of information and its processing by applications.

- **Distributed database (DDB) as a collection of multiple logically** interrelated databases distributed over a computer network,
- **Distributed database management system (DDBMS) as a software system that manages a distributed** database while making the distribution transparent to the user.



When database to be called distributed ?

- **There are multiple** computers, called **sites or nodes. These sites must be connected by an** underlying **communication network to transmit data and commands** among sites, as shown
- ■ **Logical interrelation of the connected databases- It is essential that the** information in the databases be logically related.
- ■ **Absence of homogeneity constraint among connected nodes. -It is not necessary** that all nodes be identical in terms of data, hardware, and software.

Transparency

- The internal details of the distribution are hidden from the users(hiding implementation details from end users.)

Data organization transparency (also known as *distribution or network transparency).*

This refers to freedom for the user from the operational details of the network and the placement of the data in the distributed system.

- It may be divided into location transparency and naming transparency.
- **Location transparency refers to the fact that the command used to perform** a task is independent of the location of the data and the location of the node where the command was issued. (user need not be aware about physical location of database)
- **Naming transparency implies that once a** name is associated with an object, the named objects can be accessed unambiguously without additional specification as to where the data is located. (user need not be provide any additional information about name of database)

Replication transparency.

- **copies of the same** data objects may be stored at multiple sites for better availability, performance, and reliability.
- user unaware of the existence of these copies.

Fragmentation transparency.

- **Two types of fragmentation are possible.**
- **Horizontal fragmentation-- distributes a relation (table) into subrelations** that are subsets of the tuples (rows) in the original relation.
- **Vertical fragmentation** -distributes a relation into subrelations where each subrelation is defined by a subset of the columns of the original relation.

design transparency and execution transparency—

- referring to freedom from knowing how the distributed database is designed and where a transaction executes.

Autonomy

- Autonomy determines the extent to which individual nodes or DBs in a connected DDB can operate independently
- **Design autonomy refers** to independence of data model usage and transaction management techniques among nodes.
- **Communication autonomy determines the extent to which each** node can decide on sharing of information with other nodes.
- **Execution autonomy** refers to independence of users to act as they please.

Reliability and Availability

- **Reliability** is broadly defined as the probability that a system is running (not down) at a certain time point,
- **Availability** is the probability that the system is continuously available during a time interval

Advantages of Distributed Databases

- **Improved ease and flexibility of application development.**
- Developing and maintaining applications at geographically distributed sites of an organization is facilitated due to transparency of data distribution and control.
- **Improved performance- Data localization reduces** the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks.

An advantage of distributed databases is that even when a portion of a system (i.e. a local site) is down, the overall system remains available. With replicated data, the failure of one site still allows access to the replicated copy of the data from another site. The remaining sites continue to function. The greater accessibility enhances the reliability of the system.

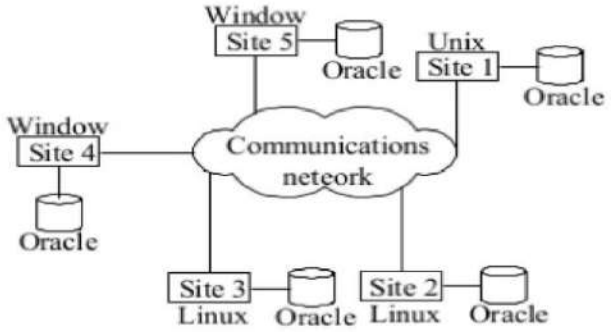
(2) Capacity and Growth

An advantage of distributed databases is that as the organisation grows, new sites can be added with little or no upheaval to the DBMS. Compare this to the situation in a centralised system, where growth entails upgrading with changes in hardware and software that effect the entire database.

Types of Distributed Database Systems

Homogeneous

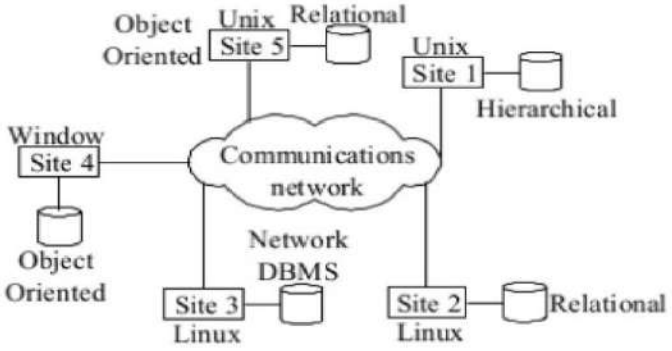
All sites of the database system have identical setup, i.e., same database system software. The underlying operating system may be different. For example, all sites run Oracle or DB2, or Sybase or some other database system. The underlying operating systems can be a mixture of Linux, Window, Unix, etc. The clients thus have to use identical client software.



Heterogeneous

Federated: Each site may run different database system but the data access is managed through a single conceptual schema. This implies that the degree of local autonomy is minimum. Each site must adhere to a centralized access policy. There may be a global schema.

Multidatabase: There is no one conceptual global schema. For data access a schema is constructed dynamically as needed by the application software.



Distributed Data Storage

Consider a relation r that is to be stored in the database.

There are two approaches to storing this relation in the distributed database:

Replication.

The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site.

Fragmentation.

The system partitions the relation into several fragments, and stores each fragment at a different site.

Data Replication

- If relation r is replicated, a copy of relation r is stored in two or more sites.

Advantages and disadvantages to replication

1)Availability

2)Increased parallelism :

- **The no of transactions can read relation r in parallel**
- **The more replicas of r there are, the greater the chance that the needed data will be found in the site where the transaction is executing.**

3) Increased overhead on update.

- **The system must ensure that all replicas of a relation r are consistent; otherwise, erroneous computations may result.**
- **Thus, whenever r is updated, the update must be propagated to all sites containing replicas. The result is increased overhead.**

Data Fragmentation

- If relation r is fragmented, r is divided into a number of fragments

  r1, r2, . . . , r n.

- These fragments contain sufficient information to allow reconstruction of the original relation r.

Horizontal fragmentation,

- **a relation r is partitioned into a number of subsets,**

r1, r2, . . . , r n.

Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

keep tuples at the sites where they are used the most, to minimize data transfer.

defined as a selection on the global relation r.

That is,we use a predicate Pi to construct fragment ri:

$$r_i = \sigma_{P_i}(r)$$

Example : Horizontal fragmentation

- Consider the account relation
- Account = (acc_no, branch_name, balance)
- If the banking system has only two branches - Hillside and Valley view, then there are two different fragments :

$$account1 = \sigma_{branch\_name = \text{"Hillside"}}(account)$$
$$account2 = \sigma_{branch\_name = \text{"Valleyview"}}(account)$$

Vertical fragmentation

- Vertical fragmentation split the relation by decomposing the scheme R of relation 'r'.
- Vertical fragmentation of r(R) involves the definition of several subset of attributes R1, R2, ... Rn, of the scheme R so that

$$R = R_1 \cup R_2 \cup \ldots\ldots\ldots \cup R_n$$

Each fragment r_i of 'r' is defined by

$$r_i = \Pi_{R_i}(r)$$

We reconstruct relation 'r' by taking the natural join; i.e.

$$r = r_1 \bowtie r_2 \bowtie \ldots\ldots \bowtie r_n$$

Table name: CUSTOMER **Database name: Ch12_Text**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	Sinex, Inc.	12 Main St.	TN	3500.00	2700.00	3	1245.00
11	Martin Corp.	321 Sunset Blvd.	FL	6000.00	1200.00	1	0.00
12	Mynux Corp.	910 Eagle St.	TN	4000.00	3500.00	3	3400.00
13	BTBC, Inc.	Rue du Monde	FL	6000.00	5890.00	3	1090.00
14	Victory, Inc.	123 Maple St.	FL	1200.00	550.00	1	0.00
15	NBCC Corp.	909 High Ave.	GA	2000.00	350.00	2	50.00

Suppose that XYZ Company's corporate management requires information about its customers in all three states

, but company locations in each state (TN, FL, and GA) require data regarding local customers only.

Based on such requirements, you decide to distribute the data by state.

Suppose that XYZ Company's corporate management requires information about its customers in all three states

, but company locations in each state (TN, FL, and GA) require data regarding local customers only. Based on such requirements, you decide to distribute the data by state.

FRAGMENT NAME	LOCATION	CONDITION	NODE NAME	CUSTOMER NUMBERS	NUMBER OF ROWS
CUST_H1	Tennessee	CUS_STATE = 'TN'	NAS	10, 12	2
CUST_H2	Georgia	CUS_STATE = 'GA'	ATL	15	1
CUST_H3	Florida	CUS_STATE = 'FL'	TAM	11, 13, 14	3

Table name: CUST_H1 **Location: Tennessee** **Node: NAS**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	Sinex, Inc.	12 Main St.	TN	3500.00	2700.00	3	1245.00
12	Mynux Corp.	910 Eagle St.	TN	4000.00	3500.00	3	3400.00

Table name: CUST_H2 **Location: Georgia** **Node: ATL**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
15	NBCC Corp.	909 High Ave.	GA	2000.00	350.00	2	50.00

Table name: CUST_H3 **Location: Florida** **Node: TAM**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
11	Martin Corp.	321 Sunset Blvd.	FL	6000.00	1200.00	1	0.00
13	BTBC, Inc.	Rue du Monde	FL	6000.00	5890.00	3	1090.00
14	Victory, Inc.	123 Maple St.	FL	1200.00	550.00	1	0.00

- For example, suppose that the company is divided into two departments: the service department and the collections department.
- Each department is located in a separate building, and each has an interest in only a few of the CUSTOMER table's attributes.

VERTICAL FRAGMENTATION OF THE CUSTOMER TABLE

FRAGMENT NAME	LOCATION	NODE NAME	ATTRIBUTE NAMES
CUST_V1	Service Bldg.	SVC	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE
CUST_V2	Collection Bldg.	ARC	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE

Table name: CUST_V1 **Location: Service Building** **Node: SVC**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE
10	Sinex, Inc.	12 Main St.	TN
11	Martin Corp.	321 Sunset Blvd.	FL
12	Mynux Corp.	910 Eagle St.	TN
13	BTBC, Inc.	Rue du Monde	FL
14	Victory, Inc.	123 Maple St.	FL
15	NBCC Corp.	909 High Ave.	GA

Table name: CUST_V2 **Location: Collection Building** **Node: ARC**

CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	3500.00	2700.00	3	1245.00
11	6000.00	1200.00	1	0.00
12	4000.00	3500.00	3	3400.00
13	6000.00	5890.00	3	1090.00
14	1200.00	550.00	1	0.00
15	2000.00	350.00	2	50.00

22. Consider a database of a company where a relation `Employee (name, address, salary, branch)` is fragmented horizontally by branch. Assume that each fragment has two replicas: one stored at site A and one stored locally at the branch site. Describe a good strategy for the following queries entered at site B:

(i) Find the name and address of all employees at site C.
(ii) Find the average salary of all employees.
(iii) Find the lowest-paid employee in the company.

Ans: **(i)** Send the query $\pi_{name,\ address}(Employee)$ to site C and send back the result to site B.
(ii) Send the query $\mathcal{E}_{AVG(price)}(Employee)$ to the site A as it contains a copy of all the replicas, compute the average salary at site A and send back the result to site B.
(iii) Send the query to find the lowest-paid employee to site A, compute the query at site A and send back the result to site B.

Distributed Transactions

There are two types of transaction

- The local transactions are those that access and update data in only one local database
- The global transactions are those that access and update data in several local databases.
- Each site in a distributed system has transaction manager & transaction coordinator

The function of transaction manager is :

- Manage the execution of those transactions that access data stored in a local site.
- Ensure ACID properties of those transactions that executes at that site .

The transaction may be either a local transaction or part of a global transaction.

Each transaction manager is responsible for :

- Maintaining a log for recovery purposes.
- Participating in an appropriate concurrency - control scheme to co-ordinate the concurrent execution of the transactions executing at that site.

The function of the transaction co-ordinator is :

- Coordinate the execution of the various transactions (both local and global) initiated at that site.

  The transaction coordinator is responsible for coordinating the execution of all the transactions initiated at that site. For each such transaction, the coordinator is responsible for :

- Starting the execution of the transaction.
- Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution.
- Coordinating the termination of the transaction which may result in the transaction being committed at all the sites or aborted at all sites.

Commit Protocols

Transaction in distributed system must either commit at all sites where transaction is processing or it must be aborted at all sites.

To ensure this property the transaction coordinator of transaction T must execute a commit protocol.

Two-phase commit protocol (2PC)

Two-phase commit protocol ensure atomicity,

it handles failures and it carries out recovery and concurrency.

Working of two-phase commit protocol

Consider a transaction T initiated at site the execution of transaction T is control by coordinator Ci , and site Si,.

when all the sites involved in transaction processing inform transaction coordinator Ci that T has completed. transaction T completes its execution .

- After getting this information transaction coordinator Ci starts 2PC protocol.

Phase 1

- Ci, adds the record <prepare 'T'> to the log and forces the log on stable storage.
- Ci, sends a prepare T message to all sites where T is executed.
- After receiving prepare T message, the transaction manager at that site determines whether it is willing to commit its portion of transaction T.
- If answer of commit decision is no, the site adds record <no T> to the log, and respond to transaction coordinator by sending an abort T message.
- If answer of commit decision is yes, the site adds a record <ready T> to the log and respond to transaction coordinator by sending an ready T message.

Phase 2

- When Ci, receives responses to the prepare T message from all the sites.
- Ci , can determine whether to commit the transaction or abort the transaction.
- Transaction T can be commited if transaction coordinator Ci, received ready T message from all the participating sites.
- Otherwise transaction is aborted.
- If transaction coordinator decides to commit the transaction, a < commit T > record is added to the log and it sends commit T message to all the participating sites.
- • If transaction coordinator Ci decides to abort the transaction, a<abort T> records is added to the log and it sends abort T message to all the participating sites.
- A site at which transaction T executed can unconditionally abort T at any time before sending the message ready T to the coordinator.
- Once the ready T, message is sent
- it is a promise by a site to follow the coordinators order to commit T or to abort T.

- In same implementation of 2PC, a site sends an acknowledge T message to the coordinator at the end of second phase.
- When coordinator receives acknowledge T message from all the sites it adds the record <complete T> to the log.

How failures are handled in 2PC

- Failure of participating site
- If co-ordinator Ci detects a site has failed, it takes following action.
- If the site fails before responding with a ready T message, the coordinator assumes that it responded with an abort T message.
- If the site fails after the coordinator received the ready T message from the site, the coordinator process commit protocol in normal fashion.
- When a failed site Sk recovers from a failure, it examine its log to determine the fate of those transactions that were in the middle of execution.
- If the log contains a <commit T> record,
  the site must execute redo (T).
- If the log contains an <abort T> record,
  the site must execute undo (T).
- If the log contains a <ready T> record,
  the site must consult coordinator Ci to determine the fate of transaction T.
- If the coordinator is down Sk site try to find the fate of transaction from other sites.
- It does this by sending query status T message to all the sites.
- If any site has this information, it notify Sk about it.
- If no site has this information the Sk can neither abort nor commit T and decision about T is postponed till Sk obtained needed information.
- If the log contains no control records like abort, commit or ready about transaction T
- i.e. site Sk failed before responding to prepare T message. Hence Sk must execute undo as per our algorithm

Network partition

When network partitions happen in the system there are two possibilities as follows.

1. The coordinator and all its participants remain in one partition. Due to such partition there is no effect on the commit protocol.

2. The coordinator and its participants are present in different partitions.

The partition in which coordinator and some sites are present, consider that the sites is other partitions have failed.

The sites that are not in the partition containing the coordinator simply execute the protocol to deal with failure of the coordinator.

Three-Phase Commit Protocol

- It is an extension of the two-phase commit protocol for avoiding blocking problem of 2PC protocol under certain assumptions.
- Assumption In 3PC

1. No network partition occurs in the system.

2. Not more than k sites fail, here k is some predetermined number.

- This protocol avoids blocking by adding third phase.
- Coordinator first ensures that at least k other sites get information that coordinator has taken decision to commit the transaction and then it writes the commit record on the log.
- If the coordinator fails, the remaining sites first select the coordinator .
- New co-ordinator checks the status of protocol.
- If any one site from the k sites will be up and has the information about commit decision taken by previous coordinator.
- If newly selected coordinator gets this information from the remaining site, then it restarts the third phase otherwise it aborts the transaction

Disadvantage of 3PC

- Though it avoid blocking problem, if k sites are not failed, but when partitioning of network happens in the system or more than k sites will fail this situation lead to blocking.
- It is having more overhead as compared to 2PC.

Concurrency Control in Distributed Database

- Concurrency control schemes are based on the serializability property.
- The most common method used to implement serializability is to allow locks on items.
- The protocols used is distributed database require updates to be done on all replicas of a data item.

Locking Protocols

To use locking protocol in the distributed database, the change is required in lock manager process to deal with replicated data.

Following are the different locking protocols.

1) Single lock-manager approach

- In this approach, the system maintains a single lock manager which resides in a single chosen site.
- All lock and unlock requests are made at that site.
- Consider Si is the site where lock manager is residing.
- A transaction which needs to lock a data item sends a lock request to Si.
- If lock can be granted immediately, the lock manager sends message to the site at which lock request was initiated.

- If lock cannot be granted immediately, the request is delayed until it can be granted
- After getting the lock the transaction can read the data item from any one site at which a replica of the data item resides.
- In the case of write operation, all sites where a replica of the data item resides must be involved in the writing.

Advantages

1. Simple implementation

- Only two messages required for handling lock request
- One message required for handling lock release.

2. Simple deadlock handling

- Deadlock handling is simple as all lock and unlock requests are made at one site.

Disadvantages

- 1. Bottleneck Since all lock requests are processed at single site, the lock manager site becomes bottleneck.
- 2. Vulnerability : If the site fails at which lock manager is residing, the concurrency controller is lost.

2) Distributed lock manager

- The lock-manager function is distributed over several sites.
- Each site has a local lock manger which administer the lock and unlock requests for data items that are stored in that site.
- A transaction which needs to lock a data item Q which is not replicated and resides at site Si sends a message to lock manager at site Si.
- If lock can be granted immediately lock manager at Si, sends a message to the initiator indicating that is has granted the lock request.
- If lock can not be granted immediately then the request is delayed until it can be granted.

Advantages

1. Simple implementation

- Only two messages required for handling lock request.

- Only one message required for handling lock release.
-

2. Reduces bottleneck

- As lock manager function is distributed over several site, requests are processed on several sites and there is no bottleneck for lock manager.

3. Removes the vulnerability problem.

Disadvantages

1. Deadlock handling is complex.

- It is complex because the lock and unlock requests are handled and made at different site. There may be intersite deadlocks.

3) Primary copy

If the data is replicated on the several site we can choose one of the replicas as the primary copy.

- For each data item Q, the primary copy of Q must reside at only one site which is called as primary site of Q.
- A transaction which needs to lock data item Q, sends the request at the primary site of Q.

Advantages of primary copy

I. It enables concurrency control for replicated data so that it can be handled same like unreplicated data.

II. It allows simple implementation. Disadvantages of primary copy I. If primary site of Q fails, Q is inaccessible even though other sites containing a replica may be accessible.

4) Majority protocol

If data item Q is replicated in n sites, then a lock request message more than one-half of the n sites in which Q is stored.

The transaction does not acquire lock on Q until it has successfully on a majority of the replicas of Q (more than one-half of n sites). must be sent to obtained a lock

Advantage

- This scheme deals with replicated data in a decentralized manner which avoids the drawbacks of central control.

Disadvantages

1. Implementation

- This protocol is complicated to implement.
- It required 2(n/2+1) message for handling lock request.
- It requires (n/2+1) messages for handling lock release.

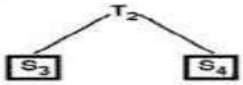
2. Deadlock handling

- complex because is addition to the problem of global deadlocks which are due to the use of distributed lock manager.
- Deadlocks are possible even if only one data item is being locked.

e.g. Consider a system with 4 sites and full replication of data item Q.

- Two transaction T_1 and T_2 requires data item Q in exclusive mode.
- Transaction T_1 may succeed is locking Q at site S_1 and S_2.

- Transaction T_2 may succeeded is blocking Q at site S_3 and S_4.



Both transaction T_1 and T_2 must wait to acquire third lock hence deadlock occurs.

5) Biased protocol

- In this protocol request for shared locks are given more favourable treatment than request for exclusive locks.
- Shared locks : If transaction wants to lock data item Q it requests a lock on Q from the lock manager at one site that contain a replica of Q.
- Exclusive locks: If transaction wants to lock data item Q, it request a lock on Q from all sites that contain a replica of Q.

Advantage

- Less overhead on read operation than the majority protocol.
- Disadvantages

1. The addition overhead on write operation.

2. Complexity in handling deadlocks.

6) Quorum consensus protocol

This protocol is a generalization of the majority protocol.

- It assigns each site a non-negative weight.
- It assigns two integers on data item x, called read quorum Q_r and write quorum Q_w for read and write operation respectively

These values must satisfy following conditions :

1) $Q_r + Q_w > s$

2) $2 * Q_w > s$

Where s is total weight of all sites at which x resides.

- For read operation, enough replicas must be read so that their total weight is $\geq Q_r$
- For write operation, enough replicas must be written so that their total weight is $\geq Q_w$.

Advantages

This protocol can permit the cost of either reads or writes to be selectively reduced by appropriately defining the read and write quorums.

If higher weights are given to some sites, fewer sites need to be accessed for acquiring locks.

By setting weights and quorums appropriately, the quorum consensus protocol can simulate the majority protocol and the biased protocols.

2.7.2 Timestamping

The idea behind timestamping is that each transaction is given a unique timestamp that the system uses in deciding serialization order.

In distributed system the need is to develop a scheme for generating unique timestamp.

There are two methods for generating unique timestamp.

1. Centralized
2. Distributed

2.11 Directory Systems

A directory is a listing of information about some class of objects. Directories can be used to find information about a specific object, or to find objects that meet a certain requirement. Today directories need to be available over a computer network e.g. organization can create directories of employees.

2.11.1 Directory Access Protocol

Directory information can be made available through web interfaces. People can access these directory information sometimes, programs also access directory information.

Directories can be used for storing other type of information e.g. web browsers can store personal bookmarks and other settings in directory system which can be used by user to access some settings from multiple locations.

Today **Lightweight Directory Access Protocol (LDAP)** is the most widely used directory access protocol. In the previous examples we have seen data can be stored is database and accessed through protocols like JDBC or ODBC. Through we have these protocols to access data in database we use specialized protocol for accessing directory information because of following reasons.

1. Directory access protocols are simplified and modified to a limited type of access to data.

   They can be implemented with database access protocols.

2. It provide a simple mechanism for giving name objects is a hierarchical fashion, same as file system directory names.

DAP protocol is used in a distributed directory system to specify what information is stored is each to the directory servers.

For example one particular directory server may store information for particular company's employee in one city and other store information of same company's employee in another city.

Several directory systems use relational database to store data, instead of creating special-purpose storage systems.

2.11.2 LDAP : Light Weight Directory Access Protocol

- In directory system implementation one or more servers are used which provide service to multiple clients.
- Clients use the application program interface defined by directory system to communicate with directory servers.
- Directory access protocols defines data models and access control for directory system.
- X.500 is a standard for accessing directory information (defined by ISO). But it is a complex protocol and not used widely.
- The LDAP provides the many of the features of X.500 with less complexity, so this protocol is widely used.

2.11.2.1 LDAP Data Model

It stores entries, which are similar to objects in its directories.

Each entry must have **Distinguished Name (DN)** which uniquely identifies the entry.

A distinguished name is made up of a sequence of **relative distinguished names (RDNs)**.

A distinguished name is made up of a sequence of **relative distinguished names (RDNs).**

For example, an entry may have following distinguished name.

Ename = person name, organization unit, Organization country

The distinguished name is a combination of name and address, starting with person's name, then giving organizational unit, the organization and country.

- The set of RDNs for DN is defined by the schema of the directory system.
- Entries can also have attributes. LDAP provides binary, string, time types for attributes with tel, postaladdress type for specifying telephone number and address.
- Attributes are multivalued by default, which make possible storing of multiple phone numbers or address against an entry.
- LDAP allows the definition of object classes with attribute names and types. Inheritance can be used in defining object classes.
- Entries are organized into a directory information tree (DIT) according to their distinguished names.
- The entire distinguished name need not be stored in an entry. The system can generate the distinguished name by traversing up the DIT from the entry. Collecting the RDN value of components to make the full distinguished name.

2.11.2.2 Data Manipulation

LDAP defines a network protocol for carrying out data definition and manipulation.

LDAP also defines a file format called LDAP Data Interchange Format (LDIF) which is used for storing and exchanging information.

The querying mechanism is very simple in LDAP which consist only selections and projections without any join.

A LDAP query must contain following .

1) A base : A node within DIT, by giving distinguished name.
2) A search condition: Combination of boolean condition on individual attributes.

   e.g. Equality, Approximate equality
3) A scope : It can be just the base or the base and its children or the entire subtree of base.
4) Attributes : Name of attributes which is to be returned.
5) Limits on number of results and resources consumption.

- In the query we have to specify whether to automatically dereferences aliases.

2.11.3 Distributed Directory Trees

Information about an organization may be split into multiple DITs.

- The sequence of RDN pair is the **suffix** for DIT which identify what information the DIT stores.
- The DITs may be organizationally and geographically separated.
- A node in a DIT may contain a referral to another node in another DIT.
- Referrals are the key component which allow integration of distributed directories.
- After getting a query on a DIT server return the referral to the client, which then issues a query on that DIT.
- Alternative technique is server itself issue query to referred DIT and return the result.
- Many LDAP implementations support master-slave and multimaster replication of DITs.

UNIT - II

Parallel Databases

A parallel database system seeks to improve performance through parallellization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion such a system, the distribution is governed solely by performance considerations.

1.2 Parallel Systems :

- ✧ Parallel systems improve processing and I/O speeds by using multiple CPUs and disks in parallel. Parallel machines are becoming increasingly common, making the study of parallel database systems correspondingly more important.
- ✧ The driving force behind parallel database systems is the demands of applications that have to query an extremely large databases or that have to process large number of transactions per second (of the order of thousands of transactions per second).
- ✧ Centralized and client-server database systems are not powerful enough to handle such applications.
- ✧ In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.
- ✧ A coarse-grain parallel machine consists of the small number of powerful processors; a massively parallel or fine-grain parallel machine uses thousands of smaller processors.
- ✧ Most high-end machines today offer some degree of coarse-grain parallelism: Two or more processor machines are common.

1.2.1 Measures of Performance of Database Systems :

There are two main measures of performance of a database system :

1. Throughout, the number of tasks that can be completed in a given time interval.
2. Response time, the amount of time it takes to complete a single task from the time it is submitted. A system that processes a large number of small transactions can improve throughout by

processing many transactions in parallel. A system that processes large transactions can improve response time as well as throughout by performing subtasks of each transaction in parallel.

1.2.2 Speedup and Scaleup :

Two important issues in studying parallelism are :

(1) **Speedup :**

Running a given task in less time by increasing the degree of parallelism is called speedup.

(2) **Scaleup :**

Handling larger tasks by increasing the degree of parallelism is scaleup.

Speedup :

- ✧ Consider a database application running on a parallel system with a certain number of processors and disks. Now suppose that we increase the size of the system by increasing the number or processors, disks, and other components of the system.
- ✧ The goal is to process the task in time inversely proportional to the number of processors and disks allocated.
- ✧ The parallel system is said to demonstrate linear speedup if the speedup is N when the larger system has N times the resources (CPU, disk, and so on) of the smaller system.
- ✧ If the speedup is less than N, the system is said to demonstrate sublinear speedup Fig. 1.1 illustrates linear and sublinear speedup.

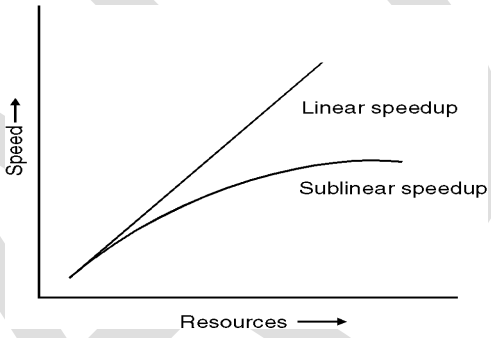


Fig. 1.1 : Speedup

Scaleup :

- ✧ Scaleup relates to the ability to process larger tasks in the same amount of time by providing more resources.
- ✧ Let Q be a task and Q_N be a task that is N times bigger than Q. Suppose execution time of task Q on machine M_S is T_S and the execution time of task Q_N on parallel machine M_L which is N times larger than M_S is T_L.

  Scaleup is defined as T_S / T_L.

  Where,

  T_L : Execution time of a task on the larger machine

  T_S : The execution, time of the same task on the smaller machine

  The parallel system M_L is said to demonstrate linear scaleup on task Q if.

  $T_L = T_S$.

If $T_L > T_S$ the system is said to demonstrate sublinear scaleup.

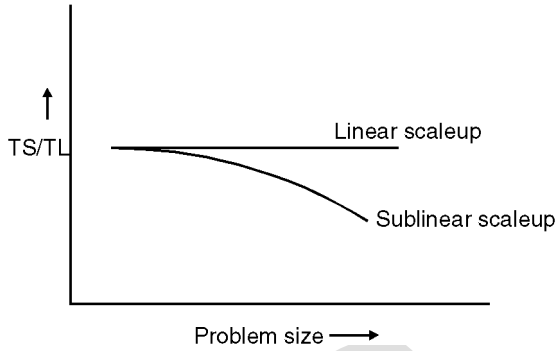


Fig. 1.2 : Scaleup

1.3 Architectures for Parallel Databases :

There are several architectural models for parallel machines. Among the most prominent ones are those in Fig. 1.3 (In the Fig. 1.3, M denotes memory, P denotes a processor, and disks are shown as cylinders)

✧ **Shared memory** : All the processors share a common memory (Fig. 1.3(a)).

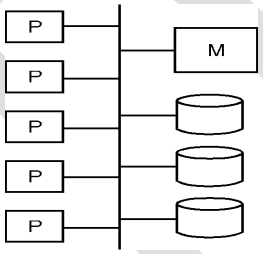


Fig. 1.3(a) : Shared memory

- **Shared disk :** All the processors share a common set of disk (Fig. 1.3(b)). Shared-disk are sometimes called clusters.

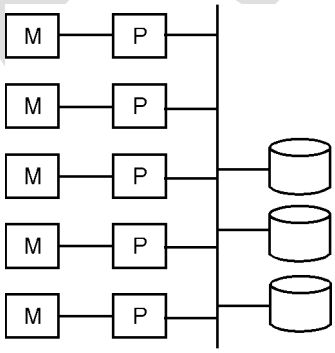


Fig. 1.3(b) : Shared disk

- **Shared nothing :** The processors share neither a common memory nor common disk (Fig. 1.3(c)).

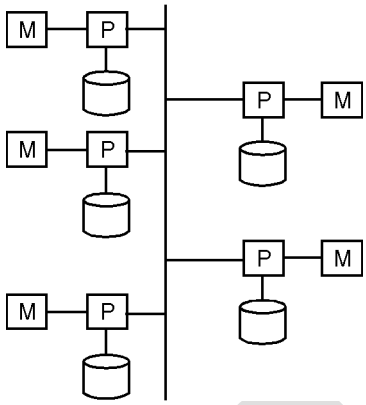


Fig. 1.3(c) : Shared nothing

- **Hierarchical :** This model is a hybrid of the preceding three architectures (Fig. 1.3(d)).

Techniques used to speedup transaction processing on data-server systems, such and lock caching and lock de-escalation, can also be in shared-disk parallel databases as well as in shared-nothing parallel databases. In fact, they are very important for efficient transaction processing in such systems.

1.3.1 Shared Memory :

In shared memory architecture, the processors and disks have access to a common memory, typically via a bus or through an interconnection network.

Advantages :

- ✧ The benefit shared memory is extremely efficient communication between processors. Data in shared memory can be accessed by any processor without being moved with software.
- ✧ A processor can send messages to other processors much faster by using memory writes (which usually rake less than a microsecond) than by sending a message through communication mechanism.

Disadvantages :

- ✧ The downside of shared-memory that the architecture is not scalable beyond 32 or 64 processors because the bus or interconnection network becomes a bottleneck (since it is shared by all processors).
- ✧ Adding more number of processors should be avoided as they most of the time in waiting for their turn on the bus to access memory.
- ✧ Shared-memory architectures usually have large memory caches at each processor so that referencing of the shared memory is avoided whenever possible.

- ✧ However, at least some of the data will not be in the cache and accesses will have to go to the shared memory. Moreover, the caches need to be kept coherent.
- ✧ Maintaining cache-coherency becomes an increasing overhead with increasing overhead with increasing number of processors.
- ✧ Consequently, shared memory machines are not capable of scaling up beyond a point; current shared-memory machines cannot support more than 64 processors.

1.3.2 Shared Disk :

In the shared-disk model, all processors can access all disks directly via an interconnection network, but the processors have private memories.

Advantages :

- ✧ Since each processor has its own memory, the memory bus is not a bottleneck.
- ✧ It offers a cheap way to provide a degree of fault tolerance.
- ✧ If a processor (or its memory) fails, the other processor can take over its tasks, since the database is resident on disks that are accessible from all processors.
- ✧ We can make the disk subsystem itself fault tolerant by using RAID architecture,
- ✧ The shared-disk architecture has found acceptance in many applications.

Disadvantages :

- ✧ The main problem with a shared-disk system is again scalability.
- ✧ Although the memory bus is no longer a bottleneck, the interconnection to the disk subsystem is now a bottleneck; it is particularly so in a situation where the database makes a large number of accesses to disks.
- ✧ Compared to shared memory systems, shared-disk systems can scale to a somewhat larger number of processors, but communication across processor is slower, since it has to go through a communication network.

Example :

DEC clusters running Rdb were One of the early commercial users of the shared disk database architecture. (Rdb is now owned by Oracle, and is tailed Oracle Rdb. Digital Equipment Corporation (DEC) is now owned by Compaq.)

1.3.3 Shared Nothing :

- ✧ In a shared-nothing system, each node of the machine consists of a processor, memory, and one or more disks.
- ✧ The processors at one node may communicate with one another processor at another node by a high-speed interconnection network.

- ✧ A node functions as the server for the data on the disk or disks that the node owns. Since local disk references are serviced by local disks at each processor.

Advantages :

- ✧ The shared-nothing model overcomes the disadvantage of requiring all I/O to go through a singly interconnection network; only queries, accesses to non local disks, and result relations pass through the network.
- ✧ Moreover, the interconnection networks for shared nothing systems are usually designed to be scalable, so that their transmission capacity increases as more nodes are added.
- ✧ Consequently, shared-nothing architectures are more scalable, and can easily support a large number of processors.

Disadvantage :

The main drawback of shared nothing systems is the costs of communication and of nonlocal disk access, which are higher than in a shared memory or shared-disk architecture since sending data involves software interaction at both ends.

Applications :

- ✧ The Teradata database machine was among (the earliest commercial systems to use the shared-nothing database architecture.
- ✧ The Grace and the Gamma research prototypes also used shared-nothing architectures.

1.3.4 Hierarchical :

- ✧ The hierarchical architecture combines the characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- ✧ At the top level, the system consists of nodes connected by an interconnection network, and do not share disks or memory with one another. Thus, the top level is a shared-nothing architecture.
- ✧ Each node of the system could actually be a shared-memory system with a few processors Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
- ✧ Thus, a system could be built as a hierarchy, with shared-memory architecture with a few processors at the base, and a shared-nothing architecture at the top, with possibly shared-disk architecture in the middle.
- ✧ Fig. 1.3(d) illustrates a hierarchical architecture with shared-memory nodes connected together in a shared nothing architecture.
- ✧ Commercial parallel database systems today run on several of these architectures.
- ✧ Attempts to reduce the complexity of programming such systems have yielded distributed virtual memory architectures, where logically there is a single shared memory, but physically there are multiple disjoint memory systems; the virtual-memory-mapping hardware, coupled with system software, allows each processor *to* view the disjoint memories as a single virtual memory.

- ✧ Since access speeds differ, depending whether the page is available locally or not, such architecture is also referred to as nonuniform memory architecture (NUMA).

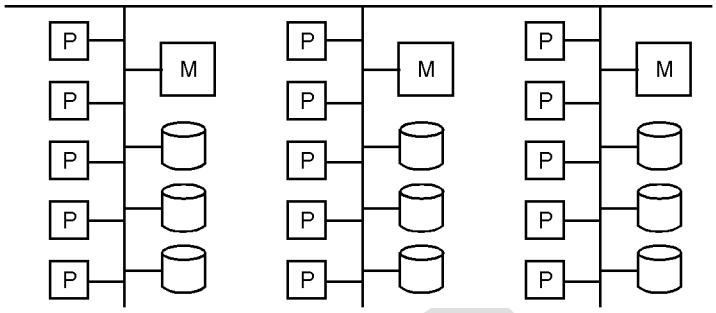


Fig. 1.3(d)

1.3.5 Parallel Query Evaluation :

Now we try to understand parallel evaluation of a relational query in a DBMS with a shared-nothing architecture. While it is possible to consider parallel execution of multiple queries, it is hard to identify in advance which queries will run concurrently. So the emphasis has been on parallel execution of a single query.

- ✧ A relational query execution plan is a graph of relational algebra operators, and the operators in a graph can be executed in parallel. If one operator consumes the output of a second operator, we have **pipelined parallelism.**(the output of the second operator is worked on by the first operator as soon as it is generated)
- ✧ If not, the two operators can proceed essentially independently. An operator is said to **block** if it produces no output until it has consumed all inputs. Pipelined parallelism is limited by the presence of operators that block.
- ✧ To evaluate different operators in parallel, we can evaluate each individual operator in a query plan in a parallel fashion. The key to evaluating operator in parallel is to *partition* the input data; we can then work on a partition in parallel and combine the results. This approach is called **a partitioned parallel evaluation.**
- ✧ An important observation, which explains why shared-nothing parallel database system have been very successful, is that database query evaluation is very amenable to data-partitioned parallel evaluation.
- ✧ The goal is to minimize data shipping by partitioning the data and structuring the algorithms to do most of the processing at individual processors.

1.4 I/O Parallelism :

Definition : I/O parallelism refers to reducing the time required to retrieve relations from disk by partitioning the relations on multiple disks. The most common form of data partitioning in a parallel database environment is *horizontal partitioning.*

In **horizontal** partitioning, the tuples of a relation are divided (or declustered) among many disks, so that each tuple resides on one disk. Several partitioning strategies have been proposed.

1.4.1 Partitioning Techniques :

There are three basic data-partitioning strategies. Assume that there are *n* disks D_O, D_I. ..., D_{n-1} across which the data are to be partitioned.

- **Round-robin :** This strategy scans the relation in any order and sends the i^{th} tuple to disk number D_i mod n. The round-robin scheme ensures an even distribution of tuples across disks; that is each disk has approximately the same number of tuples as the others.
- **Hash partitioning :** This declustering strategy designates one or more attributes from the given relation's schema as the partitioning attributes. A hash function is chosen whose range is {0, 1,-----, n – 1}. Each tuple of the original relation is hashed on the partitioning attributes. If the hash function returns i, then the tuple is placed on disk D_i.
- **Range partitioning :** This strategy distributes contiguous attribute value ranges to each disk. It chooses a partitioning attribute. **partitioning vector.** The relation is partitioned as follows. Let $\{V_0,V_1,...,V_{n-2}\}$ denote the partitioning vector, such that, if i < j, then $V_i < V_j$. Consider a tuple t such that t[A] = x. If $x < V_0$ then t goes on disk D_o.
- If $x >= V_{n-2}$, then t goes on disk D_{n-1}. If $V_i <= x < V_{i+1}$ then t goes on disk D_{i+1}.

For example, range partitioning with three disks numbered 0, 1, and 2 may assign tuples with values less than 5 to disk 0, values between 5 and 40 to disk 1, and values greater than 40 to disk 2.

1.4.2 Comparison of Partitioning Techniques :

- Once a relation has been partitioned among several disks, we can retrieve it in parallel, using all the disks.
- Similarly, when a relation is being partitioned, it can be written to multiple disks in parallel.
- Thus, the transfer rates for reading or writing an entire relation are much faster with I/O parallelism than without it.
- However, reading an entire relation is only one kind of access to data. Access to data can be classified as follows :
  1. Scanning the entire relation.
  2. Locating a tuple associatively (for example, *employee-name* - "Campbell"); these queries, called point queries, seek tuples that have a specified value for a specific attribute.
  3. Locating all tuples for which the value of a given attribute lies within a specified range (for example, 10000 < *salary* < 20000); these queries are called range queries.

  The different partitioning techniques support these types of access at different levels of efficiency :

1.4.2.1 Round-robin :

The scheme is ideally suited for applications that wish to read the entire relation sequentially for each query. With this scheme, both point queries and range queries are complicated to process, since each of the *n* disks must be used for the search.

1.4.2.2 Hash Partitioning :

- This scheme is best suited for point queries based on the partitioning attribute.

- For example if a relation is partitioned on the *telephone-number* attribute, then we can answer the query "Find the record of the employee with *telephone-number* = 555-3333" by applying the partitioning hash function to 555-3333 and then searching that disk. Directing a query to a single disk saves the startup cost of initiating a query on multiple 4 disks and leaves the other disks free to process other queries.
- Hash partitioning is also useful for sequential scans of the entire relation.
- If the hash function is a good randomizing function, and the partitioning attributes form a key of the relation, then the number of tuples in each of the disks is approximately the same, without much variance.
- Hence, the time taken to scan the relation is approximately 1/n of the time required to scan the relation in a single disk system.
- The scheme, however, is not well suited for point queries non partitioning attributes.
- Hash-based partitioning is also not well suited for answering range queries, since, typically, hash functions do not preserve proximity within a range. Therefore, all the disks need to be scanned for range queries to be answered.

1.4.2.3 Range Partitioning :

- This scheme is well suited for point and range queries on the partitioning attribute. For point queries, we can consult the partitioning vector to locate the disk where the tuple resides.
- For range queries, we consult the partitioning vector to find the range of disks on which the tuples may reside. In both cases, the search narrows to exactly those disks that might have any tuples of interest.
- An advantage of this feature is that, if there are only a few tuples in the queried range, then the query is typically sent to one disk, as opposed to all the disks.
- Since other disks can be used to answer other queries, range partitioning results in higher throughput while maintaining good response time. On the other hand, if there are many tuples in the queried range (as there are when the queried range is a larger fraction of the domain of the relation), many tuples have to be retrieved from a few disks, resulting in an I/O bottleneck (hot spot) at those disks.
- In this example of execution skew, all processing occurs in one or only a few partitions. In contrast, hash partitioning and round-robin partitioning would engage all the disks for such queries, giving a faster response time for approximately the same throughput.
- The type of partitioning also affects other relational operations, such as joins. Thus, the choice of partitioning technique also depends on the operations that need to be executed. In general, hash partitioning or range partitioning are preferred to round-robin partitioning.
- In a system with many disks, the number of disks across which to partition a relation can be chosen in this way; if a relation contains only a few tuples that will fit into a single disk block, then it is better to assign the relation to a single disk.

- ✧ Large-relations are preferably partitioned across all the available disks. If a relation consists of *m* disk blocks and there are n disks available in the system, then the relation should be allocated min(m, n) disks.

1.4.3 Handling of Skew :

When a relation is partitioned (by a technique other than round-robin), there may be a skew in the distribution of tuples, with a high percentage of tuples placed in some partitions and fewer tuples in other partitions. The ways that skew may appear are classified as:

1. Attribute-value skew
2. Partition skew

1.4.3.1 Attribute-value Skew :

- ✧ It refers to the fact that some values appear in the partitioning attributes of many tuples. All the tuples with the same value for the partitioning attribute end up in the same partition, resulting in skew. Partition skew refers to the fact that there may be load imbalance in the partitioning, even when there is no attribute skew.
- ✧ Attribute-value skew can result in skewed partitioning regardless of whether range partitioning or hash partitioning is used. If the partition vector is not chosen carefully, range partitioning may result in partition skew. Partition skew is less likely with hash partitioning, if a good hash function is chosen.
- ✧ Skew becomes an increasing problem with a higher degree of parallelism.
- ✧ For example, if a relation of 1000 tuples is divided into 10 parts, and the division is skewed, then there may be some partitions of size less than 100 and some partitions of size more than 100
- ✧ If even one partition happens to be of size 200, the speedup that we would obtain by accessing the partitions in parallel is only 5, instead of the 10 for which we would have hoped.
- ✧ If the same relation has to be partitioned into 100 parts, a partition will have 10 tuples on an average. If even one partition has 40 tuples (which is possible, given the large number of partitions) the speedup that we would obtain by accessing them in parallel would be 25, rather than 100. Thus the loss of speedup due to skew increases with parallelism.

1.4.3.2 A Balanced Range-Partitioning Skew :

- ✧ A balanced range partitioning vector can be constructed by sorting the relation. The relation is first sorted on the partitioning attributes.
- ✧ The relation is then scanned in sorted order. After every 1/n of the relation has been read the value of the partitioning attribute of the next tuple is added to the partition vector.
- ✧ Here, n denotes the number of partitions to be constructed. In case there are many tuples with the same value for the partitioning attribute, the technique can still result in some skew.

Disadvantage :

The extra I/O overhead incurred in doing the initial sort.

How I/O overhead is avoided ?

- ✧ The I/O overhead for constructing balanced range-partition vectors can be reduced by constructing and storing a frequency table, or histogram, of the attribute values for each attribute of each relation. (See Fig. 1.4).
- ✧ Histogram for an integer valued attribute that takes values in the range 1 to 25.
- ✧ A histogram takes up only a little space, so histograms on several different attributes can be stored in the system catalog.

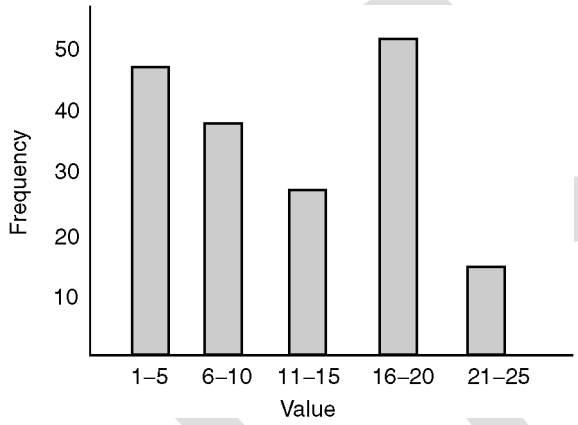


Fig. 1.4 : Example of histogram

- ✧ It is straightforward to construct a balanced range-partitioning function given a histogram on the partitioning attributes.
- ✧ If the histogram is not stored, it can be computed approximately by sampling the relation, using only tuples from a randomly chosen subset of the disk blocks of the relation.

Another approach to minimize the effect of skew :

- ✧ Another approach to minimize the effect of skew particularly with range partitioning is to use *virtual processors.*
- ✧ The virtual processor approach, we pretend there are several times as *many virtual processors* as the number of real processors.
- ✧ Any of the partitioning techniques and query evaluation techniques that we study later in this chapter can be used, but they map tuples and work to virtual processors instead of to real processors.
- ✧ Virtual processors, in turn, are mapped to real processors, usually by round-robin partitioning.
- ✧ The idea is that even if one range had many more tuples than the others because of skew, these tuples would get split across multiple virtual processor ranges.
- ✧ Round robin allocation of virtual processors to real processors would distribute the extra work among multiple real processors, so that one processor does not have to bear all the burden.

1.5 Parallelizing Individual Operations :

- ✧ It shows how various operations can be implemented in parallel in a shared-nothing architecture.
- ✧ We assume that each relation is horizontally partitioned across several disks, although this partitioning may or may not be appropriate for a given query.
- ✧ The evaluation of a query must take the initial partitioning criteria into account and repartition if necessary. We use following techniques :
  1. Bulk loading and scanning
  2. Sorting

1.5.1 Bulk Loading and Scanning :

- ✧ We begin with two simple operations: *scanning* a relation and *loading* a relation.
- ✧ Pages can be read in parallel while scanning a relation, and the retrieved tuples can then be merged, if the relation is partitioned across several disks.
- ✧ More generally, the idea also applies when retrieving all tuples that meet a selection condition. If hashing or range partitioning is used, selection queries can be answered by going to just those processors that contain relevant tuples.
- ✧ Similar observation holds for bulk loading. Further, if a relation has assorted indexes, any sorting of data entries required for building the indexes, during bulk loading can also be done in parallel.

1.5.2 Sorting :

- ✧ A simple idea is to let each CPU sort the part of the relation that is on its local disk and then merge these sorted sets of tuples. The degree of parallelism likely to be limited by the merging phase.
- ✧ A better idea is to first redistribute all tuples in the relation using range partitioning.
- ✧ For example, if we want to sort a collection of employee tuples by salary, salary values range from 10 to 210, and we have 20 processors, we could send all tuples with salary values in the range 10 to 20 to the first processor all in the range 21 to 30 to the second processor, and so on.
- ✧ Each processor then sorts the tuples assigned to it, using some sequential sorting algorithm.
- ✧ For example, a processor can collect tuples until its memory full then sort these tuples and write out a run, until all incoming tuples has been written to such sorted runs on the local disk.
- ✧ These runs can then be to create the sorted version of the set of tuples assigned to this processor entire sorted relation can be retrieved by visiting the processors in an order corresponding to the ranges assigned to them and simply scanning the tuples.

1.5.2.1 Splitting Vector :

- ✧ The basic challenge in parallel sorting is to do the range partitioning each processor receives roughly the same number of tuples; otherwise, a processor that receives a disproportionately large number of tuples to sort becomes a bottleneck and limits the scalability of the parallel sort.
- ✧ One good approach to range partitioning is to obtain a sample of the entire relation by taking at each processor that initially contains part of the relation.

- ✧ The (relatively small) sample is sorted and used to identify ranges with equal number of tuples. This set of range values, called a **splitting vector,** is then distributed all processors and used to range partition the entire relation.

1.5.2.2 Application of Sorting :

A particularly important application of parallel sorting is sorting the data entries (in tree-structured indexes. Sorting data entries can significantly speed up the process of bulk-loading an index.

1.6 Interquery Parallelism :

Definition :

- ✧ In interquery parallelism, different queries or transactions execute in parallel with one another.
- ✧ Transaction throughput can be increased by this form of parallelism.
- ✧ However, the response times of individual transactions are no faster than they would be if the transactions were run in isolation.

1.6.1 Working of Interquery Parallelism :

- ✧ The primary use of interquery parallelism is to scaleup a transaction-processing system to support a larger number of transactions per second.
- ✧ Interquery parallelism is the easiest form of parallelism to support in a database system particularly in a shared-memory parallel system.
- ✧ Database systems designed for single-processor systems can be used with few or no changes on a shared-memory parallel architecture
- ✧ Since even sequential database systems support concurrent processing, transactions that would have operated in a time-shared concurrent manner on a sequential machine operate in parallel in the shared-memory parallel architecture
- ✧ Supporting interquery parallelism is more complicated in shared disk or shared nothing architecture.
- ✧ Processors have to perform some tasks, such as locking and logging, in a coordinated fashion, and that requires that they pass messages to each other.
- ✧ A parallel database system must also ensure that two processors do not update the same data independently at the same time.
- ✧ Further, when a processor accesses or updates data, the database system must ensure that the processor has the latest version of the data in its buffer pool. The problem of ensuring that the version is the latest is known as the cache-coherency problem.

1.6.2 Protocols used in Shared Disk System :

Various protocols are available to guarantee cache coherency; often, cache-coherency protocols are integrated with concurrency-control protocols so that their overhead is reduced. One such protocol for a shared-disk system is this :

1. Before any read or write access to a page, a transaction locks the page in shared or exclusive mode, as appropriate. Immediately after the transaction obtains either a shared or exclusive lock on a page, it also reads the most recent copy of the page from the shared disk.
2. Before a transaction releases an exclusive lock on a page, it flushes the page to the shared disk; then, it releases the lock. This protocol ensures that, when a transaction sets a shared or exclusive lock on a page, it gets the correct copy of the page.

1.6.3 Advantages of Complex Protocols :

- ✧ More complex protocols avoid the repeated reading and writing to disk required by the preceding protocol. Such protocols do not write pages to disk when exclusive locks are released.
- ✧ When a shared or exclusive lock is obtained if the most recent version of a page is in the buffer pool of some processor, the page is obtained from there.
- ✧ The protocols have to be designed to handle concurrent requests.
- ✧ The shared-disk protocols can be extended to shared-nothing architectures by this scheme.
- ✧ Each page has a home processor Pi and is stored on disk D_i.
- ✧ When other processors want to read or write the page, they send requests to the home processor Pi of the page, since they cannot directly communicate with the disk. The other actions are the same as in the shared-disk protocols.
- ✧ The Oracle 8 and Oracle Rdb systems are examples of shared-disk parallel database systems that support interquery parallelism.

1.7 Intraquery Parallelism :

Definition : Intraquery parallelism refers to the execution of a single query in parallel on multiple processors and disks. Using intraquery parallelism is important for speeding up long running queries. Interquery parallelism does not support in this task since each query is run sequentially.

1.7.1 Working of Intraquery Parallelism :

- ✧ To illustrate the parallel evaluation of a query, consider a query that requires a relation to be sorted. Suppose that the relation has been partitioned across multiple disks by range partitioning on some attribute, and the sort is requested on the partitioning attribute.
- ✧ The sort operation can be implemented by sorting each partition in parallel, then concatenating the sorted partitions to get the final sorted relation.
- ✧ Thus, we can parallelize a query by parallelizing individual operations. There is another source of parallelism in evaluating a query: The *operator tree* for a query can contain multiple operations.

- ✧ We can parallelize the evaluation of the operator tree by evaluating in parallel some of the operations that do not depend on one another. We may be able to pipeline the output of one operation to another operation.
- ✧ The two operations can be executed in parallel on separate processors, one generating output that is consumed by the other, even as it is generated.

In summary, the execution of a single query can be parallelized in two ways :

1. **Intraoperation parallelism :** We can speed up processing of a query by parallelizing the execution of each individual operation, such as sort, select, project, and join.
2. **Interoperation parallelism :** We can speed up processing of a query by executing in parallel the different operations in a query expression.

1.7.2 Importance of Parallelism :

- ✧ The two forms of parallelism are complementary and can be used simultaneously on a query. Since the number of operations in a typical query is small, compared to the number of tuples processed by each operation, the first form of parallelism can scale better with increasing parallelism. However, with the relatively small number of processors in typical parallel systems today, both forms of parallelism are important.
- ✧ In the following discussion of parallelization of queries, we assume that the queries are **read only.**
- ✧ The choice of algorithms for parallelizing query evaluation depends on the machine architecture. Rather than presenting algorithms for each architecture separately.
- ✧ We use a shared-nothing architecture model in our description. Thus, we explicitly describe when data have to be transferred from one processor to another.
- ✧ We can simulate this model easily by using the other architectures, since transfer of data can be done via shared memory in a shared-memory architecture, and via shared disks in a shared-disk architecture.
- ✧ Hence, algorithms for shared-nothing architectures can be used on the other architectures too. We mention occasionally how the algorithms can be further optimized for shared-memory or shared-disk systems.
- ✧ To simplify the presentation of the algorithms, assume that there are *n* processors, P_o, P_i, … , P_{n-1}, and *n* disks D_o, D_i, ... ,D_{n-1}, where disk D_i is associated with processor P_i. A real system may have multiple disks per processor.
- ✧ It is not hard to extend the algorithms to allow multiple disks per processor. We simply allow D_i to be a set of disks. However, for simplicity, we assume here that A is a single disk

1.8 Intraoperation Parallelism :

- ✧ Since relational operations work on relations containing large sets of tuples, we can parallelize the operations by executing them in parallel on different subsets of the relations.

- ✧ Since the number of tuples in a relation can be large, the degree of parallelism is potentially enormous.
- ✧ Thus, intraoperation parallelism is natural in a database system.

1.8.1 Parallel Sort :

- ✧ Suppose that we wish to sort a relation that resides on parallel disks D_O, D_1 .... D_{n-1} .
- ✧ If the relation has been range partitioned on the attributes on which it is to be sorted, then, we can sort each partition separately, and can concatenate the results to get the full sorted relation.
- ✧ Since the tuples are partitioned on *n* disks, the time required for reading the entire relation is reduced by the parallel access.
- ✧ If the relation has been partitioned in any other way, we can sort it in one of two ways :
  1. We can range partition it on the sort attributes, and then sort each partition separately.
  2. We can use a parallel version of the external sort-merge algorithm.

1.8.1.1 Range-Partitioning Sort :

- ✧ Range-partitioning sort works in two steps first range partitioning the relation, then sorting each partition separately.
- ✧ When we sort by range partitioning the relation, it is not necessary to range-partition the relation on the same set of processors or disks as those on which that relation is stored. Suppose that we choose processors P_o , P_I, ... P_m, where m < *n* to sort the relation.
- ✧ There are two steps involved in this operation:
  1. Redistribute the tuples in the relation, using a range-partition strategy, so that all tuples that lie within the i^{th} range are sent to processor P_I , which stores the relation temporarily on disk D_i.

     To implement range partitioning, in parallel every processor reads the tuples from its disk and sends the tuples to their destination processor. Each processor P_O, P_I ,..., P_m also receives tuples belonging to its partition, and stores them locally. This step requires disk I/O and communication overhead.
  2. Each of the processors sorts its partition of the relation locally, without interaction with the other processors. Each processor executes the same operation namely, sorting on a different data set. (Execution of the same operation in parallel on different sets of data is called data parallelism.)

The final merge operation is trivial, because the range partitioning in the first phase ensures that, for $1 < i < j <$ m, the key values in processor P_i are all less than the key values in P_j.

We must do range partitioning with a good range-partition vector, so that each partition will have approximately the same number of tuples. Virtual processor partitioning can also be used to reduce skew.

1.8.1.2 Parallel External Sort Merge :

Parallel external sort-merge is an alternative to range partitioning. Suppose that a relation has already been partitioned among disks D_O , D_I ,..., D_{n-1} (it does not matter how the relation has been partitioned). Parallel external sort-merge then works this way :

1. Each processor P_i locally sorts the data on disk D_i.

2. The system then merges the sorted runs on each processor to get the final sorted output.

The merging of the sorted runs in step 2 can be parallelized by this sequence of actions :

1. The system range-partitions the sorted partitions at each processor P_i (all by the same partition vector) across the processors P_O , P_I .... P_{m-1}. It sends the tuples in sorted order, so that each processor receives the tuples in sorted streams.
2. Each processor P_i performs a merge on the streams as they are received, to get a single sorted run.
3. The system concatenates the sorted runs on processors P_O, P_I ...., P_{m-1} to get the final result.

- ✧ As described, this sequence of actions results in an interesting form of execution skew, since at first every processor sends all blocks of partition 0 to P_0, then every processor sends all blocks of partition 1 to P_1, and so on.
- ✧ Thus, while sending happens in parallel, receiving tuples becomes sequential: first only P_O receives tuples, then only P_I receives tuples, and so on.
- ✧ To avoid this problem, each processor repeatedly sends a block of data to each partition. In other words, each processor sends first block of every partition, then sends the second block of every partition so on. As a result, all processors receive data in parallel.
- ✧ Some machines, such as the Teradata DBC series machines, use specialized hardware to perform merging.
- ✧ The Y-net interconnection network in the Teradata DBC machines can merge output from multiple processors to give a single sorted output.

1.8.2 Parallel Join :

- ✧ The join operation requires that the system test pairs of tuples to see whether they satisfy the join condition; if they do, the system adds the pair to the join output. Parallel join algorithms attempt to split the pairs to be tested over several processors
- ✧ Each processor then computes part of the join locally.
- ✧ Then, the system collects the results from each processor to produce the final result.

1.8.2.1 Partitioned Join :

- ✧ For certain kinds of joins, such as equijoins and natural joins, it is possible to *partition* the two input relations across the processors, and to compute the join locally at each processor. Suppose that we are using *n* processors, and that the relations to be joined are *r* and ,s. Partitioned join then works this way:
- ✧ The system partitions the relations *r* and *s* each into *n* partitions, denoted r_0 , r_1 ... r_{n-1} and s_0, s_1 ... s_{n-1}. The system sends partitions r_1 and s_i to processor p_i, where their join is computed locally.
- ✧ The partitioned join technique works correctly only if the join is in an equijoin and if we partition r and s by the same partitioning function on their join attributes.
- ✧ The idea of partitioning is exactly the same as that behind the partitioning step of hash-join.
- ✧ In a partitioned join, however, there are two different ways of partitioning *r* and *s :*
    1. Range partitioning on the join attributes

2. Hash partitioning on the join attributes

- ✧ In either case, the same partitioning function must be used for both relations. For range partitioning, the same partition vector must be used for both relations.
- ✧ For hash partitioning, the same hash function must be used on both relations. Fig. 1.5 depicts the partitioning in a partitioned parallel join.
- ✧ Once the relations are partitioned, we can use any join technique locally at each processor P_i to compute the join of r_i and s_i.
- ✧ For example, hash-join, merge-join, or nested-loop join could be used. Thus, we can use partitioning to parallelize any join technique

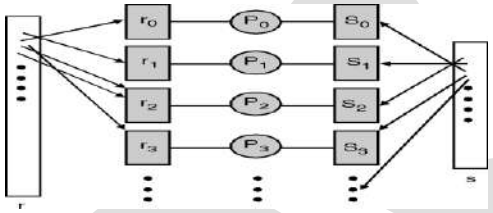


Fig. 1.5 : Partitioned parallel join

- ✧ If one or both of the relations r and s are already partitioned on the join attributes(by either hash partitioning or range partitioning), the work needed or partitioning is reduced greatly. If the relations are not partitioned, or are partitioned on attributes other than the join attributes, then the tuples need to be partitioned.
- ✧ Each processor Pi reads in the tuples on disk D_i, computes for each tuple *t* the partition *j* to which t belongs, and sends tuple t to processor P_j. Processor P_j stores the tuples on disk D_j.
- ✧ We can optimize the join algorithm used locally at each processor to reduce I/O by buffering some of the tuples to memory, instead of writing them to disk.
- ✧ Skew presents a special problem when range partitioning is used, since a partition vector that splits one relation of the join into equal-sized partitions may split the other relations into partitions of widely varying size.
- ✧ The partition vector should be such that $| r_i | + | s_i |$ (that is, the sum of the sizes of r_i and s_i) is roughly equal over all the i = 0,1...., n – 1. With a good hash function, hash partitioning is likely to have a smaller skew, except when there are many tuples with the same values for the join attributes.

1.8.2.2 Fragment and Replicate Join :

- ✧ Partitioning is not applicable to all types of joins. For instance, if the join condition is an inequality, such as $r \bowtie r \cdot a < s \cdot b\ s_i$ it is possible that all tuples in r join with some tuple in *s* (and vice versa).
- ✧ Thus, there may be no easy way of partitioning *r* and s so that tuples in partition r_i, join with only tuples in partition s_i.

✧ We can parallelize such joins by using a technique called fragment and replicate. We first consider a special case of fragment and replicate join as :

Asymmetric fragment and replicate join :

It works as follows :

1. The system partitions one of the relations say, r. Any partitioning technique can be used on r, including round-robin partitioning.
2. The system replicates the other relation, *s*, across all the processors.
3. Processor P_i then locally computes the join of r_i; with all of .s, using any join technique.

The asymmetric fragment-and-replicate scheme appears in Fig. 1.6(a). If r is already stored by partitioning, there is no need to partition it further in step 1. All that is required is to replicate s across all processors.

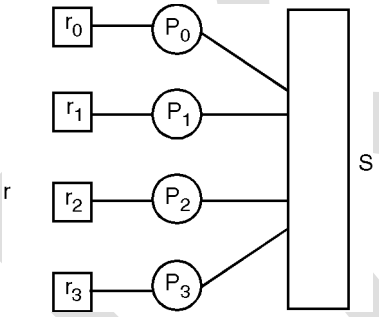


Fig. 1.6(a) : Asymmetric fragment and raplicate

Fragment and replicate join :

The general case of fragment and replicate join appears in Fig. 1.6 (b), it works as follows :

✧ The system partitions relation r into *n* partitions r_0, r_1 ... r_{n-1} and partitions *s* into *m* partitions, s_0, s_1,... s_{m-1} as before, any partitioning technique may be used on r and on s.

✧ The values of *m* and *n* do not need to be equal, but they must be chosen so that there are atleast m * n processors.

✧ Asymmetric fragment and replicate is simply a special case of general fragment and replicate, where m = 1, Fragment and replicate reduces the sizes of the relations at each processor, compared to asymmetric fragment and replicate.

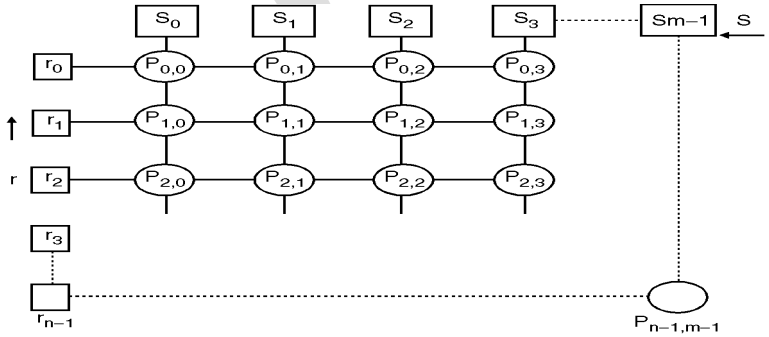


Fig. 1.6(b) : Fragment and replicate schemes

- ✧ Fragment and replicate works with any join condition, since every tuple in r can be tested with every tuple in s. Thus, it can be used where partitioning cannot be.
- ✧ Fragment and replicate usually has a higher cost than partitioning when both relations are of roughly the same size, since at least one of the relations has to be replicated.
- ✧ However, if one of the relations say, *s* is small, it may be cheaper to replicate s across all processors, rather than to repartition *r* and *s* on the join attributes. In such a case, asymmetric fragment and replicate is preferable, even though partitioning could be used.

1.8.2.3 Partitioned Parallel Hash-Join :

The partitioned hash-join of can be parallelized. Suppose that we have *n* processors, P_0, P_1 .... P_{n-1}, and two relations *r* and *s,* such that the relations r and s are partitioned across multiple disks. If the size of s is less than that of r the parallel hash-join algorithm proceeds this way :

1. Choose a hash function say, h_1 that takes the join attribute value of each tuple in *r* and .s and maps the tuple to one of the *n* processors. Let r_i denote the tuples of relation r that are mapped to processor P_i, similarly, let s_i, denote the tuples of relation s that are mapped to processor P_i. Each processor P_i reads the tuples of s that are on its disk D_i and sends each tuple to the appropriate processor on the basis of hash function h_1.
2. As the destination processor P_i receives the tuples of , s_i it further partitions them by another hash function, h_2, which the processor uses to compute the hash-join locally. The partitioning at this stage is exactly the same as in the partitioning phase of the sequential hash-join algorithm. Each processor P_i executes this step independently from the other processors.
3. Once the tuples of and have been distributed, the system redistributes the larger relation r across the m processors by the hash function h_1 in the same way as before. As it receives each tuple, the destination processor repartitions it by the function h_2 , just as the probe relation is partitioned in the sequential hash-join algorithm.
4. Each processor P_i executes the build and probe phases of the hash-join algorithm on the local partitions r_i and s_i of r and s to produce a partition of the final result of the hash-join.

The hash-join at each processor is independent of that at other processors, and receiving the tuples of r_i and s_i is similar to reading them from disk. We can use hybrid hash join to cache some of the incoming tuples in memory, Thus avoid cost of writing them and of reading them back in.

1.8.2.4 Parallel Nested-Loop Join :

- ✧ To illustrate the use of fragment and replicate-based parallelization, consider the case where the relation s is much smaller than relation r.
- ✧ Suppose that relation r is stored by partitioning; the attribute on which it is partitioned does not matter. Suppose too that there is an index on a join attribute of relation r at each of the partitions of relation r.

- ✧ We use asymmetric fragment and replicate, with relation s being replicated and with the existing partitioning of relation r.
- ✧ Each processor P_j here a partition of relation s is stored reads the tuples of relation s stored in D_j, and replicates the tuples to every other processor P_i At the end of this phase, relation s is replicated at all sites that store tuples of relation r.
- ✧ Now, each processor Pi performs an indexed nested-loop join of relation s with the i^{th} partition of relation r. We can overlap the indexed nested-loop join with the distribution of tuples of relation s, to reduce the costs of writing the tuples of relation s to disk, and of reading them back.
- ✧ However, the replication of relation s must be synchronized with the join so that there is enough space in the in memory buffers at each processor P_i to hold the tuples of relation s that have been received but that have not yet been used in the join.

1.8.3 Other Relational Operations :

The evaluation of other relational operations also can be parallelized :

Selection :

- ✧ Let the selection be a(r). Consider first the case where σ_θ is of the form $a_i = v_i$, where a_i, is an attribute and v is a value. If the relation r is partitioned on a_i, the selection proceeds at a single processor.
- ✧ If θ is of the form I <o> i < <i—that is, 9 is a range selection and the relation has been range-partitioned on a_i, then the selection proceeds at each processor whose partition overlaps with the specified range of values. In all other cases, the selection proceeds in parallel at all the processors.

Duplicate elimination :

- ✧ Duplicates can be eliminated by sorting; either of the parallel sort techniques can be used, optimized to eliminate duplicates as soon as they appear during sorting.
- ✧ We can also parallelize duplicate elimination by partitioning the tuples (by either range or hash partitioning) and eliminating duplicates locally at each processor.

Projection :

- ✧ Projection without duplicate elimination can be performed as tuples are read in from disk in parallel.
- ✧ If duplicates are to be eliminated, either of the techniques just described can be used.

Aggregation :

- ✧ Consider an aggregation operation. We can parallelize the operation by partitioning the relation on the grouping attributes, and then computing the aggregate values locally at each processor. Either hash partitioning or range partitioning can be used.
- ✧ If the relation is already partitioned on the grouping attributes, the first step can be skipped.
- ✧ We can reduce the cost of transferring tuples during partitioning by partly computing aggregate values before partitioning, at least for the commonly used aggregate functions.

- ✧ Consider an aggregation operation on a relation r, using the sum aggregate function on attribute B, with grouping on attribute *A.* The system can perform the operation at each processor P_i on those r tuples stored on disk D_i.
- ✧ This computation results in tuples with partial sums at each processor; there is one tuple at P_i for each value for attribute *A* present in r tuples stored on D_i.
- ✧ The system partitions the result of the local aggregation on the grouping attribute *A,* and performs the aggregation again (on tuples with the partial sums) at each processor P_i to get the final result.
- ✧ As a result of this optimization, fewer tuples need to be sent to other processors during partitioning.

1.9 Interoperation Parallelism :

There are two forms of interoperation parallelism :

1. Pipelined parallelism
2. Independent parallelism.

1.9.1 Pipelined Parallelism :

- ✧ Pipelining forms an important source of economy of computation for database query processing. In pipelining, the output tuples of one operation, *A,* are consumed by a second operation, *B,* even before the first operation has produced the entire set of tuples in its output
- ✧ The major advantage of pipelining execution in a sequential evaluation is that we can carry out a sequence such operations without writing any of the intermediate results to disks.
- ✧ Parallel systems use pipelining primarily for the same reason that sequential systems do.
- ✧ However, pipelines are a source of parallelism as well, in the same way that instruction pipelines are source of parallelism in hardware design.
- ✧ It is possible, to run operations A and B simultaneously on different processors, so that *B* consumes tuples in parallel with *A* producing them. This form of **parallelism is called pipelined parallelism.**

Consider a join of four relations :

$r1 \bowtie r2 \bowtie r3 \bowtie r4$

- ✧ We can set up a pipeline that allows the three joins to be computed in parallel. Suppose processor P_1 is assigned the computation of temp1 $\leftarrow r_1 \bowtie r_2$ and P_2 is assigned the computation of $r_3 \bowtie temp_1$.
- ✧ As P_1 computes tuples in $\leftarrow r_1 \bowtie r_2$, it makes these tuples available to processor P_2. Thus, P_2 has available to it some of the tuples in $r_1 \bowtie r_2$ before P_1 has finished its computation. P_2 can use those tuples that are available to begin computation of $temp_1 \bowtie r_3$, even before $r_1 \bowtie r_2$ is fully computed by P_1.
- ✧ Likewise, as P_2 computes tuples in $(r_1 \bowtie r_2) \bowtie r_3$, it makes these tuples available to P_3, which computes the join of these tuples with r_4.
- ✧ Pipelined parallelism is useful with a small number of processors, but does not scaleup as well.
- ✧ First, pipeline chains generally don't attain sufficient length to provide a high degree of parallelism.

- ✧ Second, it is not possible to pipeline relational operators that do not produce output until all inputs have been accessed, such as the set-difference operation.
- ✧ Third, only marginal speedup is obtained for the frequent cases in which one operator's execution cost is much higher than are those of the others.
- ✧ All things considered, when the degree of parallelism is high, pipelining. is less important source of parallelism than partitioning.
- ✧ The real reason for using pipelining is that pipelined executions can avoid writing intermediate results to disk.

1.9.2 Independent Parallelism :

Operations in a query expression that do not depend on one another can be executed in parallel. This form of parallelism is called **independent parallelism .**

Consider the join r1 ⋈ r2 ⋈ r3 ⋈ r4. Clearly, we can compute $temp_1 \leftarrow r_1 \bowtie r_2$ in parallel with $temp_2 \leftarrow r_3 \bowtie r_4$. When these two computations complete, we compute

$$temp_1 \bowtie temp_2$$

To obtain further parallelism, we can pipeline the tuples in $temp_1$ and $temp_2$ into the computation of $temp_1 \bowtie temp_2$, which is itself carried out by a pipelined join

Like pipelined parallelism, independent parallelism does not provide a high degree of parallelism and is less useful in a highly parallel system, although it is useful with a lower degree of parallelism.

Data partitioning :

- ✧ Partitioning a relation involves distributing its tuples over several disks. Data partitioning has its origins in centralized systems that had to partition files, either because the file was too big for one disk, or because the file access rate could not be supported by a single disk.
- ✧ Distributed databases use data partitioning when they place relation fragments at different network sites, Data partitioning allows parallel database systems to exploit the I/O bandwidth of multiple disks by reading and writing them in parallel.
- ✧ This approach provides I/O bandwidth superior to RAID-style systems without needing any specialized hardware.
- ✧ The simplest partitioning strategy distributes tuples among the fragments in a *roundrobin* fashion.
- ✧ This is the partitioned version of the classic entry-sequence file. Round robin partitioning is excellent if all applications want to access the relation by sequentially scanning all of it on each query.
- ✧ The problem with round-robin partitioning is that applications frequently want to associatively access tuples, meaning that the application wants to find all the tuples having a particular attribute value.
- ✧ The SQL query looking for the Smith's in the phone book is an example of an associative search.

- ***Hash partitioning*** is ideally suited for applications that want only sequential and associative access to the data.
- Tuples are placed by applying a *hashing* function to an attribute of each tuple. The function specifies the placement of the tuple on a particular disk.
- Associative access to the tuples with a specific attribute value can be directed to a single disk, avoiding the overhead of starting queries on multiple disks.
- Hash partitioning mechanisms are provided by Arbre, Bubba, Gamma, and Teradata.

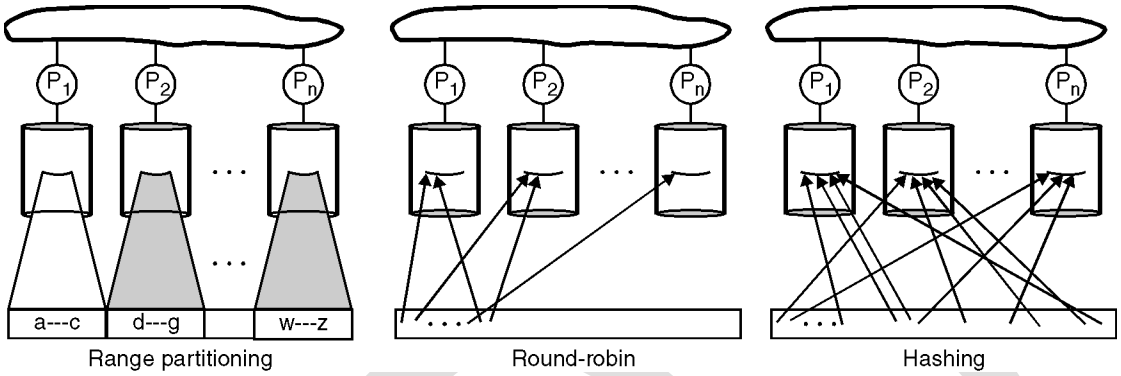


Fig. 1.11

- Database systems pay considerable attention to clustering related data together in physical storage.
- If a set of tuples are routinely accessed together, the database system attempts to store them on the same physical page.
- For example, if the Smith's of the phone book are routinely accessed in alphabetical order, then they should be stored on pages in that order, these pages should be clustered together on disk to allow sequential prefetching and other optimizations.
- Clustering is very application specific. For example, tuples describing nearby streets should be clustered together in geographic databases, tuples describing the line items of an invoice should be clustered with the invoice tuple in an inventory control application. Hashing tends to randomize data rather than cluster it.
- ***Range partitioning*** clusters tuples with similar attributes together in the same partition. It is good for sequential and associative access, and is also good for clustering data.
- Fig. 1.11 shows range partitioning based on lexicographic order, but any clustering algorithm is possible.
- Range partitioning derives its name from the typical SQL range queries such as latitude BETWEEN 37 AND 39Arbre, Bubba, Gamma, Oracle, and Tandem provide range partitioning. The problem with range partitioning is that it risks *data skew*, where all the data is place in one partition, and *execution skew* in which all the execution occurs in one partition.
- **Hashing and round-robin** are less susceptible to these skew problems. Range partitioning can minimize skew by picking non-uniformly-distributed partitioning criteria.
- Bubba uses this concept by considering the access frequency *(heat)* of each tuple when creating partitions a relation; the goal being to balance the frequency with which each partition is accessed (its *temperature*) rather than the actual number of tuples on each disk (its volume) [COPE88].

- ✧ While partitioning is a simple concept that is easy to implement, it raises several new physical database design issues.
- ✧ Each relation must now have a partitioning strategy and a set of disk fragments. Increasing the degree of partitioning usually reduces the response time for an individual query and increases the overall throughput of the system.
- ✧ For sequential scans, the response time decreases because more processors and disks are used to execute the query.
- ✧ For associative scans, the response time improves because fewer tuples are stored at each node and hence the size of the index that must be searched decreases.
- ✧ There is a point beyond which further partitioning actually increases the response time of a query. This point occurs when the cost of starting a query on a node becomes a significant fraction of the actual execution time

UNIT –III

15.4.3 Disadvantages of OODBMSs

- Lack of universal data model.
- Lack of experience.
- Lack of standards.
- Competition posed by RDBMS and the emerging ORDBMS products.
- Query optimization compromises encapsulation.
- Locking at object level may impact performance.
- Complexity due to increased functionality provided by an OODBMS.
- Lack of support for views.
- Lack of support for security.

Creation of Values of Complex Types

In SQL:1999 **constructor function**s are used to create values of structured types. A function with the same name as a structured type is a constructor function for the structured type. For instance, we could declare a constructor for the type *Publisher* like this:

```
create function Publisher (n varchar(20), b varchar(20))
        returns Publisher
        begin
                set name = n;
                set branch = b;
                end
```

We can then use *Publisher*('McGraw-Hill', 'New York') to create a value of the type *Publisher*.

Inheritance

Inheritance

Inheritance can be at the level of types, or at the level of tables. We first consider inheritance of types, then inheritance at the level of tables.

Type Inheritance

Suppose that we have the following type definition for people:

```
create type Person
(name varchar(20),
address varchar(20))
```

We may want to store extra information in the database about people who are students, and about people who are teachers. Since students and teachers are also people, we can use inheritance to define the student and teacher types in SQL:1999:

```
create type Student
under Person
(degree varchar(20),
department varchar(20))
create type Teacher
        under Person
(salary integer,
department varchar(20))
```

Both Student and Teacher inherit the attributes of Person—namely, name and address. Student and Teacher are said to be subtypes of Person, and Person is a supertype of Student, as well as of Teacher.

For instance, if our type system supports multiple inheritance, we can define a type for teaching assistant as follows:

```
create type TeachingAssistant
        under Student, Teacher
```

TeachingAssistant would inherit all the attributes of Student and Teacher. There is a problem, however, since the attributes name, address, and department are present in Student, as well as in Teacher.

The attributes name and address are actually inherited from a common source, Person. So there is no conflict caused by inheriting them from Student as well as Teacher.

However, the attribute department is defined separately in Student and Teacher. In fact, a teaching assistant may be a student of one department and a teacher in another department.

To avoid a conflict between the two occurrences of department, we can rename them by using an as clause, as in this definition of the type TeachingAssistant:

```
create type TeachingAssistant
under Student with (department as student-dept),
Teacher with (department as teacher-dept)
```

The SQL:1999 standard also requires an extra field at the end of the type definition, whose value is either final or not final. The keyword final says that subtypes may not be created from the given type, while not final says that subtypes may be created.

Table Inheritance

Subtables in SQL:1999 correspond to the E-R notion of specialization/generalization.
For instance, suppose we define the people table as follows:

```
create table people of Person
```

We can then define tables students and teachers as subtables of people, as follows:

```
create table students of Student
under people
create table teachers of Teacher
under people
```

The types of the subtables must be subtypes of the type of the parent table. Thereby, every attribute present in people is also present in the subtables.

Further, when we declare students and teachers as subtables of people, every tuple present in students or teachers becomes also implicitly present in people.

Thus, if a query uses the table people, it will find not only tuples directly inserted into that table, but also tuples inserted into its subtables, namely students and teachers. However, only those attributes that are present in people can be accessed.

Multiple inheritance is possible with tables, just as it is possible with types.
For example, we can create a table of type TeachingAssistant:

```
create table teaching-assistants of TeachingAssistant
under students, teachers
```

5.2 OBJECT RELATIONAL DATABASE

Object relational database systems (ORDBMS) are the relational database systems that have been extended to include the features of object-oriented paradigm. The SQL:1999 extends the SQL to support the complex data types and object-oriented features such as inheritance. In this section, we discuss implementation of these features in SQL.

5.2.1 Structured Types

In addition to built-in data types, SQL:1999 allows us to create new types which are called user-defined types in SQL. Structured type is a form of user-defined type that allows representing complex data types. Using structured types, composite attributes as well as multivalued attributes of E-R diagrams are represented efficiently.

Using Type Constructors

Structured data types are defined using type constructors, namely, *row* and *array*. Row type is used to specify the type of a composite attribute of a relation. A row type can be specified using the syntax given here.

```
CREATE TYPE <Type_Name> AS [ROW](<attribute1> <data_type1>,
                                 <attribute2> <data_type2>,
                                        :
                                 <attributen> <data_typen>
                                );
```

For example, a row type to represent a composite attribute `Address` with component attributes `HouseNo`, `Street`, `City`, `State`, and `Zip` can be specified as follows.

```
CREATE TYPE AddressType AS
          (HouseNo   VARCHAR(30),
           Street    VARCHAR(15),
           City      VARCHAR(12),
           State     VARCHAR(6),
           Zip       INTEGER(6));
```

We can now use the defined type (that is, `AddressType`) as a type for an attribute while creating a relation. For example, a relation `PUBLISHER` can be created by declaring an attribute `Paddress` of type `AddressType` as shown here.

```
CREATE TABLE PUBLISHER(
            P_ID       VARCHAR(20),
            Pname      VARCHAR(50),
            Paddress   AddressType);
```

The attribute Paddress in the PUBLISHER relation is now a composite attribute having HouseNo, Street, City, State, and Zip as its components.

The row types discussed so far have names associated with them. SQL provides an alternative way of defining composite attributes by using unnamed row types. To illustrate this, consider the following declaration.

```
CREATE TABLE PUBLISHER(
P_ID      VARCHAR(20),
Pname     VARCHAR(50),
Paddress  ROW(HouseNo   VARCHAR(30),
              Street    VARCHAR(15),
              City      VARCHAR(12),
              State     VARCHAR(6),
              Zip       INTEGER(6)));
```

Note that the attribute Paddress has unnamed type and rows of the relation also have an unnamed type.

Now in order to access the components of a composite attribute "dot" notation is used. For example, Paddress.City returns the city component of the Paddress attribute. For example, consider the following query.

```
SELECT Paddress.HouseNo, Paddress.City
FROM PUBLISHER;
```

In addition to creating an attribute of user-defined type in a relation, SQL also allows creating a relation whose rows are of a user-defined type. For example, we can define a type PublisherType, which can be further used to create a relation PUBLISHER as follows.

```
CREATE TYPE PublisherType AS(
     P_ID         VARCHAR(20),
     Pname        VARCHAR(50),
     Paddress     AddressType);

CREATE TABLE PUBLISHER OF PublisherType;
```

An array type is used to specify an attribute (multivalued attribute) whose value will be a collection. For example, an author can have many phone numbers, thus, the attribute Phone can be represented using array type in the type AuthorType as follows.

```
CREATE TYPE AuthorType AS(
Aname     VARCHAR(30)     NOT NULL,
State     VARCHAR(15),
City      VARCHAR(15),
Zip       VARCHAR(10),
Phone     VARCHAR(20)     ARRAY[5],
URL       VARCHAR(30));
```

Note that the attribute Phone can store at most five values. Now we can create a relation AUTHOR of type AuthorType, and can insert a row in it using the following statements.

```
CREATE TABLE AUTHOR OF AuthorType;

INSERT INTO AUTHOR VALUES
```

```
('James Erin', 'Georgia', 'Atlanta', '31896',
ARRAY['376045','376123'], 'www.ejames.com');
```

In order to access or update array elements, we use array index. For example, the following query retrieves the name and first phone number of the authors who live in *Georgia*.

```
SELECT Aname, Phone[1]
FROM AUTHOR WHERE State = 'Georgia';
```

Defining Methods We can also define methods on the structured types. The methods are declared as part of the type definition as shown here.

```
CREATE TYPE PublisherType AS(
      P_ID       VARCHAR(20),
      Pname      VARCHAR(50),
      Paddress   VARCHAR(25))
      METHOD ShowName (P_ID
      VARCHAR(20))
      RETURNS    VARCHAR(50));
```

Here a method `ShowName` is declared that takes `P_ID` as parameter and it is supposed to return the name of the publisher. The method body is created separately as shown here.

```
CREATE INSTANCE METHOD ShowName(P_ID VARCHAR(20))
      RETURNS VARCHAR(50)
      FOR PublisherType
BEGIN
      RETURN SELF.Pname;
END
```

In this declaration, the `FOR` clause indicates that the method is declared for the type `PublisherType`, and the keyword `INSTANCE` indicates that the method executes on an instance of `PublisherType`. Note that the method returns `Pname` by using the variable `SELF`, which refers to the current instance of `PUBLISHER` on which the method is invoked. For example, we can invoke the method `ShowName` as follows:

```
SELECT ShowName('P002')
FROM PUBLISHER;
```

In object database system, each object is assigned an object identifier (OID), which uniquely identifies the object over its entire lifetime. Database system is responsible for ensuring the uniqueness of objects by assigning a system generated OID to the objects. The value of OID is used internally by the system for object identification and to manage inter object references. However, its value is invisible to database users. Another important property of OID is immutability, which means, the value of OID of any object should not change. In addition, each OID should be used only once. In other words, even if the object is removed from the database, its OID should not be assigned to another object.

The value of OID can be used to refer to the object from anywhere in the database. In SQL:1999, every tuple in a table—defined in terms of structured type—can be treated as an object, and therefore, can be assigned a unique OID. In such tables, an attribute of a type may be a reference (specified using keyword `REF`) to a tuple of same (or different) table. To understand the concept, consider the type `Author-BookType` and the table `AuthorBook`, which are shown here.

```
CREATE TYPE AuthorBookType AS(
     Book_id REF(BookType) SCOPE BOOK,
     Author_id REF(AuthorType) SCOPE AUTHOR);

CREATE TABLE AuthorBook OF AuthorBookType;
```

In the type `AuthorBookType`, the keyword `SCOPE` specifies that the values in the attributes `Book_id` and `Author_id` are references to the rows in the `BOOK` and `AUTHOR` table, respectively. This concept of references is similar to the concept of the foreign keys. Note that the referenced table (`BOOK` and `AUTHOR`) must have an attribute that stores the identifier of the row. This attribute, known as **self-referential attribute**, is specified by using `REF IS` clause while creating the table as shown here.

```
CREATE TABLE AUTHOR OF AuthorType
REF IS A_ID SYSTEM GENERATED;
```

This statement specifies that the values of the attribute `A_ID` are generated automatically by the database system.

UNIT-IV

XML: Extensible Markup Language

- XML neither programming language nor presentation language.
- It is used to transfer data between applications and databases.
- It describes the data and focuses on what data is.
- *XML tags are not predefined in XML. You must define your own tag*
- XML uses a DTD (Document Type Definition) to formally describe the data.

Features of XML

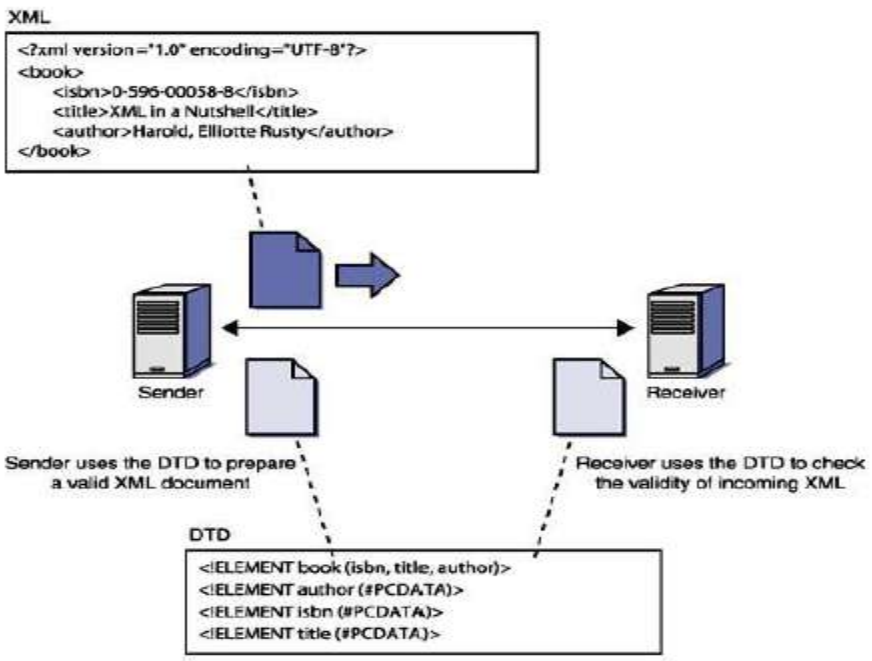
- XML is heavily used as a format for document storage and processing, both online and offline.
- Enhances search ability, making it possible for search engines to categorize data instead of wasting processing power on context-based full-text searches.
- XML does not allow References to external data entities. Named character references are not allowed in XML.
- XML does not allow empty comment declaration.
- XML is extensible, because it only specifies the structural rules of tags. No specification on tags them self.
- Excellent for handling data with a complex structure
- Handles data in a tree structure having one-and only one-root element
- Excellent for long-term data storage and data reusability
- Xml separate data from HTML
- XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.
- This makes it much easier to create data that can be shared by different applications.
- XML Simplifies Data Transport
- XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
- XML increase data availability.

Difference Between XML & HTML

Sr No.	HTML	XML
1	HTML was designed to display data, with focus on how data looks	XML was designed to describe data, with focus on what data is
2	It is not case Sensitive	XML is case Sensitive
3	HTML is a markup language itself	XML provides a framework for defining markup languages.
4	HTML is a presentation language	XML is neither a programming language nor a presentation language.
5	HTML is used for designing a web-page to be rendered on the client side	XML is used basically to transport data between the application and the database.
6	HTML has it own predefined tags	XML flexible i.e custom tags can be defined and the tags are invented by the author of the XML document
7	HTML is about displaying data,hence static	XML is about carrying information,hence dynamic.
8	HTML is not strict if the user does not use the closing tags	XML makes it mandatory for the user the close each tag that has been used.

DTD XML

- A Document Type Definition (DTD) defines the legal building blocks of an XML document.
- It defines the document structure with a list of legal elements and attributes.
- A DTD can be declared inline inside an XML document, or as an external reference.



- **Internal DTD Declaration**
- If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example XML document with an internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)> ]>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading> <body>Don't
forget me this weekend</body> </note>
```

- !DOCTYPE note defines that the root element of this document is note
- !ELEMENT note defines that the note element contains four elements: "to, from, heading, body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

External DTD Declaration

- If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:
- <!DOCTYPE root-element SYSTEM "filename">
- This is the same XML document as above, but with an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And this is the file "note.dtd" which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Why Use a DTD?

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data

XML Schema:

- XML Schema is an XML-based alternative to DTD.
- An XML schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

What is an XML Schema?

- The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements

- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes

  defines default and fixed values for elements and attributes

XML Schemas are the Successors of DTDs

- We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs.

  Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces
- XML Schemas are much more powerful than DTDs

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

- Another great strength about XML Schemas is that they are written in XML.

Some benefits of that XML Schemas are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM

  You can transform your Schema with XSLT

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
- However, an XML element with a data type like this:

  <date type="date">2004-03-11</date>

- ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

- XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

• Reuse your Schema in other Schemas

• Create your own data types derived from the standard types

• Reference multiple schemas in the same document

What is a Simple Element?

- A simple element is an XML element that can contain only text.
- It cannot contain any other elements or attributes.
- However, the "only text" restriction is quite misleading. The text can be of many different types.

- It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.
- You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

The syntax for defining a simple element is:

- <xs:element name="xxx" type="yyy"/>

  where xxx is the name of the element and yyy is the data type of the element.
- XML Schema has a lot of built-in data types. The most common types are:

  • xs:string • xs:decimal xs:integer • xs:boolean • xs:date • xs:time

Example

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age> <dateborn>1970-
03-27</dateborn>
```

And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

API for XML

DOM

- *document object model* (DOM), treats XML content as a tree, with each element represented by a node, called a DOMNode.
- Programs may access parts of the document in a navigational fashion, beginning with the root.
- DOM libraries are available for most common programming langauges and are even present in Web browsers, where it may be used to manipulate the document displayed to the user.
- The Java DOM API provides an interface called Node, and interfaces Element and Attribute, which inherit from the Node interface.
- The Node interface provides methods such as getParentNode(), getFirstChild(), and getNextSibling(),to navigate the DOM tree, starting with the root node.

- Subelements of an element can be accessed by name getElementsByTagName(name), which returns a list of all child elements with a specified tag name.

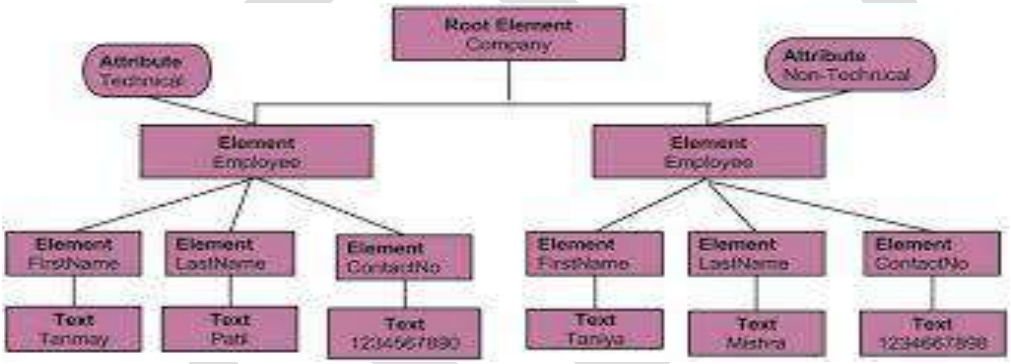
- individual members of the list can be accessed by the method item(i), which returns the *i*th element in the list.

- Attribute values of an element can be accessed by name, using the method getAttribute(name). The text value of an element is modeled as a Text node.

- The method getData() on the Text node returns the text contents.

- DOM also provides a variety of functions for updating the document by adding and deleting attribute and element children of a node, setting node values, and so on.

- DOM can be used to access XML data stored in databases, and an XML database can be built using DOM as its primary interface for accessing and modifying data.



-

SAX (Simple API for XML) is an event sequential access parser API developed by the XML-DEV mailing list for XML documents.

- SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM).

Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

- It does not first create any internal structure

- Client does not specify what methods to call

- Client just overrides the methods of the API and place his own code inside there

- When the parser encounters start-tag, end-tag,etc., it thinks of them as **events**
- When such an **event occurs, the handler automatically** ***calls back to a particular method overridden by the client, and feeds as arguments the method what it sees***
- SAX parser is event-based,
- it works like an event handler in Java (e.g. MouseAdapter)

Advantage:

(1) It is simple

(2) It is memory efficient

(3) It works well in stream application

Disadvantage:

- The data is broken into pieces and clients never have all the information as a whole unless they create their own data structure

Querying and Transformation

In particular, tools for querying and transformation of XML data are essential to extract information from large bodies of XML data, and to convert data between different representations (schemas) in XML.

Just as the output of a relational query is a relation, the output of an XML query can be an XML document. As a result, querying and transformation can be combined into a single tool.

Several languages provide increasing degrees of querying and transformation capabilities:

• XPath is a language for path expressions, and is actually a building block for the remaining two query languages.

• XSLT was designed to be a transformation language, as part of the XSL style sheet system, which is used to control the formatting of XML data into HTML or other print or display languages. Although designed for formatting, XSLT can generate XML as output, and can express many interesting queries.

• XQuery has been proposed as a standard for querying of XML data

XPath

XPath addresses parts of an XML document by means of path expressions.

The language can be viewed as an extension of the simple path expressions in object-oriented and object-relational databases

A **path expression** in XPath is a sequence of location steps separated by "/"
The result of a path expression is a set of values.

```
<bank-2>
    <account account-number="A-401" owners="C100 C102">
        <branch-name> Downtown </branch-name>
        <balance> 500 </balance>
    </account>
    <account account-number="A-402" owners="C102 C101">
        <branch-name> Perryridge </branch-name>
        <balance> 900 </balance>
    </account>
    <customer customer-id="C100" accounts="A-401">
        <customer-name>Joe</customer-name>
        <customer-street> Monroe </customer-street>
        <customer-city> Madison </customer-city>
    </customer>
    <customer customer-id="C101" accounts="A-402">
        <customer-name>Lisa</customer-name>
        <customer-street> Mountain </customer-street>
        <customer-city> Murray Hill </customer-city>
    </customer>
    <customer customer-id="C102" accounts="A-401 A-402">
        <customer-name>Mary</customer-name>
        <customer-street> Erin </customer-street>
        <customer-city> Newark </customer-city>
    </customer>
</bank-2>
```

For instance, on the document in above Figure

the XPath expression
/bank-2/customer/name

would return these elements:

<name>Joe</name>
<name>Lisa</name>
<name>Mary</name>

The expression
/bank-2/customer/name/text()
would return the same names, but without the enclosing tags.

X-Query

• The best way to explain XQuery is to say that XQuery is to XML what SQL is to database tables.

• It is the language for querying XML data.

• XQuery is a language for finding and extracting elements and attributes from XML documents.

• XQuery is designed to query XML data – not just XML files, but anything that can appear as XML.

Uses of XQuery

• Extract information to use in a Web Service

• Query XML documents

• Read data from databases and generate reports

• Transform XML data

FLWOR

• For – binds a variable to each item returned by the in expression

Let – allows variable assignments

Where – used to specify criteria for result

Order by – defines the sort-order

Return – specifies what is to be returned

General expression: FLWOR expression FOR < for-variable > IN < in-expression >
LET < let-variable > := < let-expression> [WHERE < filter-expression>]
[ORDER BY < order-specification >] RETURN <expression>

Example: retrieve the name of instructors who have a salary that is higher than 30000

• for $x in doc(”university.xml")/university/instructor
where $x/salary>30000
return <instr> {$x/name} </instr>

For/Let Clause :

• for <variable> in <expression>, . .

• Variables begin with $.

• To bind values to single or multiple variables FOR/LET clause used

• When iteration are required FOR is used.

• LET clause values to a single or multiple variables as FOR clause does but without iteration

Where Clause

Where clause is optional. It is used to specify one or more conditions as per the requirement.
It is used to restrict the number of nodes returned by expression

Return Clause :

For each query return clause is evaluated.
The result produced are concatenated & return to users

XSL Transformations:

XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

XPath is a language for navigating in XML documents.

XSLT stands for XSL Transformations

With XSLT you can add/remove elements and attributes to or from the output file.

You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display.

XSLT uses XPath to find information in an XML document.

XPath is used to navigate through elements and attributes in XML documents.

An XSL style sheet is basically a series of pattern-action rules and looks like an XML document with a mixture of two kinds of elements: those defined by XSL and those defined by the object language. The patterns are similar to CSS's selectors, but the action part may create an arbitrary number of "objects." The action part of the rule is called the "template" in XSL, and a template and a pattern together are referred to as a "template rule."

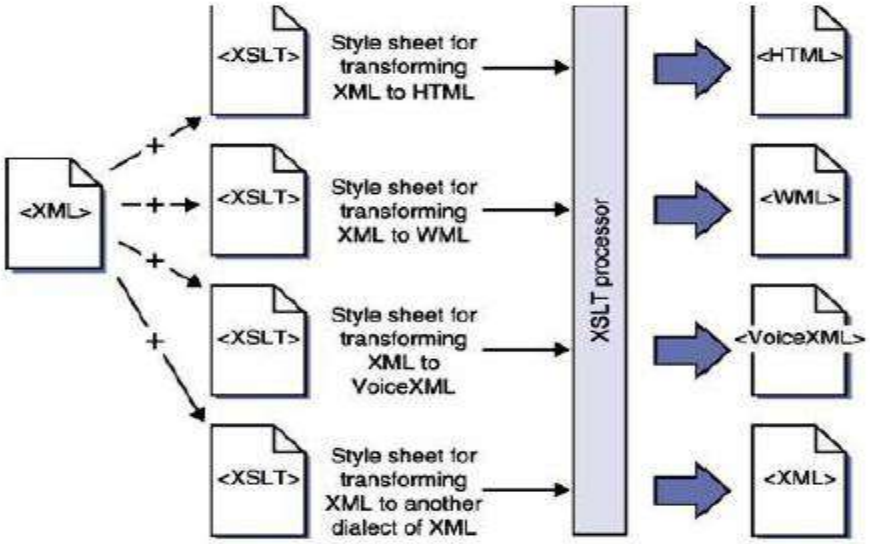


Figure :XSLT Model

Writing XSLT is different from writing a program that specifies in a step-by-step manner what a processor is expected to do. XSLT transformations occur through a series of tests that compare parts of an XML document against templates defined in an XSLT document. Templates act like rules that are matched against the contents of a document; when a match occurs, whatever the template specifies is output. For example, assume an XML document contains the fragment

```
<title> A History of PI </title>
```

Given a template that matches for a title element

```
<xsl:template match="title">
  <H2>
   <xsl:value-of/>
  </H2>
</xsl:template>
```

the output will be

```
<H2> A History of PI </H2>
```

where the `<xsl:value-of/>` tag is replaced by the actual value of the title element. Using this capability, one can go from XML to HTML or other forms of XML.

The following is a complete style sheet that generates a Web page of all books in an XML document:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="books">
   <html>
   <head>
        <title>ZwiftBooks - At Your Doorstep</title>
   </head>
   <body>
   <h2>Weekly Specials</h2>

   <xsl:apply-templates/>

   </body>
   </html>
</xsl:template>

<xsl:template match="book">
   <p></p><b>Title:</b> <xsl:value-of select="title/><br/>
           <em>Author:</em> <xsl:value-of select="author"/>
           <em>ISBN:</em> <xsl:value-of select="isbn"/>

   <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```


Let's look at a portion of this page in a bit more detail.

```
<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>
```

The slash in the first line tells the processor that this node applies to the root level of the XML document. Think of the root level as an imaginary pair of tags surrounding the entirety of an XML document which must be addressed before you can get to the actual tags. The `apply-templates` tag tells the processor to look at everything that occurs beneath the current level; in this case it means, "examine everything." This command can be found in most XSLT documents.

In our example note that there are two templates: one that matches for books and another that matches for a book. When a `books` element is found, the outer structure of an HTML page is printed. Within the `books` template there is another `apply-templates` element which says to keep trying to find more matches in the XML document. When the XSLT matches a `book` element, the `book` template is activated and details of the book are printed as HTML.

XML applications

B2B Exchange :XML provides necessary standards which is required to exchange B2B data amongst the organizations.
XML is less expensive & flexible

• **Legacy System Integration** :XML provides the facility to integrate legacy system data with modern e-commerce System

It also transfer data from multiple heterogeneous databases to data warehouse
• **Web Page Development** :for creation of web page XML provide many features.

• **Database Support** :A DBMS which supports XML exchange creates new system by enabling integration with external system

On the other side database in its native format stores the XML data.

Database Meta-dictionaires: for databases XML can also facilitate creation of meta dictionaries & vocabularies.
Meta dictionary created are not dependent on the DBMS type, it uses common language for data description.
• **XML Databases** : To handle vast XML data exchange, to manage & utilize data efficiently many XML software are available in the market.

Such products also includes OODBMS with XML interfaces to full XML database engines & servers.
XML provides advanced features than provided by traditional DBMS it used to handle complex relationship.
XML can store the contents of book which include chapters, paragraphs, headers etc.

4. Discuss the basic characteristics of a data warehouse.

Ans: The four basic characteristics of a data warehouse are:

- **Subject oriented**. A data warehouse is organized around a major subject such as customer, products and sales. That is, data are organized according to a subject instead of application. For example, an insurance company using a data warehouse would organize its data by customer, premium and claim instead of by different policies.
- **Nonvolatile:** A data warehouse is always a physically separated store of data. Due to this separation, data warehouse does not require transaction processing, recovery, concurrency control and so on. The data are not overwritten or deleted once they enter the data warehouse, but are only loaded, refreshed and accessed for queries.
- **Time varying:** Data are stored in a data warehouse to provide a historical perspective. Thus, the data in the data warehouse are time-variant or historical in nature.
- **Integrated:** A data warehouse is usually constructed by integrating multiple, heterogeneous sources such as relational databases and flat files. The database contains data from most or all of an organization's operational applications, and these data are made consistent.

Table 3.1 Comparison between OLTP and OLAP systems.

Feature	*OLTP*	*OLAP*
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements, decision support
DB design	ER based, application-oriented	star/snowflake, subject-oriented
Data	current; guaranteed up-to-date	historical; accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions
Number of users	thousands	hundreds
DB size	100 MB to GB	100 GB to TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Why Have a Separate DataWarehouse?

Because operational databases store huge amounts of data, you may wonder, "why not perform on-line analytical processing directly on such databases instead of spending additional time and resources to construct a separate data warehouse?"

A major reason for such a separation is to help promote the high performance of both systems. An operational database is designed and tuned from known tasks and workloads, such as indexing and hashing using primary keys, searching for particular records, and optimizing "canned" queries. On the other hand, data warehouse queries are often complex.

They involve the computation of large groups of data at summarized levels, and may require the use of special data organization, access, and implementation methods based on multidimensional views.

Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks.

Moreover, an operational database supports the concurrent processing of multiple transactions. Concurrency control and recovery mechanisms, such as locking and logging, are required to ensure the consistency and robustness of transactions.

An OLAP query often needs read-only access of data records for summarization and aggregation.

Concurrency control and recovery mechanisms, if applied for such OLAP operations, may jeopardize the execution of concurrent transactions and thus substantially reduce the throughput of an OLTP system.

A Multidimensional Data Model

Data warehouses and OLAP tools are based on a multidimensional data model. This model views data in the form of a data cube

Stars, Snowflakes, and Fact Constellations: Schemas for Multidimensional Databases

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships between them. Such a data model is appropriate for on-line transaction processing.

A data warehouse, however, requires a concise, subject-oriented schema that facilitates on-line data analysis.

The most popular data model for a data warehouse is a multidimensional model.

Such a model can exist in the form of a star schema, a snowflake schema, or a fact constellation

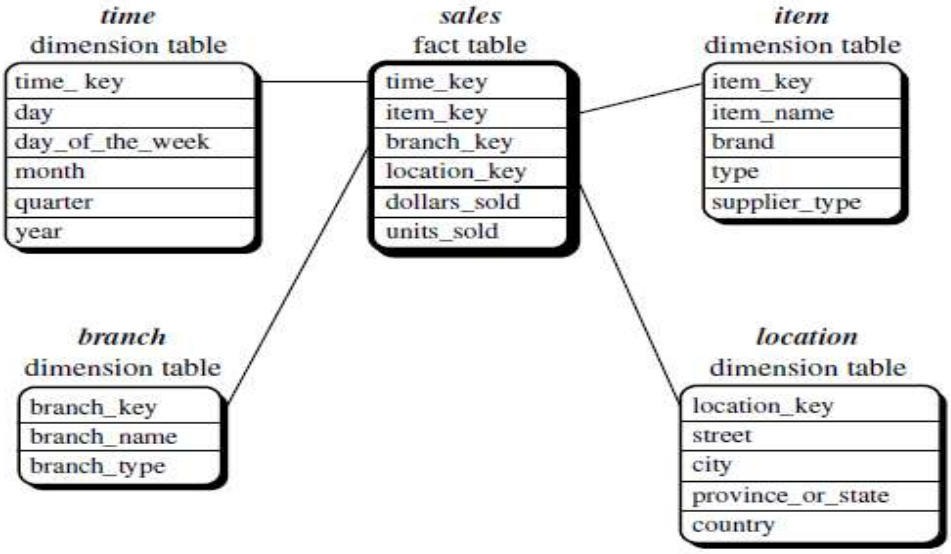
schema. Let's look at each of these schema types.

Star schema: The most common modeling paradigm is the star schema, in which the data warehouse contains (1) a large central table (fact table) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (dimension tables), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

Example 3.1 Star schema. A star schema for AllElectronics sales is shown in Figure 3.4. Sales are considered along four dimensions, namely, time, item, branch, and location. The schema contains

a central fact table for sales that contains keys to each of the four dimensions, along with two measures: dollars sold and units sold.To minimize the size of the fact table, dimension identifiers (such as time key and item key) are system-generated identifiers. Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the location dimension table contains the attribute set flocation key, street, city, province or state, countryg. This constraint may introduce some redundancy. For example, "Vancouver" and "Victoria" are both cities in the Canadian province of British Columbia. Entries for such cities in the location dimension table will create redundancy among the attributes province or state and country, that is, (..., Vancouver, British Columbia, Canada) and (..., Victoria, British Columbia, Canada).Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are normalized, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake**.**



e 3.4 Star schema of a data warehouse for sales.

The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies. Such a table is easy to maintain and saves storage space. However, this saving of space is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed to execute a query.

Snowflake schema. A snowflake schema for *AllElectronics* sales is given in Figure 3.5. Here, the *sales* fact table is identical to that of the star schema in Figure 3.4. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for *item* in the star schema is normalized in the snowflake schema, resulting in new *item* and *supplier* tables. For example, the *item* dimension table now contains the attributes *item_key, item_name, brand, type*, and *supplier_key*, where *supplier_key* is linked to the *supplier* dimension table, containing *supplier_key* and *supplier_type* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two new tables: *location* and *city*. The *city_key* in the new *location* table links to the *city* dimension. Notice that further normalization can be performed on *province_or_state* and *country* in the snowflake schema shown in Figure 3.5, when desirable. ■

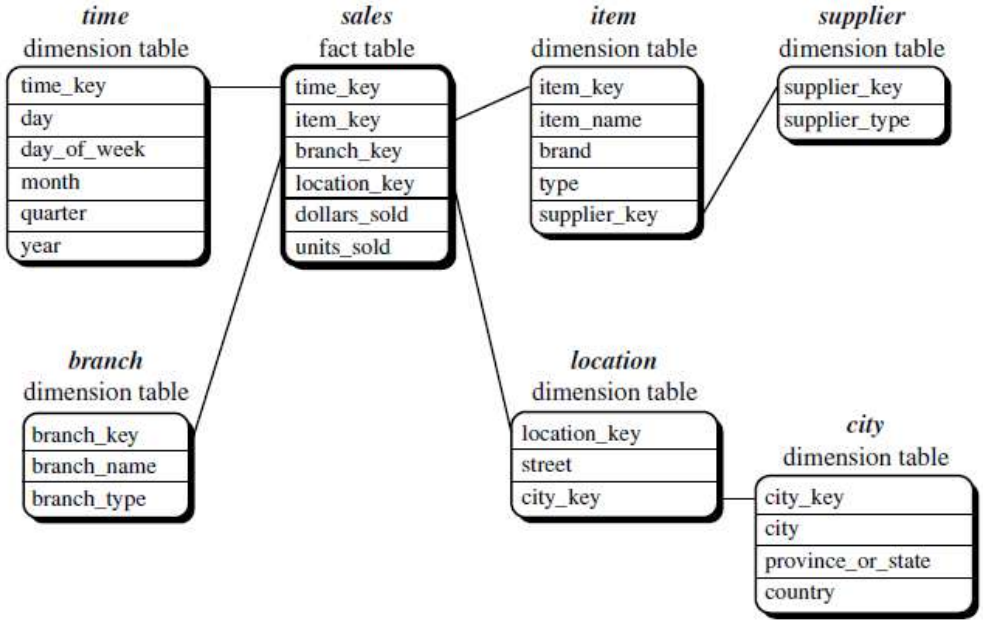


Figure 3.5 Snowflake schema of a data warehouse for sales.

Fact constellation: Sophisticated applications may require multiple fact tables to share dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a galaxy schema or a fact constellation.

Example 3.3 Fact constellation. A fact constellation schema is shown in Figure 3.6. This schema specifies two fact tables, sales and shipping. The sales table definition is identical to that of the star schema (Figure 3.4).

The shipping table has five dimensions, or keys: item key, time key, shipper key, from location, and to location, and two measures: dollars cost and units shipped.

A fact constellation schema allows dimension tables to be shared between fact tables.

For example, the dimensions tables for time, item, and location are shared between both the sales and shipping fact tables.

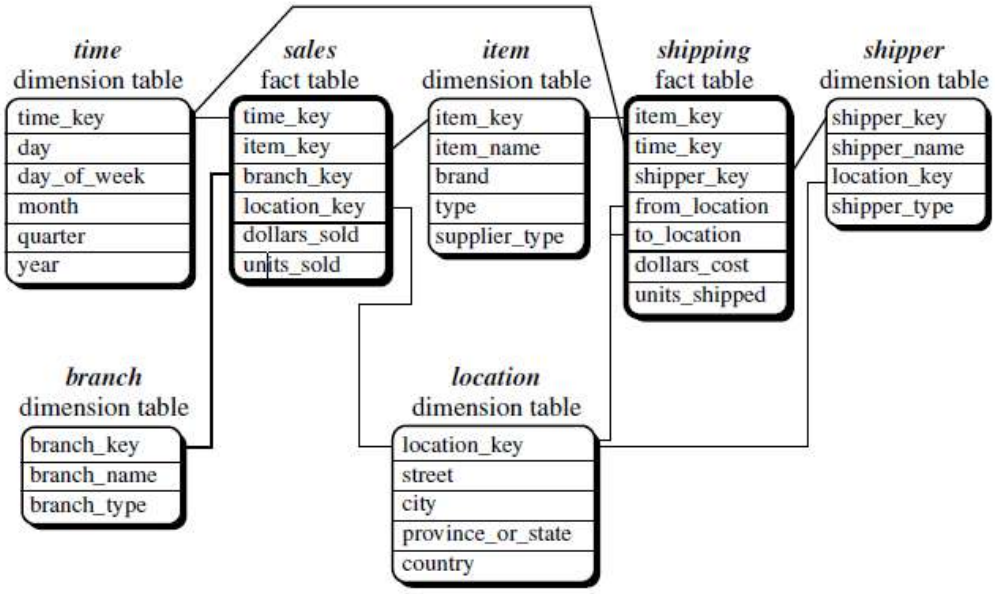


Figure 3.6 Fact constellation schema of a data warehouse for sales and shipping.

Examples for Defining Star, Snowflake, and Fact Constellation Schemas

"How can I define a multidimensional schema for my data?" Just as relational query languages like SQL can be used to specify relational queries, a data mining query language can be used to specify data mining tasks. In particular, we examine how to define data warehouses and data marts in our SQL-based data mining query language, DMQL. Data warehouses and data marts can be defined using two language primitives, one for cube definition and one for dimension definition. The cube definition statement has the following syntax:

define cube hcube namei [hdimension listi]: hmeasure listi

The dimension definition statement has the following syntax:

define dimension hdimension namei as (hattribute or dimension listi)

Let's look at examples of how to define the star, snowflake, and fact constellation schemas of Examples 3.1 to 3.3 using DMQL.

Example 3.4 Star schema definition. The star schema of Example 3.1 and Figure 3.4 is defined in

DMQL as follows:

define cube sales star [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)

define dimension item as (item key, item name, brand, type, supplier type)

define dimension branch as (branch key, branch name, branch type)

define dimension location as (location key, street, city, province or state,

country)

The define cube statement defines a data cube called sales star, which corresponds to the central sales fact table of Example 3.1. This command specifies the dimensions and the two measures, dollars sold and units sold. The data cube has four dimensions, namely, time, item, branch, and location. A define dimension statement is used to define each of the dimensions.

Example 3.5 Snowflake schema definition. The snowflake schema of Example 3.2 and Figure 3.5 is

defined in DMQL as follows:

define cube sales snowflake [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)

define dimension item as (item key, item name, brand, type, supplier (supplier key, supplier type))

define dimension branch as (branch key, branch name, branch type)

define dimension location as (location key, street, city (city key, city, province or state, country))

This definition is similar to that of sales star (Example 3.4), except that, here, the item

and location dimension tables are normalized.

For instance, the item dimension of the sales star data cube has been normalized in the sales snowflake cube into two dimension tables, item and supplier. Note that the dimension definition for supplier is specified within

the definition for item. Defining supplier in this way implicitly creates a supplier key in the item dimension table definition. Similarly, the location dimension of the sales star data cube has been normalized in the sales snowflake cube into two dimension tables, location and city. The dimension definition for city is specified within the definition for location.

In this way, a city key is implicitly created in the location dimension table definition.

Finally, a fact constellation schema can be defined as a set of interconnected cubes.

Below is an example.

Example 3.6 Fact constellation schema definition. The fact constellation schema of Example 3.3 and

Figure 3.6 is defined in DMQL as follows:

define cube sales [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)

define dimension item as (item key, item name, brand, type, supplier type)

define dimension branch as (branch key, branch name, branch type)

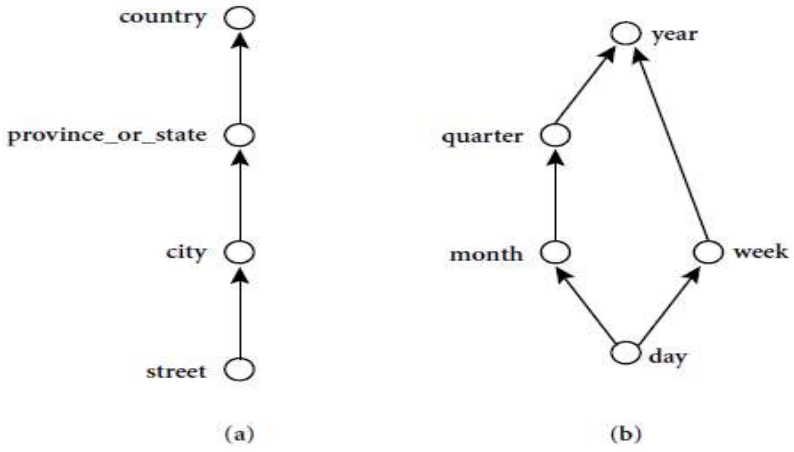
define dimension location as (location key, street, city, province or state, country)

Concept Hierarchies

A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Consider a concept hierarchy for the dimension location.City values for location includeVancouver,Toronto,NewYork,andChicago.

Each city, however, can be mapped to the province or state to which it belongs. For example, Vancouver can bemappedtoBritishColumbia, andChicagotoIllinois.Theprovinces and states can in turn be mapped to the country to which they belong, such as Canada or the USA. These mappings forma concept hierarchy for the dimension location, mapping a set of low-level concepts (i.e., cities) to higher-level, more general concepts (i.e., countries). The concept hierarchy described above is illustrated in Figure 3.7.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension location is described by the attributes number, street, city, province or state, zipcode,andcountry.Theseattributes arerelatedbyatotalorder, forming a concept hierarchy such as "street < city < province or state < country". This hierarchy is shown in Figure 3.8(a). Alternatively, the attributes of a dimension may be organized in a partial order, forming a lattice. An example of a partial order for the time dimension based on the attributes day, week, month, quarter, and year is "day < fmonth <quarter; weekg < year".2 This lattice structure is shown in Figure 3.8(b). A concept hierarchy



.8 Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location*; (b) a lattice for *time*.

OLAP Operations in the Multidimensional Data Model

"How are concept hierarchies useful in OLAP?" In the multidimensional model, data are organized into multiple dimensions, and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides users with the flexibility to view data from different perspectives. A number of OLAP data cube operations exist to materialize these different views, allowing interactive querying and analysis of the data at hand. Hence, OLAP provides a user-friendly environment for interactive data analysis.

Example 3.8 OLAP operations. Let's look at some typical OLAP operations for multidimensional data.

Each of the operations described below is illustrated in Figure 3.10. At the center of the figure is a data cube for AllElectronics sales. The cube contains the dimensions location, time, and item, where location is aggregated with respect to city values, time is aggregated with respect to quarters, and item is aggregated with respect to itemtypes.

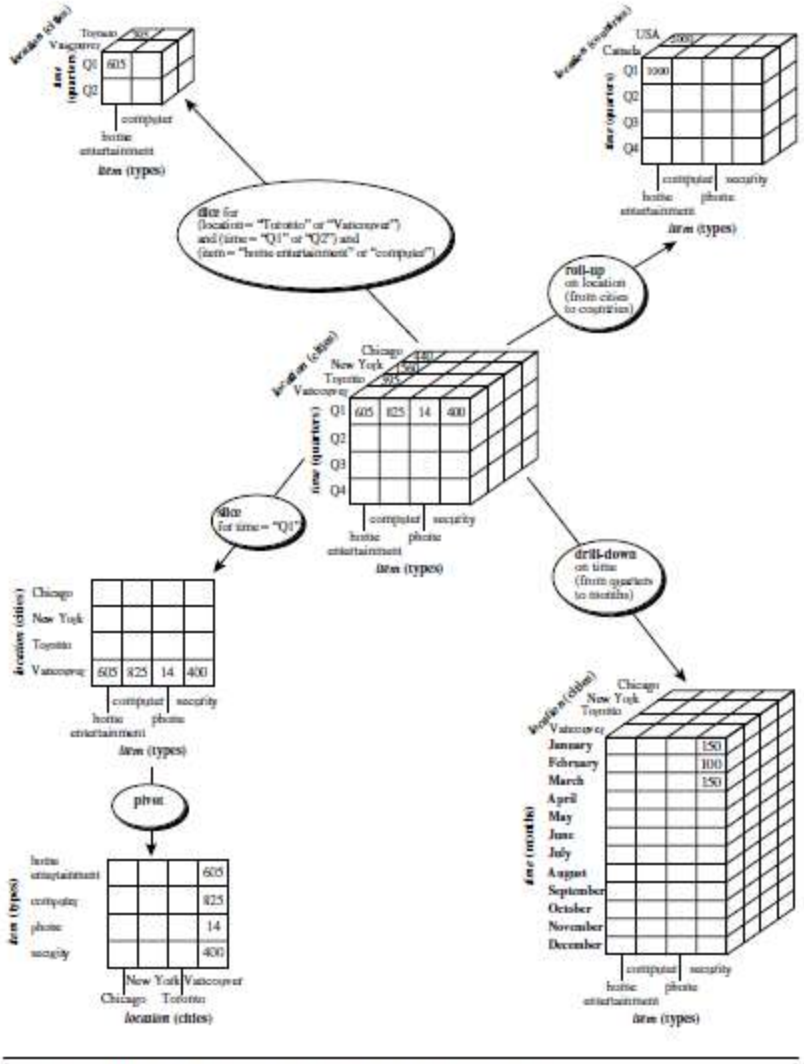


Figure 3.10 Examples of typical OLAP operations on multidimensional data.

Roll-up: The roll-up operation (also called the drill-up operation by some vendors) performs aggregation on a data cube, either by climbing up a concept hierarchy for a dimension or by dimension reduction. Figure 3.10 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for location given in Figure 3.7. This hierarchy was defined as the total order "street < city < province or state < country." The roll-up operation shown aggregates the data by ascending the location hierarchy from the level of city to the level of country. In other words, rather than grouping the data by city, the resulting cube groups the data by country.

When roll-up is performed by dimension reduction, one or more dimensions are removed fromthe given cube. For example, consider a sales data cube containing only the two dimensions location and time. Roll-up may be performed by removing, say, the time dimension, resulting in an aggregation of the total sales by location, rather than by location and by time.

Drill-down: Drill-down is the reverse of roll-up. It navigates from less detailed data to

more detailed data. Drill-down can be realized by either stepping down a concept hierarchy for a dimension or introducing additional dimensions. Figure 3.10 shows the result of a drill-down operation performed on the central cube by stepping down a concept hierarchy for time defined as "day < month < quarter < year." Drill-down occurs by descending the time hierarchy fromthe level of quarter to the more detailed level of month. The resulting data cube details the total sales per month rather than summarizing them by quarter.

Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube. For example, a drill-down on the central cube of Figure 3.10 can occur by introducing an additional dimension, such as customer group.

Slice and dice: The slice operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 3.10 shows a slice operation where the sales data are selected from the central cube for the dimension time using the criterion time = "Q1". The dice operation defines a subcube by performing a selection on two or more dimensions. Figure 3.10 shows a dice operation on the central cube based on the following selection criteria that involve three dimensions: (location = "Toronto" or "Vancouver") and (time = "Q1" or "Q2") and (item =

"home entertainment" or "computer").

Pivot (rotate): Pivot (also called rotate) is a visualization operation that rotates the data axes in view in order to provide an alternative presentation of the data. Figure 3.10 shows a pivot operation where the item and location axes in a 2-D slice are rotated

The Process of DataWarehouse Design

A data warehouse can be built using a top-down approach, a bottom-up approach, or a combination of both. The top-down approach starts with the overall design and planning.

It is useful in cases where the technology is mature and well known, and where the business problems that must be solved are clear and well understood. The bottom-up approach starts with experiments and prototypes. This is useful in the early stage of business modeling and technology development. It allows an organization to move forward at considerably less expense and to evaluate the benefits of the technology before making significant commitments. In the combined approach, an organization can exploit the planned and strategic nature of the top-down approach while retaining the rapid implementation and opportunistic application of the bottom-up approach. From the software engineering point of view, the design and construction of a data

warehouse may consist of the following steps: planning, requirements study, problemanalysis, warehouse design, data integration and testing, and finally deployment of the datawarehouse.

Large software systems can be developed using two methodologies: the waterfall method or the spiral method. The waterfall method performs a structured and systematic analysis at each step before proceeding to the next, which is like a waterfall, falling from one step to the next. The spiral method involves the rapid generation of increasingly functional systems, with short intervals between successive releases. This is considered a good choice for data warehouse development, especially for data marts, because the turnaround time is short, modifications can be done quickly, and new designs and technologies can be adapted in a timely manner.

In general, the warehouse design process consists of the following steps:

1. Choose a business process to model, for example, orders, invoices, shipments, inventory, account administration, sales, or the general ledger. If the business process is organizational and involves multiple complex object collections, a data warehouse model should be followed. However, if the process is departmental and focuses on the analysis of one kind of business process, a data mart model should be chosen.

2. Choose the grain of the business process. The grain is the fundamental, atomic level of data to be represented in the fact table for this process, for example, individual transactions, individual daily snapshots, and so on.

3. Choose the dimensions that will apply to each fact table record. Typical dimensions are time, item, customer, supplier, warehouse, transaction type, and status.

4. Choose the measures that will populate each fact table record. Typical measures are numeric additive quantities like dollars sold and units sold.

The goals of an initial data warehouse implementation should be specific, achievable, and measurable. This involves determining the time and budget allocations, the subset of the organization that is to be modeled, the number of data sources selected, and the number and types of departments to be served.

Once a data warehouse is designed and constructed, the initial deployment of the warehouse includes initial installation, roll-out planning, training, and orientation.

Platform upgrades and maintenance must also be considered. Data warehouse administration includes data refreshment, data source synchronization, planning for disaster recovery, managing access control and security, managing data growth, managing database performance, and data warehouse enhancement and extension. Scope management includes controlling the number and range of queries, dimensions, and reports; limiting the size of the data warehouse; or limiting the schedule, budget, or resources.

Various kinds of data warehouse design tools are available. Data warehouse development tools provide functions to define and edit metadata repository contents (such as schemas, scripts, or rules), answer queries, output reports, and ship metadata to and from relational database system catalogues. Planning and analysis tools study the impact of schema changes and of refresh performance when changing refresh rates or time windows.

Data Warehouse Models:

There are three data warehouse models.

1. Enterprise warehouse: An enterprise warehouse collects all of the information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data aswell as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond. An enterprise data warehouse may be

implemented on traditional mainframes, computer superservers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

2. Data mart:

A data mart contains a subset of corporate-wide data that is of value to aspecific group of users. The scope is confined to specific selected subjects. For example,a marketing data mart may confine its subjects to customer, item, and sales. Thedata contained in data marts tend to be summarized.

Data marts are usually implemented on low-cost departmental servers that areUNIX/LINUX- or Windows-based. The implementation cycle of a data mart ismore likely to be measured in weeks rather than months or years. However, itmay involve complex integration in the long run if its design and planning werenot enterprise-wide.

Depending on the source of data, data marts can be categorized as independent ordependent. Independent data marts are sourced fromdata captured fromone or moreoperational systems or external information providers, or fromdata generated locallywithin a particular department or geographic area. Dependent data marts are sourceddirectly from enterprise data warehouses.

Virtual warehouse:

A virtual warehouse is a set of views over operational databases. Forefficient query processing, only some of the possible summary views may be materialized.

A virtual warehouse is easy to build but requires excess capacity on operational database servers.

Data Pre-processing

Why Data Pre-processing?

Data in the real world is dirty. That is it is incomplete or noisy or inconsistent.

- Incomplete: means lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- e.g., occupation=“ ”
- Noisy: means containing errors or outliers
- e.g., Salary=“-10”
- Inconsistent: means containing discrepancies in codes or names
- e.g., Age=“42” Birthday=“03/07/1997”
- e.g., Was rating “1,2,3”, now rating “A, B, C”
- e.g., discrepancy between duplicate records

Why Is Data Dirty?

Data is dirty because of the below reasons.

- Incomplete data may come from
- “Not applicable” data value when collected
- Different considerations between the time when the data was collected and when it is analyzed.
- Human / hardware / software problems
- Noisy data (incorrect values) may come from

- Faulty data collection instruments
- Human or computer error at data entry
- Errors in data transmission
- Inconsistent data may come from
- Different data sources
- Functional dependency violation (e.g., modify some linked data)
- Duplicate records also need data cleaning

Why Is Data Pre-processing Important?

Data Pre-processing is important because:

- If there is No quality data, no quality mining results!
- Quality decisions must be based on quality data
- e.g., duplicate or missing data may cause incorrect or even misleading statistics.
- Data warehouse needs consistent integration of quality data
- Data extraction, cleaning, and transformation comprises themajority of the work of building a data warehouse

Multi-Dimensional Measure of Data Quality

A well-accepted multidimensional view has the following properties:

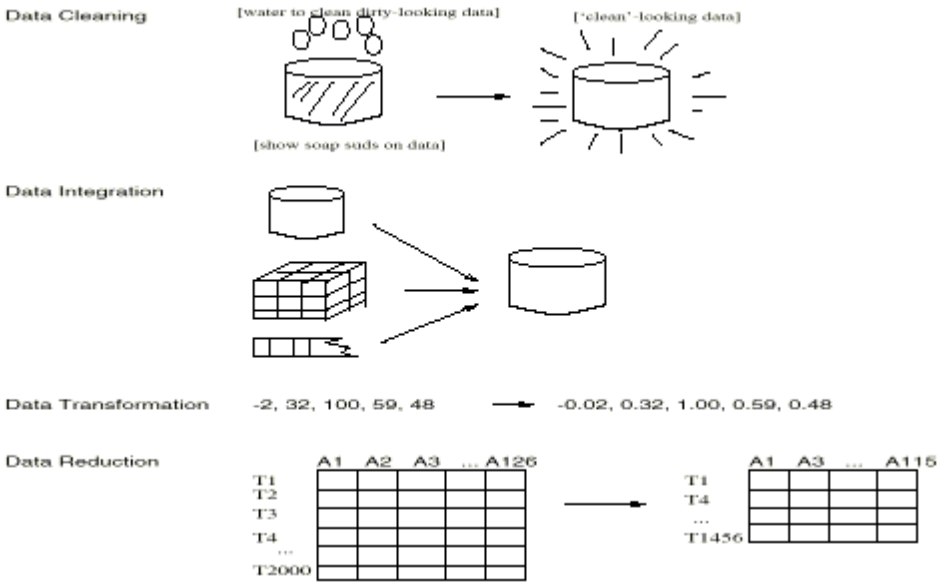
- Accuracy
- Completeness
- Consistency

Major Tasks in Data Pre-processing

Major tasks in data pre-processing are data cleaning, data integration, data transformation, data reduction and data discretization.

- Data cleaning
- Data Cleaning includes, filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies.
- Data integration
- Data Integration includes integration of multiple databases, data cubes, or files.
- Data transformation
- Data Transformation includes normalization and aggregation.
- Data reduction
- Data reduction is achieved by obtaining reduced representation of data in volume but produces the same or similar analytical results.
- Data Discretization
- Data Discretization is part of data reduction but with particular importance,especially for numerical data.

Forms of Data Pre-processing



Data Cleaning

- Importance of Data Cleaning
- "Data cleaning is one of the three biggest problems in data warehousing"—Ralph Kimball
- "Data cleaning is the number one problem in data warehousing"—DCI survey
- Data cleaning tasks are:
- fill in missing values
- Identifying outliers and smoothing out noisy data
- Correcting inconsistent data
- Resolving redundancy caused by data integration

Data Cleaning

Missing Data

Eg. Missing customer income attribute in the sales data

Methods of handling missing values:

a) Ignore the tuple

1) When the attribute with missing values does not contribute to any of the classes or has missing class label.

2) Effective only when more number of missing values are
there for many attributes in the tuple.

3) Not effective when only few of the attribute values are
missing in a tuple.

b) Fill in the missing value manually

1) This method is time consuming

2) It is not efficient

3) The method is not feasible

c) Use of a Global constant to fill in the missing value

1) This means filling with “Unknown” or “Infinity”

2) This method is simple

3) This is not recommended generally

d) Use the attribute mean to fill in the missing value

That is, take the average of all existing income values and fill in the missing
income value.

e) Use the attribute mean of all samples belonging to the same class as that of the
given tuple.

Say, there is a class “Average income” and the tuple with the missing value
belongs to this class and then the missing value is the mean of all the values in
this class.

f) Use the most probable value to fill in the missing value

This method uses inference based tools like Bayesian Formula, Decision tree
etc.

Noisy Data

Apply data smoothing techniques like the ones given below:

a) Binning Methods

Simple Discretization Methods: Binning

- Equal-width (distance) partitioning
- Divides the range into N intervals of equal size: uniform grid
- if A and B are the lowest and highest values of the attribute, the width of intervals will be: W = (B –A)/N.
- The most straightforward, but outliers may dominate presentation
- Skewed data is not handled well
- Equal-depth (frequency) partitioning
- Divides the range into N intervals, each containing approximately same number of samples

Data Integration

- Combines data from multiple sources into a single store.
- Includes multiple databases, data cubes or flat files

Schema integration

- Integrates meta data from different sources
- Eg. A.cust_id = B.cust_no

Entity Identification Problem

- Identify real world entities from different data sources
- Eg. Pay_type filed in one data source can take the values ‘H’ or ‘S’, Vs in another data source it can take the values 1 or 2

Detecting and resolving data value conflicts:

- For the same real world entity, the attribute value can be different in different data sources
- Possible reasons can be - Different interpretations, different representation and different scaling
- Eg. Sales amount represented in Dollars (USD) in one data source and as Pounds ($) in another data source.

Handling Redundancy in data integration:

- When we integrate multiple databases data redundancy occurs

- Object Identification – Same attributes / objects in different data sources may have different names.

- Derivable Data – Attribute in one data source may be derived from

Attribute(s) in another data source

Eg. Monthly_revenue in one data source and Annual revenue in

another data source.

- Such redundant attributes can be detected using Correlation Analysis

- So, Careful integration of data from multiple sources can help in

reducing or avoiding data redundancy and inconsistency which will

in turn improve mining speed and quality.

Data Reduction

Why Data Reduction?

- A database of data warehouse may store terabytes of data

- Complex data analysis or mining will take long time to run on the

complete data set

What is Data Reduction?

- Obtaining a reduced representation of the complete dataset

- Produces same result or almost same mining / analytical results as that of

original.

Data Reduction Strategies:

1. Data cube Aggregation

2. Dimensionality reduction – remove unwanted attributes

3. Data Compression

4. Numerosity reduction – Fit data into mathematical models

5. Discretization and Concept Hierarchy Generation

Unit –VI

INTRODUCTION TO DATABASE : SECURITY ISSUES

Types of Security

Database security is a very broad area that addresses many issues, including the following:

• Legal and ethical issues regarding the right to access certain information. Some information may be deemed to be private and cannot be accessed legally by unauthorized persons.

In the United States, there are numerous laws governing privacy of information.

• Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available-for example, credit ratings and personal medical records.

• System-related issues such as the system levels at which various security functions should be enforced-for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMSlevel.

• The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications-for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

Threats to Databases. Threats to databases result in the loss or degradation of some or all of the following security goals: integrity, availability, and confidentiality.

• Loss of integrity: Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.

• Lossof availability: Database availability refers to making objects available to a human user or a program to which they have a legitimate right.

• Loss of confidentiality: Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security.

Unauthorized, unanticipated, or unintentional disclosure could result in lossof public confidence, embarrassment, or legal action against the organization.

To protect databases against these types of threats four kinds of countermeasures can be implemented: access control, inference control, flow control, and encryption.

• Discretionary security mechanisms: These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

• Mandatory security mechanisms: These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization.

For example, a typical security policy is to permit users at a certain classification level to see only the data items classified at the user's own (or lower) classification level. An extension of this is role-based security, which enforces policies and privileges based on the concept of roles.

Database Security and the DBA

The database administrator (DBA) is the central authority for managing a database system. The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization.

The DBA has a DBAaccount in the DBMS, sometimes called a system or superuser account, which provides powerful capabilities that are not made available to regular database accounts and users.'

DBA-privileged commands include commands for granting and revoking privileges to individual accounts, users, or user groups and for performing the following types of actions:

1. Account creation: This action creates a new account and password for a user or a group of users to enable access to the DBMS.

2. Privilege granting: This action permits the DBA to grant certain privileges to certain accounts.

3. Privilege revocation: This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.

4. Security level assignment: This action consists of assigning user accounts to the appropriate security classification level.

The DBA is responsible for the overall security of the database system. Action 1 in the preceding list is used to control access to the DBMS as a whole, whereas actions 2 and 3 are used to control discretionary database authorization, and action 4 is used to control mandatory authorization.

Access Protection, User Accounts, and Database Audits

Whenever a person or a group of persons needs to access a database system, the individual

or group must first apply for a user account. The DBA will then create a new account number and password for the user if there is a legitimate need to access the database.

The user must log in to the DBMS by entering the account number and password whenever database access is needed. The DBMS checks that the account number and password are valid; if they are, the user is permitted to use the DBMS and to access the database.

Application programs can also be considered as users and can be required to supply passwords. It is straightforward to keep track of database users and their accounts and passwords by creating an encrypted table or file with the two fields AccountNumber and Password.

This table can easily be maintained by the DBMS. Whenever a new account is created, a new record is inserted into the table. When an account is canceled, the corresponding record must be deleted from the table.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each login session, which consists of the sequence of database interactions that a user performs from the time of logging in to the time of logging off.

When a user logs in, the DBMS can record the user's account number and associate it with the terminal from which the user logged in.

All operations applied from that terminal are attributed to the user's account until the user logs off. It is particularly important to keep track of update operations that are applied to the database so that, if the database is tampered with, the DBA can find out which user did the tampering.

To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify the system log. the system log includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.

We can expand the log entries so that they also include the account number of the user and the online terminal to that applied each operation recorded in the log.

If any tampering with the database is suspected, a database audit is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.

Security issue based on granting/revoking of privileges

Discretionary access control (also called security scheme) is based on the concept of access rights (also called privileges) and mechanism for giving users such privileges. It grants the privileges (access rights) to users on different objects, including the capability to access specific data files, records or fields in a specified mode, such as, read, insert, delete or update or combination of these. A user who creates a database object such as a table or a view automatically gets all applicable privilege on that object. The DBMS keeps track of how these privileges are granted to other users. Discretionary security schemes are very flexible. However, it has certain weaknesses, for example, a devious unauthorised user can trick an authorised user into disclosing sensitive data.

Granting /Revoking of privileges

Granting and revoking privileges to the users is the responsibility of database administrator (DBA) of the DBMS. DBA classifies users and data in accordance with the policy of the organisation. DBA privileged commands include commands for granting and revoking privileges to individual accounts, users or user groups. It performs the following types of actions:

(a) *Account creation:* Account creation action creates a new account and password for a user or a group of users to enable them to access the DBMS.

(b) *Privilege granting:* Privilege granting action permits the DBA to grant certain privileges (access rights) to certain accounts.

(c) *Privilege revocation:* Privilege revoking action permits the DBA to revoke (cancel) certain privileges (access rights) that were previously given to certain accounts.

(d) *Security level assignment:* Security level assignment action consists of assigning user accounts to the appropriate security classification level.

Having an account and a password do not necessarily entitle a user or user groups to access all the functions of the DBMS. Generally, following two levels of privilege assignment is done to access the database system:

(a) ***The account level privilege assignment:*** At the account level privilege assignment, the DBA specifies the particular privileges that each account holds independently of the relations in the database. The account level privileges apply to the capabilities provided to the account itself and can include the following in SQL:

CREATE SCHEMA privilege	: to create a schema
CREATE TABLE privilege	: to create a table
CREATE VIEW privilege	: to apply schema changes such as
ALTER privilege	: adding or removing attributes from relations
DROP privilege	: to delete relation or views
MODIFY privilege	: to delete, insert, or update Tuples
SELECT privilege	: to retrieve information from the database using SELECT query

(b) ***The relation (or table) level privilege assignment:*** At relation or table level of privilege assignment, the DBA controls the privilege to access each individual relation or view in the database. Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual attributes (columns) of relations. Granting and revoking of relation privileges is controlled by assigning an owner account for each relation *R* in a database. The owner account is typically the account that was used when the relation was first created. The owner of the relation is given all privileges on the relation. In SQL, the following types of privileges can be granted on each individual relation *R*:

SELECT privilege on *R*	: to read or retrieve tuples from *R*
MODIFY privileges on *R*	: to modify (UPDATE, INSERT and DELETE) tuples of *R*
REFERENCES privilege on *R*	: to reference relationship *R*

GRANT {ALL | privilege-list}
ON {table-name [(column-comma-list)] | view-name [(column-comma-list)]}
TO {PUBLIC | user-list}
[WITH GRANT OPTION]

Meaning of the various clauses is as follows:

ALL	All the privileges for the object for which the user issuing the GRANT has grant authority, is granted.
privilege-list	Only the listed privileges are granted.
ON	It specifies the object on which the privileges are granted. It can be a table or a view.
column-comma-list	The privileges are restricted to the specified columns. If this is not specified, the grant is given for the entire table/view.
TO	It is used to identify the users to whom the privileges are granted.
PUBLIC	It means that the privileges are granted to all known users of the system who has valid User ID and Password.
user-list	The privileges will be granted to the user(s) specified in the list.
WITH GRANT OPTION	It means that the recipient has the authority to grant the privileges that were granted to him to another user.

Some of the examples of granting privileges are given below.

```
GRANT SELECT
ON EMPLOYEE
TO ABHISHEK, MATHEW
```

This means that the users 'ABHISHEK' and 'MATHEW' are authorised to perform SELECT operations on the table (or relation) EMPLOYEE.

```
GRANT SELECT
ON EMPLOYEE
```

15. What is statistical database? Explain with example. Also discuss the various security problems raised in these databases. How can we prevent them?

Ans: A statistical database contains confidential information about individuals or events. These databases are mainly used to generate statistical information about the stored information. Such databases accept only **statistical queries**, which involve statistical functions such as `SUM`, `AVG`, `COUNT`, `MIN`, `MAX` and so on. However, users are not allowed to retrieve information about a particular individual.

Consider a relation `BankEmp` with the attributes `ECode`, `EName`, `Sex`, `State`, `Salary`, `Branch` and `Designation`. Using statistical queries, one can retrieve the number of clerks, maximum salary, average salary of clerks and so on. However, users are not allowed to retrieve the salary of a particular employee.

Implementing security in such a database raises new problems because it is possible to infer the information about specific individuals from a sequence of statistical queries. For example, suppose that the user is interested in retrieving the salary of *John Macmillan* living in *New York*, who is a clerk in *Los Angeles* branch. User issues a statistical query to count the number of clerks in *Los Angeles* branch who are living in *New York*. The query can be defined a

```
SELECT COUNT (*) FROM BankEmp WHERE (State = 'New York' AND Sex
= 'M' AND Designation = 'Clerk' AND Branch = 'Los Angeles');
```

If the result of this query is 1, the next statistical query can be issued with the same condition to find the `Salary` of John Macmillan. The query can be defined a

```
SELECT SUM (Salary) FROM BankEmp WHERE (State = 'New York' AND
Sex = 'M' AND Designation = 'Clerk' AND Branch = 'Los Angeles');
```

Even if the result of the first query is not 1 but a small value, say 4 or 5, the approximate salary of John Macmillan could be inferred by applying `MAX`, `MIN`, and `AVG` functions in statistical queries.

One possible approach to prevent the chances of inferring information about individual is to reject certain statistical queries. For example, if the number of tuples specified by selection condition falls below a particular number, say `N`, the statistical query can be rejected. Alternatively, by rejecting the sequence of statistical queries from the same user that refers to the same set of tuples, it is possible to prevent retrieval of individual information. Another approach is database partitioning, in which records are classified and stored in small size groups. In this case, any statistical query can refer to complete group or set of groups, but the query referring to subset of any group is rejected.

PL/SQL Security – Locks – types and levels of locks, Implicit locking, explicit locking.

1. Oracle is a multi-user system and several users may access a table at the same time for either viewing it or for manipulating it through INSERT, UPDATE and DELETE commands.
2. In order to maintain the integrity of the database, Oracle uses a technique of concurrency control through the mechanism of **locking**.
3. Thus, *locks are mechanisms to ensure data integrity while allowing maximum concurrent access to data*.
4. This locking mechanism is fully automatic and does not require user intervention.
5. The Oracle engine automatically locks table data while executing SQL statements. This type of locking is called **Implicit locking**.

Implicit Locking:
Implicit Locking is Oracle's default locking strategy. Oracle has to decide on two issues:

1. Types of locks to be applied,
2. Level of lock to be applied.

Types of Locks:
The type of lock to be placed on a resource depends on the operation being performed on that resource. The two types of operations on tables are:

- Read Operations (SELECT statement)
- Write Operations (INSERT, UPDATE, DELETE statements)

Read operations do not change the underlying table and hence simultaneous read operations can be performed on a table without causing any damage to the table data. Oracle engine places a **Shared** lock on a table when its data is being viewed.

Write operations cause a change in table data due to the INSERT, UPDATE and DELETE commands. Hence, simultaneous write operations can adversely affect the table integrity. Simultaneous write operations will cause loss of data consistency in the table. Therefore, Oracle places an **Exclusive** lock on a table or parts of a table when data is being written in a table.

The rules for locking are as follows:

- Data that is being changed cannot be read (write and read cannot be performed concurrently)
- Writers wait for other writers if they attempt to update the same rows at the same time.

Types of Locks supported by Oracle are:
Shared Locks:

- This lock is placed on a resource whenever a read operation is being performed.
- Multiple shared locks can be placed simultaneously on a resource.

Exclusive Locks:

- This lock is placed on a resource whenever a write operation (INSERT/UPDATE/DELETE) is being performed.
- Only one exclusive lock can be placed on a resource at a time. The first user who acquires an exclusive lock will continue to be the sole owner of the resource and no other user can acquire an exclusive lock on that resource.

Levels of Locks:
A table can be decomposed into rows and a row can be decomposed into fields. So we can design a locking system which can lock the individual fields of a record. Thus, more than one user can work on a single record in a table – i.e., each user can be working on a *different field* of the *same record*, at the *same* time. However, Oracle does not provide a field level lock.

The three levels of locking provided by Oracle are:
- Row level
- Page level
- Table level

The level of locking to be used is decided by the Oracle engine; this depends on the presence or absence of the WHERE condition in the SQL statement. The rules are as follows:
- If the WHERE clause evaluates to only a **single row** in the table, a **row level** lock is used.
- If the WHERE clause evaluates to a **set of data**, a **page level** lock is used.
- If there is no WHERE clause (i.e., the **entire table** is being accessed by the query), a **table level** lock is used.

Exclusive Locks:
- This lock is placed on a resource whenever a write operation (INSERT/UPDATE/DELETE) is being performed.
- Only one exclusive lock can be placed on a resource at a time. The first user who acquires an exclusive lock will continue to be the sole owner of the resource and no other user can acquire an exclusive lock on that resource.

Need for exclusive locking:
Although the Oracle engine has a default locking strategy, explicit locking is often needed. The following example explains the need for explicit locking:
Consider two client computers (Client A and Client B) are entering sales orders. Each time a sales order is prepared, the QOH of the product must be updated in the PRODUCTS_MSTR table. If client A fires an UPDATE command on a record of the PRODUCTS_MSTR table, then Oracle will implicitly lock the record so that no further data manipulation can be done by any other user till the lock is **released**. The lock will be released only when Client A fires a COMMIT or a ROLLBACK command.

In the mean time, if Client B tries to view the same record, the Oracle engine will display the old data for the record as the transaction for that record has not been completed by Client A. Thus, Client B is viewing wrong information.

In such a situation, Client A must explicitly lock the record so that no other user can access the record even for viewing purposes till client A completes his transaction.

Such a lock that is *applied by the user* on a row, page or the entire table is called an **explicit lock**. Explicit locking is user-defined strategy and it always overrides Oracle's default locking strategy.

Tables and rows can be explicitly locked by using either the SELECT…FOR UPDATE statement or the LOCK TABLE statement.

The SELECT…FOR UPDATE statement:
This statement is used for acquiring exclusive row-level locks. This statement informs the Oracle engine that data currently being used needs to be updated. It is followed by UPDATE statements with a WHERE clause.

Example 1:
Two clients A and B are recording the transactions performed in a bank for a particular account number simultaneously.
Client A fires the statement:
Client A> SELECT * FROM ACCT_MSTR WHERE ACCT_NO = 'SB7' FOR UPDATE;

When the above SELECT statement is fired, Oracle's engine locks the record SB7. This lock will be released only when Client A issues the COMMIT or ROLLBACK command.

Now Client B fires a SELECT statement which points to record SB7 which has already been locked by Client A.
Client B> SELECT * FROM ACCT_MSTR WHERE ACCT_NO = 'SB7' FOR UPDATE;

The Oracle engine ensures that Client B's SQL statement waits for lock to be released on ACCT_MSTR by a COMMIT or ROLLBACK statement. In order to avoid unnecessary waiting time, a NOWAIT option can be used to inform Oracle engine to terminate the SQL command if the record has already been locked. In this case Oracle engine generates a message for Client B stating that the resource is busy. So in order not to wait for an uncertain period, Client B can issue the following query:

Client B> SELECT * FROM ACCT_MSTR WHERE ACCT_NO = 'SB7' FOR UPDATE **NOWAIT**;

The SELCT… FOR UPDATE clause cannot be used with the following:

- DISTINCT and GROUP BY clauses.
- SET OPERATORS and GROUP functions.

Using the LOCK TABLE statement:
Oracle's default locking strategy can be overridden by creating a lock in a specific mode. This is done with the LOCK TABLE clause:

LOCK TABLE <tablename> [,<tablename>]…
In {ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE }
[NOWAIT]

In this syntax:

- <tablename> indicates the names of the tables / views to be locked.
- IN decides what other locks on the same resource can exist simultaneously. E.g., if there is an exclusive lock on the table, no other user can update rows on the table. It can have the following values:
  - EXCLUSIVE – Query is allowed on the locked resource but other activities are not permitted
  - SHARE – Queries are allowed on the table but does not allow updates on the table
  - ROW EXCLUSIVE – this lock is acquired when using the commands UPDATE, INSERT, or DELETE.
  - SHARE EXCLUSIVE – they are used to look at the whole table, to carry out selective updates, and to allow other users to look at rows in the table but not lock the table or update the rows.