Unit 6

DBMS

BTCS 501-18

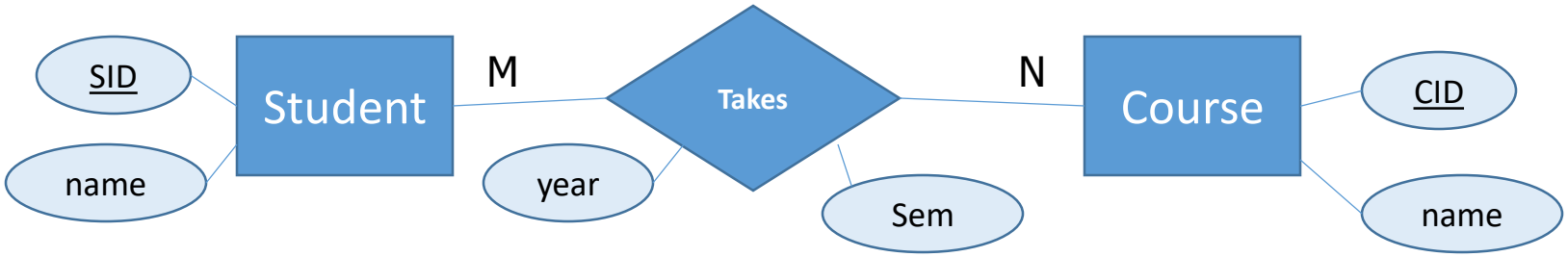
- OODBMS

Overview

- Relational quick review
- Why OO?
- What is an Object?
- What is an object ID?
- Classes, subclasses and inheritance
- UML
- OQL

Relational DBMS Quick Review

- Data model: ER.



- Data is stored in tables.
- Each row is a record.
- Relationships between tables (PK-FK).

SID	name
424112211	Nora M
424221122	Sara S
424331133	Hala L

SID	CID	year	Sem
424112211	CAP364	1425	1
424112211	CAP430	1426	2
424331133	CAP364	1426	1

CID	name
CAP364	DB2
CAP430	Security

Weaknesses of RDBMS

- Poor Representation of “Real World” Entities.
- Poor Support for Integrity and Enterprise Constraints.
- Homogeneous Data Structure.
- Limited Operations.
- Difficulty Handling Recursive Queries.
- Schema changes are difficult.
- RDBMSs are poor at navigational access.

Why OO? Advanced DB Applications

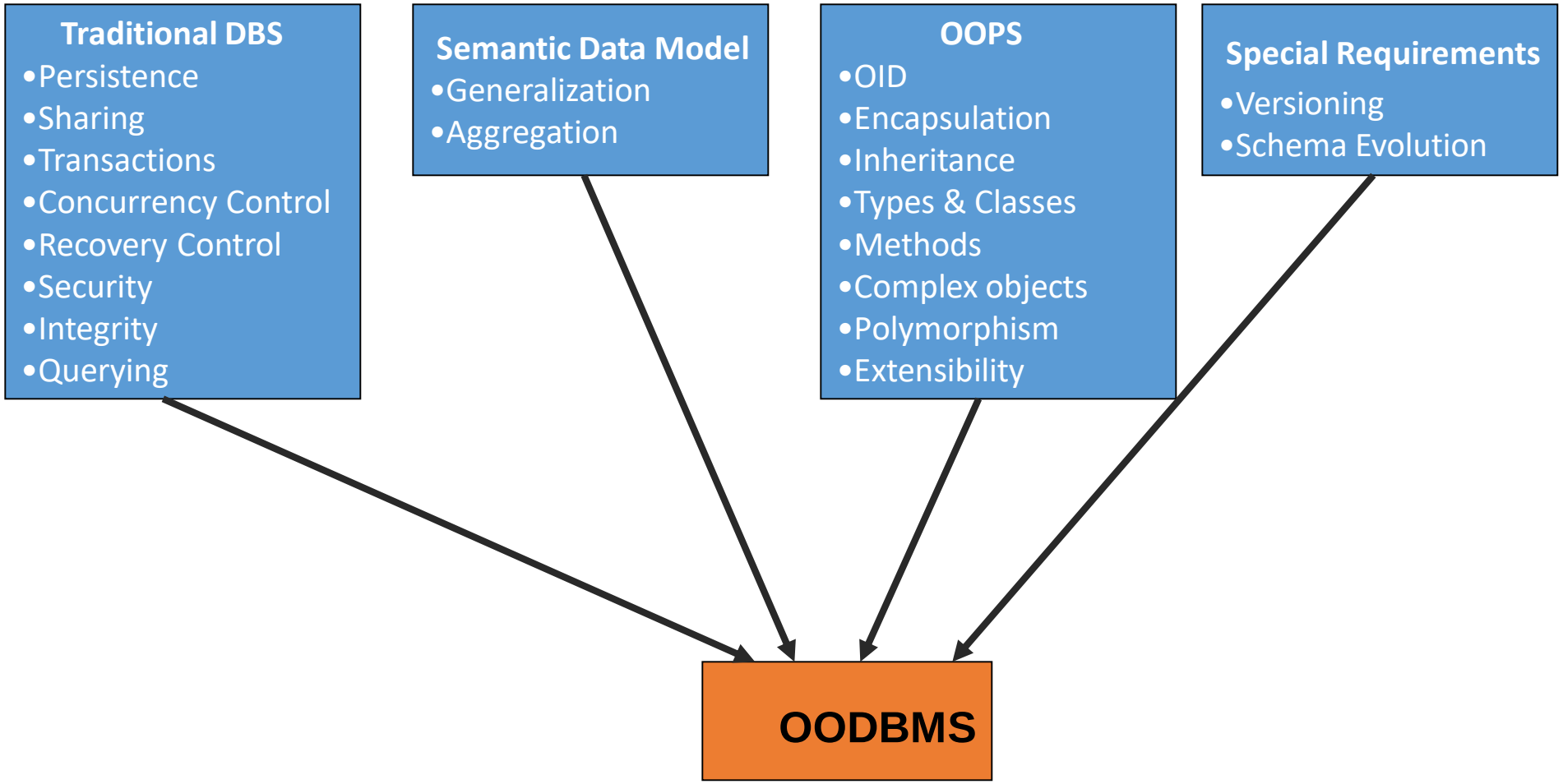
- Computer-Aided Design (CAD).
- Computer-Aided Manufacturing (CAM).
- Computer-Aided Software Engineering (CASE).
- Network Management Systems.
- Office Information Systems (OIS) and Multimedia Systems.
- Digital Publishing.
- Geographic Information Systems (GIS).
- Interactive and Dynamic Web sites.
- Other applications with complex and interrelated objects and procedural data – Read pages 805-808 in book.

What is an OODBMS anyway?

- OODBMS (Object-oriented DB Management System) is a database with data stored in *objects* and collections NOT rows and tables.

- OO Concepts:
  - Abstraction, encapsulation, information hiding.
  - Objects and attributes.
  - Object identity.
  - Methods and messages.
  - Classes, subclasses, superclasses, and inheritance.
  - Overloading.
  - Polymorphism and dynamic binding.

OODBMS


Traditional DBS
•Persistence
•Sharing
•Transactions
•Concurrency Control
•Recovery Control
•Security
•Integrity
•Querying
Semantic Data Model
•Generalization
•Aggregation
OOPS
•OID
•Encapsulation
•Inheritance
•Types & Classes
•Methods
•Complex objects
•Polymorphism
•Extensibility
Special Requirements
•Versioning
•Schema Evolution
OODBMS

Object

- Uniquely identifiable entity that contains both
  - the *attributes* that describe the state of a real-world object

  and
  - the *actions* associated with it.

- Object encapsulates both *state* and *behavior*; an entity only models *state*.

- Persistent objects vs. transient objects.
- *Everything* in an OODBMS is an object.

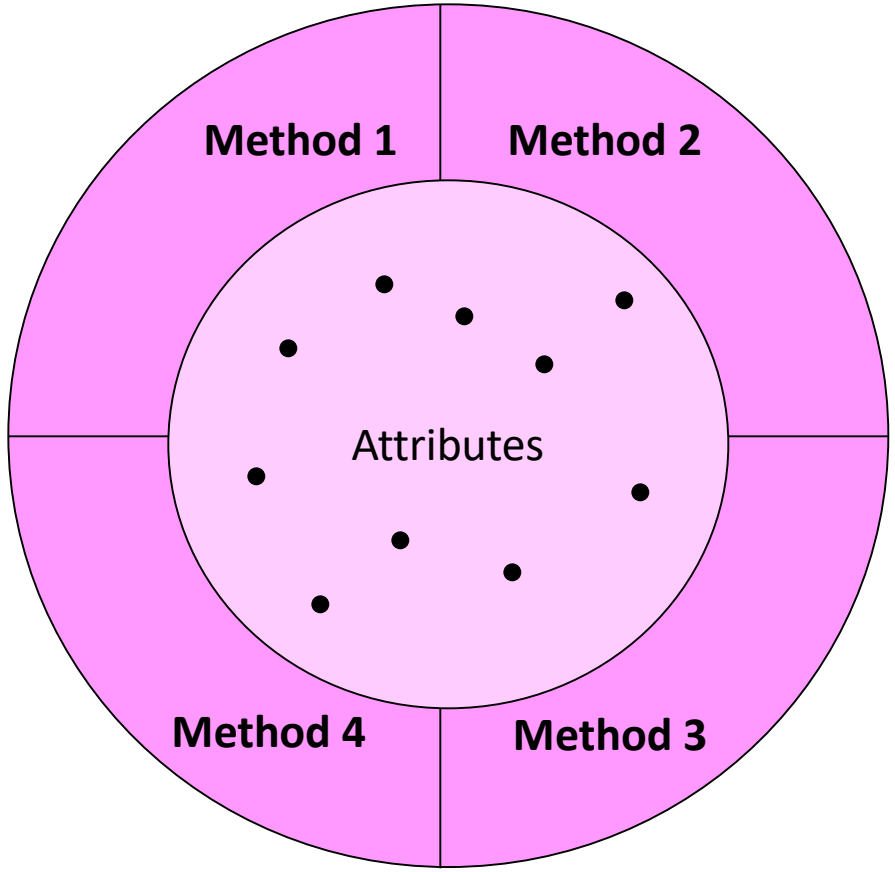
Object Identity (OID)

- Object identifier (OID) assigned to object when it is created that is:

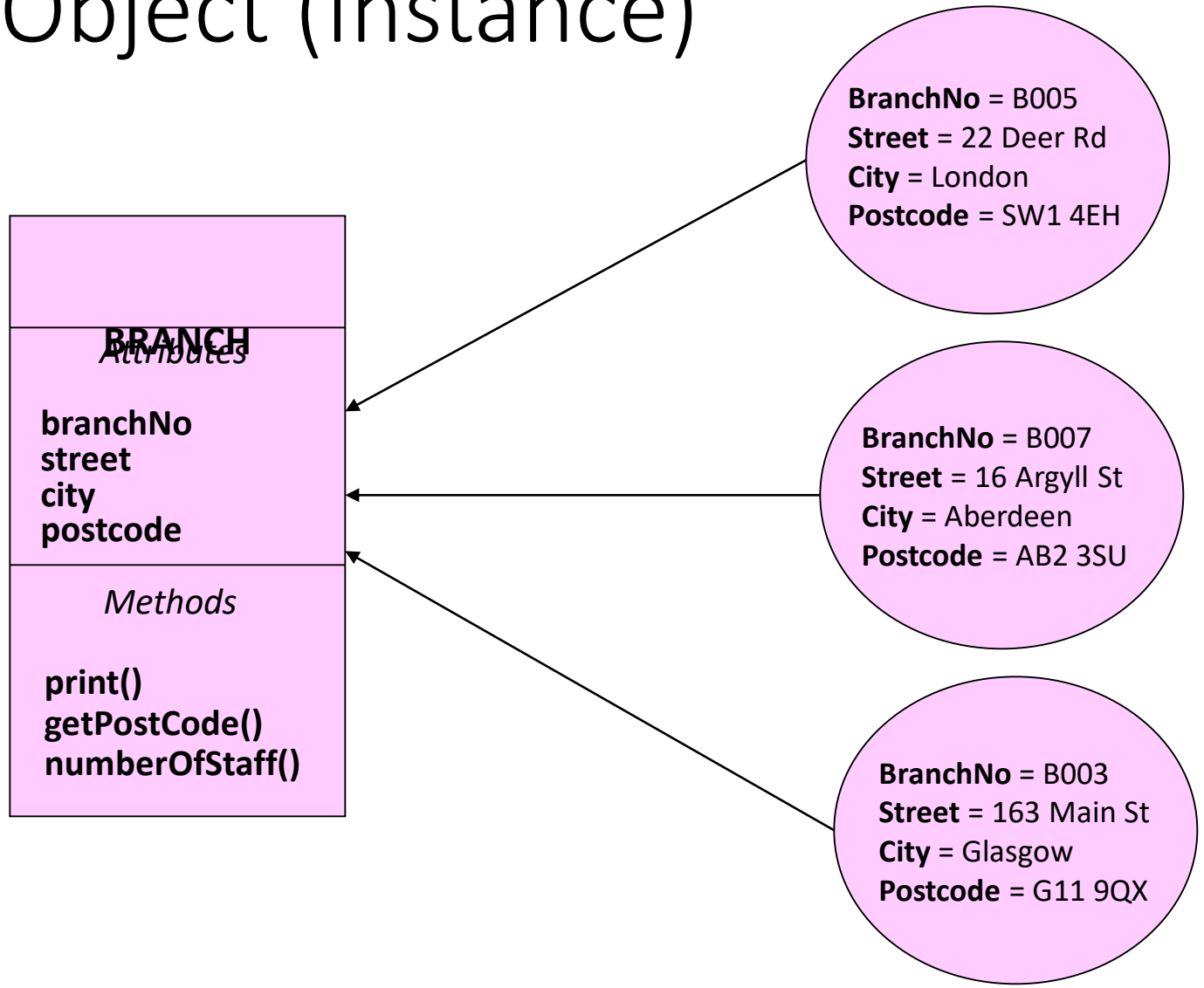
  - System-generated.
  - Unique to that object.
  - Invariant.
  - Independent of the values of its attributes (that is, its state).
  - Invisible to the user (ideally).

Encapsulation

- Object



Class and Object (Instance)

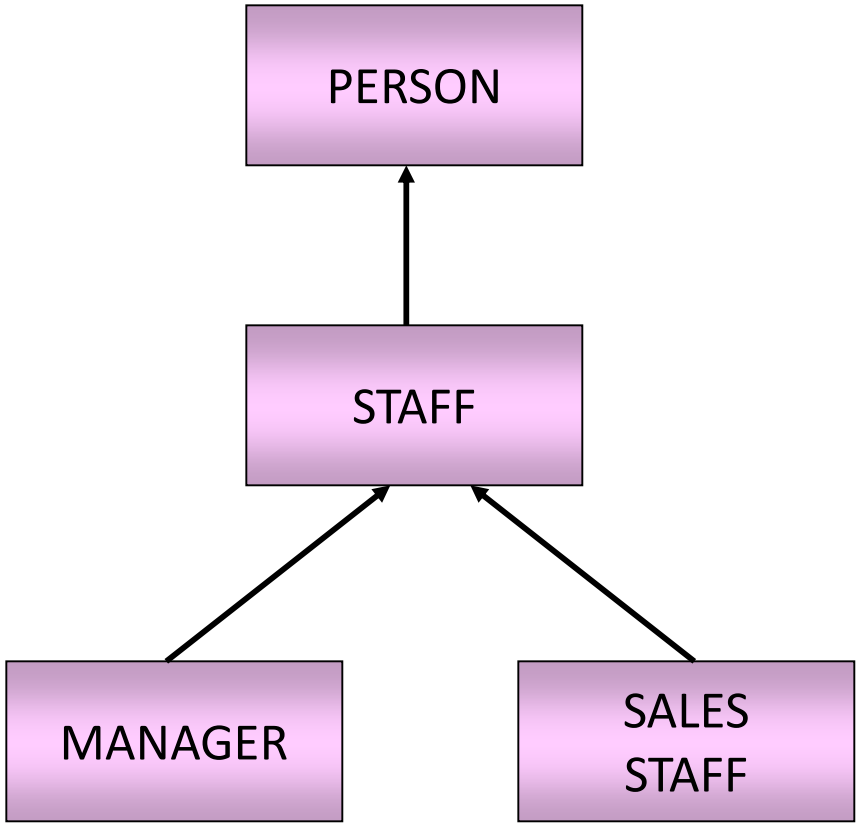


Subclasses, Superclasses and Inheritance

Inheritance allows one class of objects to be defined as a special case of a more general class.

- **Special cases are *subclasses* and more general cases are *superclasses*.**
- **Process of forming a superclass is *generalization*; forming a subclass is *specialization*.**
- **Subclass inherits all properties of its superclass and can define its own unique properties.**
- **Subclass can redefine inherited methods (override).**
- **All instances of subclass are also instances of superclass.**

Single Inheritance



Complex Objects

- A Complex object is something that can be viewed as a single thing in the real world but it actually consists of many sub-objects.

- 2 Types:
  - Unstructured.
    - Their structure hard to determine.
    - Requires a large amount of storage.
    - BLOB (Binary Large Objects): images & long test strings.
  - Structured.
    - Clear structure.
    - Sub-objects in a part-of relationship.
    - Will talk more about in next slide.

Object Query Language -- OQL

- A Query language for OODBMS.
- OQL can be used for both associative and navigational access:
  - Associative query returns collection of objects (like SQL).
  - Navigational query accesses individual objects and object relationships used to navigate from one object to another.
  - RDBMS
  - OODBMS
  - ORDBMS=RDBMS+OODBMS-SQL3 SQL4

OQL vs. SQL: A Simple Example

Query:
List the names of the children of employees working in the sales department.

OQL

```
select
   c.fname, c.lname
from
   Depts d, d.employs e,
   e.hasChildren c
where
   d.name = “Sales”
```

SQL

```
select
   c.fname, c.lname
from
   Depts d, Employee e,
   Children c
where
   d.name = “Sales” and
   d.deptID = e.deptID and
   c.parentID = e.empID
```

Commercial OODBMs

- GemStone from Gemstone Systems Inc.,
- Itasca from Ibex Knowledge Systems SA,
- Objectivity/DB from Objectivity Inc.,
- ObjectStore from eXcelon Corp.,
- Ontos from Ontos Inc.,
- Poet from Poet Software Corp.,
- Jasmine from Computer Associates/Fujitsu,
- Versant from Versant Object Technology.

Imp questions

- Disscuss oodbms
- RDBMS VS oodbms
- Rdbms vs oodbms vs ordbmms

ORDBMS vs. ORDBMS

- similarities: both support user-defined ADTs, constructed types, reference types,
  object identity, query language
- differences: ORDBMSs add new data types to RDBMS; OODBMSs add
- DBMS functionalities to a programming language

- Integration with host language
  - OODBMS: seamless integration with C++/Small talk
  - ORDBMS: integration is only through embedded SQL in a host language

Continued…

- Application requirement
  - OODBMS:
    - few large objects fetched occasionally: few disk I/O
    - long duration transactions on in-memory objects
    - ability to cache objects in memory
  - ORDBMS:
    - large collection of data
    - extensive disk I/O
    - short transactions

Continued..

- Query language
  - OODBMS:
    - Query processing is relatively inefficient
    - No standard available
  - ORDBMS:
    - Query facilities is the centerpiece
    - SQL-based standards available: SQL3, SQL4

RDBMS VS ORDBMS

RDBMS VS OODBMS

Basis of Difference	OODBMS	RDBMS
Primary Objective	Data encapsulation and Data Independence	Data Independence from application programs
Reorganization	Classes can be reorganized without affecting the mode of using them	Data can be reorganized without affecting the mode of using them
Storage	Data and Methods	Only Data
Encapsulation	Data accessible through class methods only	No such provision
Adhesion	Data can be chained	No such provision
Representation	Class of objects directly model the real life applications	Application will have to be converted in accordance with the relational tables
Data access speed	Access to data is faster as no joins are needed	Joins are needed so data access is slower

Comparison

Comparison

Criteria	RDBMS	ODBMS	ORDBMS
Defining standard	SQL2	ODMG-2.0	SQL3 (in process)
Support for object-oriented features	Does not support; It is difficult to map program object to the database	Supports extensively	Limited support; mostly to new data type
Usage	Easy to use	OK for programmers; some SQL access for end users	Easy to use except for some extensions
Support for complex relationships	Does not support abstract datatypes	Supports a wide variety of datatypes and data with complex inter-relationships	Supports Abstract datatypes and complex relationships
Performance	Very good performance	Relatively less performance	Expected to perform very well

Dbms vs rdbms

Difference between DBMS and RDBMS

Although DBMS and RDBMS both are used to store information in physical database but there are some remarkable differences between them.

The main differences between DBMS and RDBMS are given below:

No.	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	**Normalization is not** present in DBMS.	**Normalization is** present in RDBMS.
4)	DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
8)	DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.
9)	Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc.

After observing the differences between DBMS and RDBMS, you can say that RDBMS is an extension of DBMS. There are many software products in the market today who are compatible for both DBMS and RDBMS. Means today a RDBMS application is DBMS application and vice-versa.

Difference..

OODBMS vs ORDBMS

- ORDBMS vs. ORDBMS
  - similarities: both support user-defined ADTs, constructed types, reference types, object identity, query language
  - differences: ORDBMSs add new data types to RDBMS; OODBMSs add DBMS functionalities to a programming language
- Integration with host language
  - OODBMS: seamless integration with C++/Small talk
  - ORDBMS: integration is only through embedded SQL in a host language
- Application requirement
  - OODBMS:
    - few large objects fetched occasionally: few disk I/O
    - long duration transactions on in-memory objects
    - ability to cache objects in memory
  - ORDBMS:
    - large collection of data
    - extensive disk I/O
    - short transactions
- Query language
  - OODBMS:
    - Query processing is relatively inefficient
    - No standard available
  - ORDBMS:
    - Query facilities is the centerpiece
    - SQL-based standards available: SQL3, SQL4

Distributed Database

1. Distributed Database Concepts
2. Data Fragmentation, Replication and Allocation
3. Types of Distributed Database Systems
4. Query Processing
5. Concurrency Control and Recovery
6. 3-Tier Client-Server Architecture

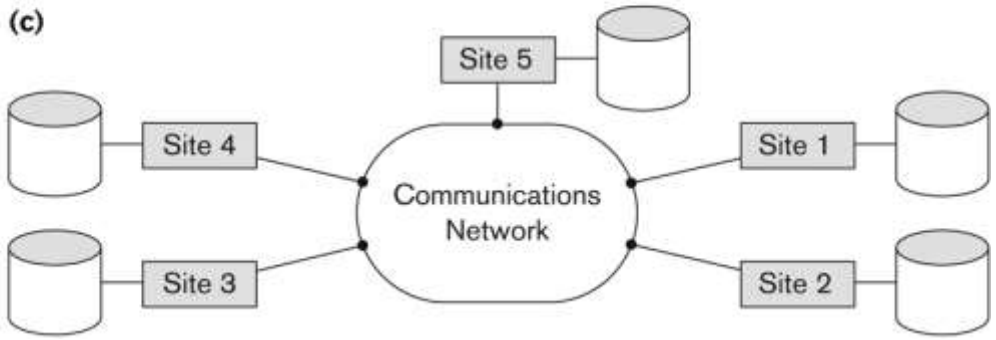
Distributed Database Concepts

- A transaction can be executed by multiple networked computers in a unified manner.
- A **distributed database (DDB)** processes Unit of execution (a transaction) in a distributed manner. A distributed database (DDB) can be defined as
  - A distributed database (DDB) is a collection of multiple logically related database distributed over a computer network, and a distributed database management system as a software system that manages a distributed database while making the distribution transparent to the user.

Distributed Database System

- Advantages
  - Management of distributed data with different **levels of transparency**:
    - This refers to the physical placement of data (files, relations, etc.) which is not known to the user (distribution transparency).

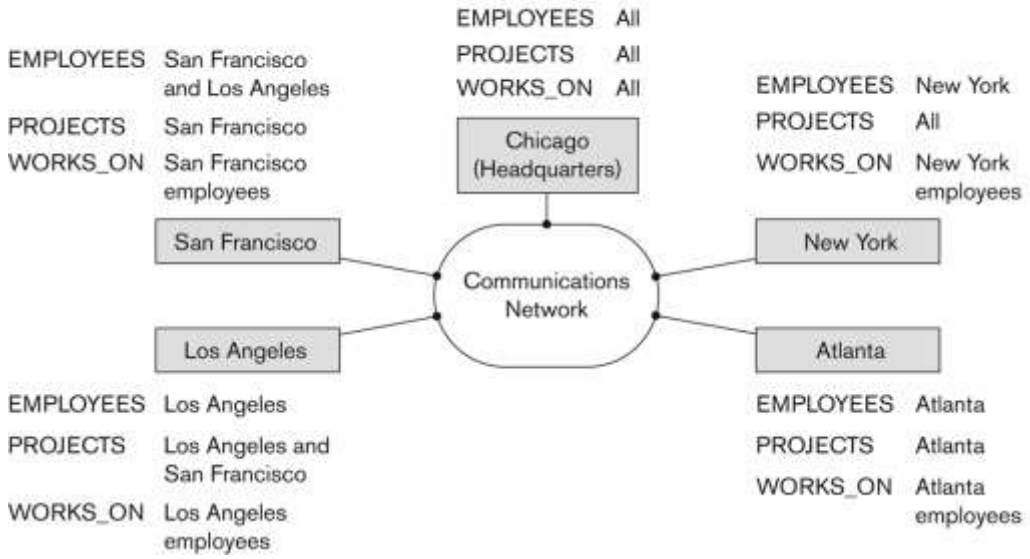
Figure 25.1
Some different database system architectures. (a) Shared nothing architecture. (b) A networked architecture with a centralized database at one of the sites. (c) A truly distributed database architecture.



Distributed Database System

- Advantages (transparency, contd.)
  - The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown below.

Figure 25.2
Data distribution and replication among distributed databases.



Distributed Database System

- Advantages (transparency, contd.)
  - **Distribution and Network transparency**:
    - Users do not have to worry about operational details of the network.
      - There is Location transparency, which refers to freedom of issuing command from any location without affecting its working.
      - Then there is Naming transparency, which allows access to any names object (files, relations, etc.) from any location.

Distributed Database System

- Advantages (transparency, contd.)
  - **Replication transparency**:
    - It allows to store copies of a data at multiple sites as shown in the above diagram.
    - This is done to minimize access time to the required data.
  - **Fragmentation transparency**:
    - Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation).

Distributed Database System

- Other Advantages
  - **Increased reliability and availability**:
    - Reliability refers to system live time, that is, system is running efficiently most of the time. Availability is the probability that the system is continuously available (usable or accessible) during a time interval.
    - A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.

Distributed Database System

- Other Advantages (contd.)
  - **Improved performance**:
    - A distributed DBMS fragments the database to keep data closer to where it is needed most.
    - This reduces data management (access and modification) time significantly.
  - **Easier expansion (scalability)**:
    - Allows new nodes (computers) to be added anytime without chaining the entire configuration.

Data Fragmentation, Replication and Allocation

- **Data Fragmentation**
  - Split a relation into logically related and correct parts. A relation can be fragmented in two ways:
    - **Horizontal Fragmentation**
    - **Vertical Fragmentation**

Data Fragmentation, Replication and Allocation

- **Horizontal fragmentation**
  - It is a horizontal subset of a relation which contain those of tuples which satisfy selection conditions.
  - Consider the Employee relation with selection condition (DNO = 5). All tuples satisfy this condition will create a subset which will be a horizontal fragment of Employee relation.
  - A selection condition may be composed of several conditions connected by AND or OR.
  - Derived horizontal fragmentation: It is the partitioning of a primary relation to other secondary relations which are related with Foreign keys.

Data Fragmentation, Replication and Allocation

- **Vertical fragmentation**
  - It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation.
  - Consider the Employee relation. A vertical fragment of can be created by keeping the values of Name, Bdate, Sex, and Address.
  - Because there is no condition for creating a vertical fragment, each fragment must include the primary key attribute of the parent relation Employee. In this way all vertical fragments of a relation are connected.

Data Fragmentation, Replication and Allocation

- **Representation**
  - **Horizontal fragmentation**
    - Each horizontal fragment on a relation can be specified by a σ_{Ci} (R) operation in the relational algebra.
    - Complete horizontal fragmentation
    - A set of horizontal fragments whose conditions C1, C2, …, Cn include all the tuples in R- that is, every tuple in R satisfies (C1 OR C2 OR … OR Cn).
    - Disjoint complete horizontal fragmentation: No tuple in R satisfies (Ci AND Cj) where $i \neq j$.
    - To reconstruct R from horizontal fragments a UNION is applied.

Data Fragmentation, Replication and Allocation

- **Representation**
  - **Vertical fragmentation**
    - A vertical fragment on a relation can be specified by a $\Pi_{Li}(R)$ operation in the relational algebra.
    - Complete vertical fragmentation
    - A set of vertical fragments whose projection lists L1, L2, …, Ln include all the attributes in R but share only the primary key of R. In this case the projection lists satisfy the following two conditions:
    - $L1 \cup L2 \cup ... \cup Ln = ATTRS(R)$
    - $Li \cap Lj = PK(R)$ for any i j, where ATTRS (R) is the set of attributes of R and PK(R) is the primary key of R.
    - To reconstruct R from complete vertical fragments a OUTER UNION is applied.

Data Fragmentation, Replication and Allocation

- **Representation**
  - **Mixed (Hybrid) fragmentation**
    - A combination of Vertical fragmentation and Horizontal fragmentation.
    - This is achieved by SELECT-PROJECT operations which is represented by $\Pi_{Li}(\sigma_{Ci}(R))$.
    - If C = True (Select all tuples) and L ≠ ATTRS(R), we get a vertical fragment, and if C ≠ True and L ≠ ATTRS(R), we get a mixed fragment.
    - If C = True and L = ATTRS(R), then R can be considered a fragment.

Data Fragmentation, Replication and Allocation

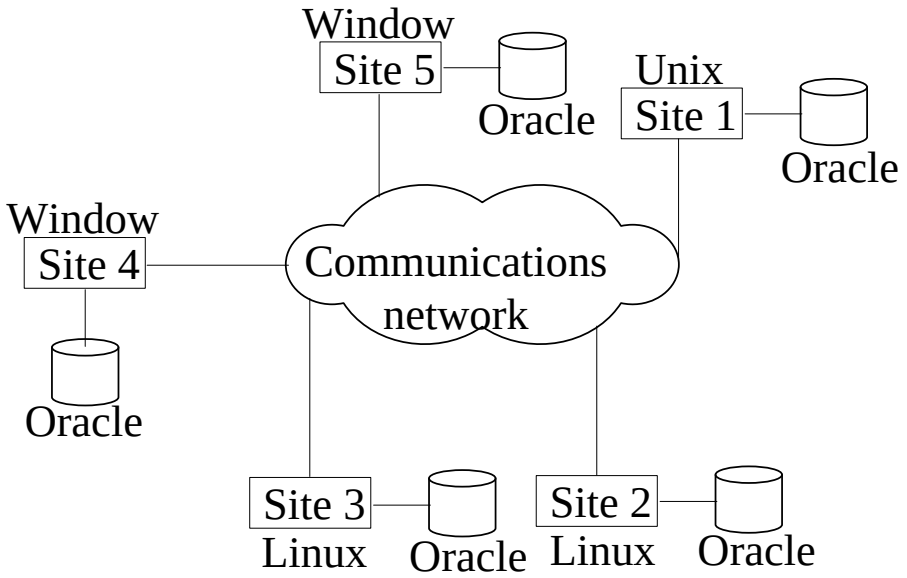
- **Fragmentation schema**
  - A definition of a set of fragments (horizontal or vertical or horizontal and vertical) that includes all attributes and tuples in the database that satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of UNION (or OUTER JOIN) and UNION operations.
- **Allocation schema**
  - It describes the distribution of fragments to sites of distributed databases. It can be fully or partially replicated or can be partitioned.

Data Fragmentation, Replication and Allocation

- **Data Replication**
  - Database is replicated to all sites.
  - In full replication the entire database is replicated and in partial replication some selected part is replicated to some of the sites.
  - Data replication is achieved through a replication schema.
- **Data Distribution (Data Allocation)**
  - This is relevant only in the case of partial replication or partition.
  - The selected portion of the database is distributed to the database sites.

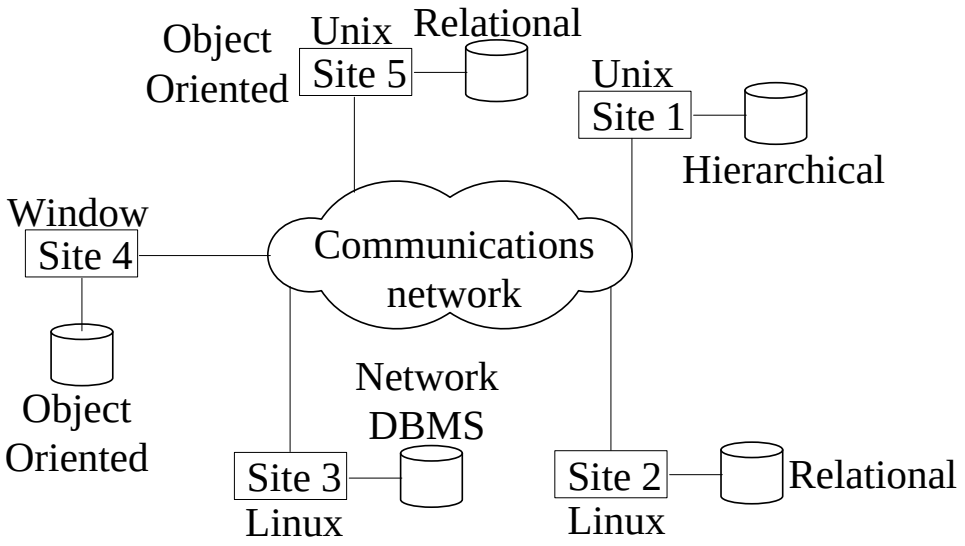
Types of Distributed Database Systems

- 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.



Types of Distributed Database Systems

- 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.



Types of Distributed Database Systems

- Federated Database Management Systems Issues
  - Differences in data models:
    - Relational, Objected oriented, hierarchical, network, etc.
  - Differences in constraints:
    - Each site may have their own data accessing and processing constraints.
  - Differences in query language:
    - Some site may use SQL, some may use SQL-89, some may use SQL-92, and so on.

Query Processing in Distributed Databases

- Issues
  - Cost of transferring data (files and results) over the network.
    - This cost is usually high so some optimization is necessary.
    - Example relations: Employee at site 1 and Department at Site 2
      - Employee at site 1. 10,000 rows. Row size = 100 bytes. Table size = 10^6 bytes.
      - Department at Site 2. 100 rows. Row size = 35 bytes. Table size = 3,500 bytes.

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Superssn	Dno

Dname	Dnumber	Mgrssn	Mgrstartdate

    - Q: For each employee, retrieve employee name and department name Where the employee works.
    - Q: $\Pi_{Fname,Lname,Dname}$ (Employee $\bowtie_{Dno = Dnumber}$ Department)

Query Processing in Distributed Databases

- Result
  - The result of this query will have 10,000 tuples, assuming that every employee is related to a department.
  - Suppose each result tuple is 40 bytes long. The query is submitted at site 3 and the result is sent to this site.
  - Problem: Employee and Department relations are not present at site 3.

Query Processing in Distributed Databases

- Strategies:
  1. Transfer Employee and Department to site 3.
     - Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
     - Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
  3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3.
     - Total bytes transferred = 400,000 + 3500 = 403,500 bytes.
- Optimization criteria: minimizing data transfer.

Query Processing in Distributed Databases

- Strategies:
  1. Transfer Employee and Department to site 3.
     - Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
     - Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
  3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3.
     - Total bytes transferred = 400,000 + 3500 = 403,500 bytes.
- Optimization criteria: minimizing data transfer.
  - Preferred approach: strategy 3.

Query Processing in Distributed Databases

- Consider the query
  - Q': For each department, retrieve the department name and the name of the department manager
- Relational Algebra expression:
  - $\Pi_{Fname,Lname,Dname}$ (Employee $\underset{Mgrssn = SSN}{\bowtie}$ Department)

Query Processing in Distributed Databases

- The result of this query will have 100 tuples, assuming that every department has a manager, the execution strategies are:
  1. Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 100 = 4000 bytes.
     - Total transfer size = 4000 + 1,000,000 = 1,004,000 bytes.
  3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 4000 + 3500 = 7500 bytes.

Query Processing in Distributed Databases

- The result of this query will have 100 tuples, assuming that every department has a manager, the execution strategies are:
  1. Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3500 = 1,003,500 bytes.
  2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 100 = 4000 bytes.
     - Total transfer size = 4000 + 1,000,000 = 1,004,000 bytes.
  3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 4000 + 3500 = 7500 bytes.
- Preferred strategy: Choose strategy 3.

Query Processing in Distributed Databases

- Now suppose the result site is 2. Possible strategies :
  1. Transfer Employee relation to site 2, execute the query and present the result to the user at site 2.
     - Total transfer size = 1,000,000 bytes for both queries Q and Q'.
  2. Transfer Department relation to site 1, execute join at site 1 and send the result back to site 2.
     - Total transfer size for Q = 400,000 + 3500 = 403,500 bytes and for Q' = 4000 + 3500 = 7500 bytes.

Query Processing in Distributed Databases

- Semijoin:
  - Objective is to reduce the number of tuples in a relation before transferring it to another site.

Concurrency Control and Recovery

- Distributed Databases encounter a number of concurrency control and recovery problems which are not present in centralized databases. Some of them are listed below.
  - Dealing with multiple copies of data items
  - Failure of individual sites
  - Communication link failure
  - Distributed commit
  - Distributed deadlock

Concurrency Control and Recovery

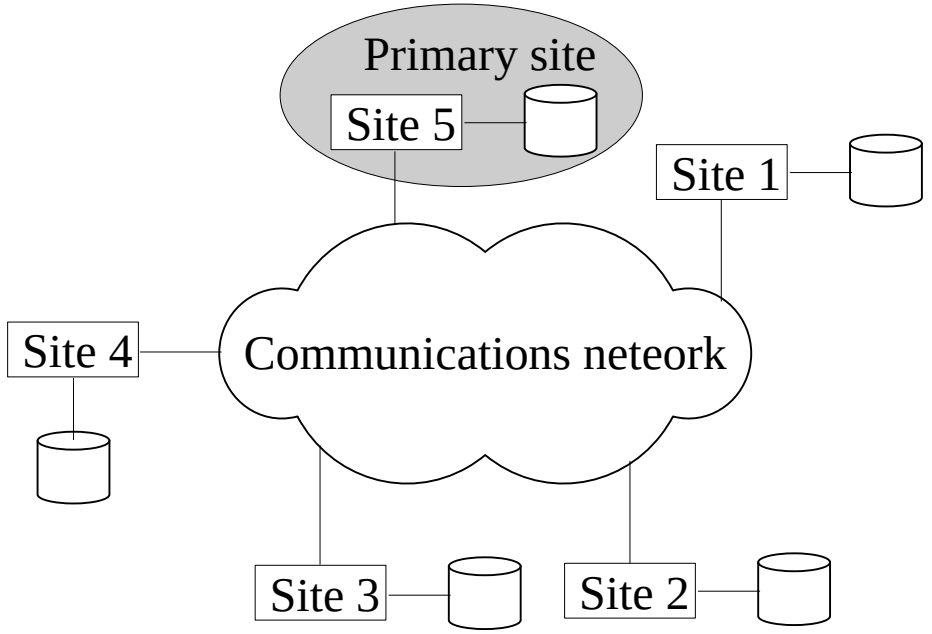
- Details
  - Dealing with multiple copies of data items:
    - The concurrency control must maintain global consistency. Likewise the recovery mechanism must recover all copies and maintain consistency after recovery.
  - Failure of individual sites:
    - Database availability must not be affected due to the failure of one or two sites and the recovery scheme must recover them before they are available for use.

Concurrency Control and Recovery

- Details (contd.)
  - Communication link failure:
    - This failure may create network partition which would affect database availability even though all database sites may be running.
  - Distributed commit:
    - A transaction may be fragmented and they may be executed by a number of sites. This require a two or three-phase commit approach for transaction commit.
  - Distributed deadlock:
    - Since transactions are processed at multiple sites, two or more sites may get involved in deadlock. This must be resolved in a distributed manner.

Concurrency Control and Recovery

- Distributed Concurrency control based on a distributed copy of a data item
  - Primary site technique: A single site is designated as a primary site which serves as a coordinator for transaction management.



Concurrency Control and Recovery

- Transaction management:
  - Concurrency control and commit are managed by this site.
  - In two phase locking, this site manages locking and releasing data items. If all transactions follow two-phase policy at all sites, then serializability is guaranteed.

Concurrency Control and Recovery

- Transaction Management
  - Advantages:
    - An extension to the centralized two phase locking so implementation and management is simple.
    - Data items are locked only at one site but they can be accessed at any site.
  - Disadvantages:
    - All transaction management activities go to primary site which is likely to overload the site.
    - If the primary site fails, the entire system is inaccessible.
  - To aid recovery a backup site is designated which behaves as a shadow of primary site. In case of primary site failure, backup site can act as primary site.

Concurrency Control and Recovery

- Primary Copy Technique:
  - In this approach, instead of a site, a data item partition is designated as primary copy. To lock a data item just the primary copy of the data item is locked.
- Advantages:
  - Since primary copies are distributed at various sites, a single site is not overloaded with locking and unlocking requests.
- Disadvantages:
  - Identification of a primary copy is complex. A distributed directory must be maintained, possibly at all sites.

Concurrency Control and Recovery

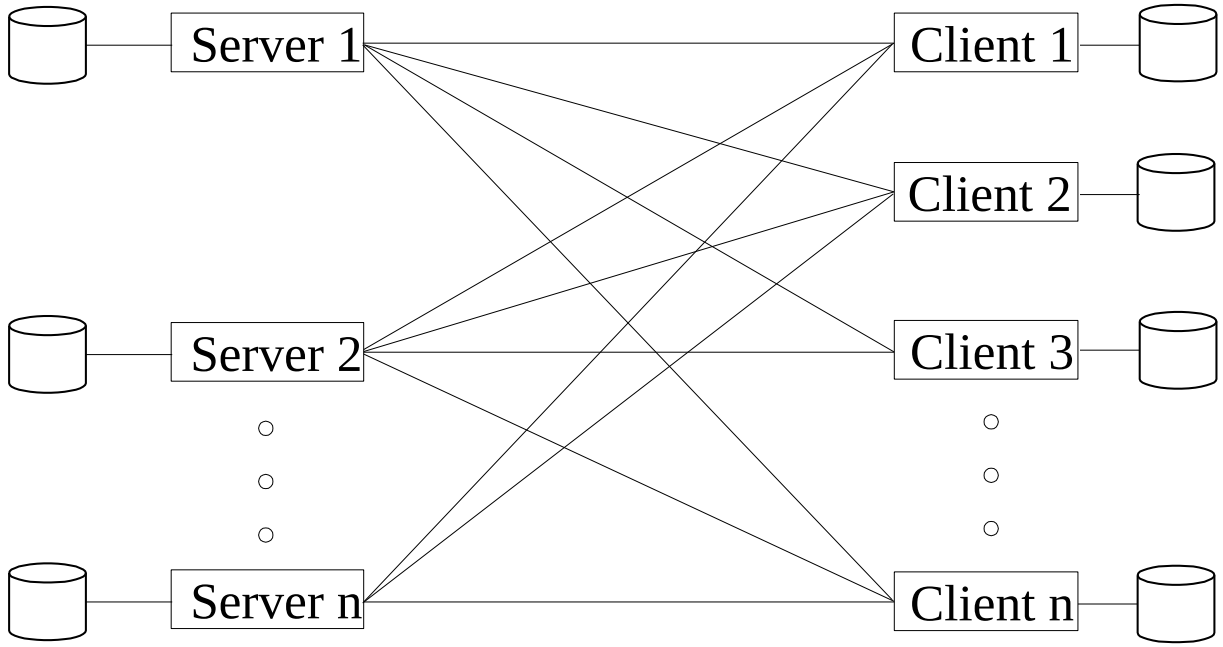
- Recovery from a coordinator failure
  - In both approaches a coordinator site or copy may become unavailable. This will require the selection of a new coordinator.
- Primary site approach with no backup site:
  - Aborts and restarts all active transactions at all sites. Elects a new coordinator and initiates transaction processing.
- Primary site approach with backup site:
  - Suspends all active transactions, designates the backup site as the primary site and identifies a new back up site. Primary site receives all transaction management information to resume processing.
- Primary and backup sites fail or no backup site:
  - Use election process to select a new coordinator site.

Concurrency Control and Recovery

- Concurrency control based on voting:
  - There is no primary copy of coordinator.
  - Send lock request to sites that have data item.
  - If majority of sites grant lock then the requesting transaction gets the data item.
  - Locking information (grant or denied) is sent to all these sites.
  - To avoid unacceptably long wait, a time-out period is defined. If the requesting transaction does not get any vote information then the transaction is aborted.

Client-Server Database Architecture

- It consists of clients running client software, a set of servers which provide all database functionalities and a reliable communication infrastructure.



Client-Server Database Architecture

- Clients reach server for desired service, but server does reach clients.
- The server software is responsible for local data management at a site, much like centralized DBMS software.
- The client software is responsible for most of the distribution function.
- The communication software manages communication among clients and servers.

Client-Server Database Architecture

- The processing of a SQL queries goes as follows:
  - Client parses a user query and decomposes it into a number of independent sub-queries. Each subquery is sent to appropriate site for execution.
  - Each server processes its query and sends the result to the client.
  - The client combines the results of subqueries and produces the final result.

Recap

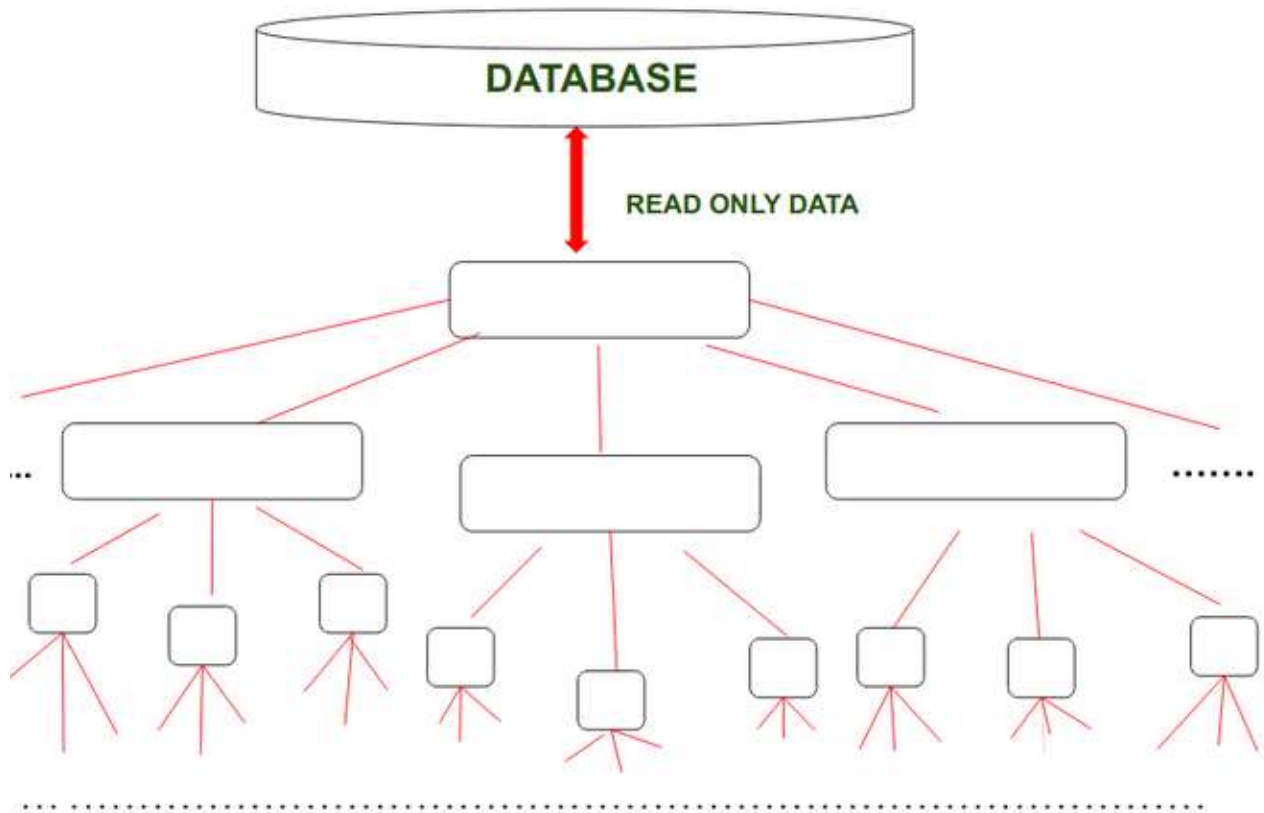
- Distributed Database Concepts
- Data Fragmentation, Replication and Allocation
- Types of Distributed Database Systems
- Query Processing
- Concurrency Control and Recovery
- 3-Tier Client-Server Architecture

Logical Database

- A Logical Database is a special type of ABAP (Advance Business Application and Programming) that is used to retrieve data from various tables and the data is interrelated to each other.
- Also, a logical database provides a read-only view of Data.

Structure Of Logical Database:

- A Logical database uses only a hierarchical structure of tables i.e. Data is organized in a Tree-like Structure and the data is stored as records that are connected to each other through edges (Links).
- Logical Database contains Open SQL statements which are used to read data from the database.
- The logical database reads the program, stores them in the program if required, and passes them line by line to the application program.


DATABASE
READ ONLY DATA

Features and Tasks of Logical Database

- We can select only that type of Data that we need.
- Data Authentication is done in order to maintain security.
- Logical Database uses hierarchical Structure due to this data integrity is maintained.
- With the help of the Logical database, we will read the same data from multiple programs.
- A logical database defines the same user interface for multiple programs.
- Logical Database ensures the Authorization checks for the centralized sensitive database.
- With the help of a Logical Database, Performance is improved. Like in Logical Database we will use joins instead of multiple SELECT statements, which will improve response time and this will increase the Performance of Logical Database.
-

WEB Database

- The Web-based database management system is one of the essential parts of DBMS and is used to store web application data.
- A web-based Database management system is used to handle those databases that are having data regarding E-commerce, E-business, blogs, e-mail, and other online applications.

Requirement of WEB Database

- The ability and right to use valuable corporate data in a fully secured manner.
- Provides data and vendor's autonomous connectivity that allows freedom of choice in selecting the DBMS for present and future use.
- The capability to interface to the database, independent of any proprietary Web browser and/or Web server.
- A connectivity solution that takes benefit of all the features of an organization's DBMS.
- An open-architectural structure that allows interoperability with a variety of systems and technologies; such as:
  - Different types of Web servers
  - Microsoft's Distributed Common Object Model (DCOM) / Common Object Model (COM)
  - CORBA / IIOP
  - Java / RMI which is Remote Method Invocation
  - XML (Extensible Markup Language)
  - Various Web services (SOAP, UDDI, etc.)
- A cost-reducing way which allows for scalability, development, and changes in strategic directions and helps lessen the costs of developing and maintaining those applications
- Provides support for transactions that span multiple HTTP requests.
- Gives minimal administration overhead.

Benefits of WEB Database

- Provides simplicity
- Web-DBMS is Platform independence
- Provides Graphical User Interface (GUI)
- Standardization
- Provides Cross-platform support
- Facilitates transparent network access
- Scalability
- Innovation

Object and Object-Relational Databases

- **Object databases (ODB)**
  - **Object data management systems (ODMS)**
  - Meet some of the needs of more complex applications
  - Specify:
    - Structure of complex objects
    - Operations that can be applied to these objects

Overview of Object Database Concepts

- Introduction to object-oriented concepts and features
  - Origins in OO programming languages
  - Object has two components:
    - State (value) and behavior (operations)
  - Instance variables
    - Hold values that define internal state of object
  - Operation is defined in two parts:
    - Signature or interface and implementation

Overview of Object Database Concepts (cont’d.)

- Inheritance
  - Permits specification of new types or classes that inherit much of their structure and/or operations from previously defined types or classes
- Operator overloading
  - Operation’s ability to be applied to different types of objects
  - Operation name may refer to several distinct implementations

Object Identity, and Objects versus Literals

- Unique identity
  - Implemented via a unique, system-generated object identifier (OID)
  - **Immutable**
- Most OO database systems allow for the representation of both objects and literals (or values)

Complex Type Structures for Objects and Literals

- Structure of arbitrary complexity
  - Contain all necessary information that describes object or literal
- Nesting **type constructors**
  - Construct complex type from other types
- Most basic constructors:
  - **Atom**
  - **Struct (or tuple)**
  - **Collection**

Complex Type Structures for Objects and Literals (cont'd.)

- Collection types:
  - **Set**
  - **Bag**
  - **List**
  - **Array**
  - **Dictionary**
- **Object definition language (ODL)**
  - Used to define object types for a particular database application

Figure 11.1
Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

```
define type EMPLOYEE
    tuple (  Fname:        string;
             Minit:        char;
             Lname:        string;
             Ssn:          string;
             Birth_date:   DATE;
             Address:      string;
             Sex:          char;
             Salary:       float;
             Supervisor:   EMPLOYEE;
             Dept:         DEPARTMENT;

define type DATE
    tuple (  Year:         integer;
             Month:        integer;
             Day:          integer; );

define type DEPARTMENT
    tuple (  Dname:        string;
             Dnumber:      integer;
             Mgr:          tuple (  Manager:     EMPLOYEE;
                                    Start_date:  DATE; );
             Locations:    set(string);
             Employees:    set(EMPLOYEE);
             Projects:     set(PROJECT); );
```

Encapsulation of Operations and Persistence of Objects

- Encapsulation
  - Related to abstract data types and information hiding in programming languages
  - Define **behavior** of a type of object based on operations that can be externally applied
  - External users only aware of interface of the operations
  - Divide structure of object into visible and hidden attributes

Object Behavior/Operations

- See figure 11.2

Encapsulation of Operations

- **Object constructor**
  - Used to create a new object
- **Destructor** operation
  - Used to destroy (delete) an object
- **Modifier** operations
  - Modify the states (values) of various attributes of an object
- **Retrieve** information about the object
- Dot notation used to apply operations to object

Persistence of Objects

- **Transient objects**
  - Exist in executing program
  - Disappear once program terminates
- **Persistent objects**
  - Stored in database and persist after program termination
  - **Naming mechanism**
  - **Reachability**

Type Hierarchies and Inheritance

- Inheritance
  - Definition of new types based on other predefined types
  - Leads to **type** (or **class**) **hierarchy**
- Type: **type name** and list of visible (public) **functions**
  - Format:
    - `TYPE_NAME: function, function, ..., function`

Type Hierarchies and Inheritance (cont'd.)

- **Subtype**
  - Useful when creating a new type that is similar but not identical to an already defined type
  - Example:
    - `EMPLOYEE subtype-of PERSON: Salary, Hire_date, Seniority`
    - `STUDENT subtype-of PERSON: Major, Gpa`

Type Hierarchies and Inheritance (cont'd.)

- **Extent**
  - Store collection of persistent objects for each type or subtype
  - Extents are subsets of the extent of class OBJECT
- **Persistent collection**
  - Stored permanently in the database
- **Transient collection**
  - Exists temporarily during the execution of a program

Other Object-Oriented Concepts

- **Polymorphism** of operations
  - Also known as **operator overloading**
  - Allows same operator name or symbol to be bound to two or more different implementations
  - Depending on type of objects to which operator is applied
- **Multiple inheritance**
  - Subtype inherits functions (attributes and methods) of more than one supertype

Other Object-Oriented Concepts (cont’d.)

- **Selective inheritance**
  - Subtype inherits only some of the functions of a supertype

Summary of Object Database Concepts

- Object identity
- Type constructor
- Encapsulation of operations
- Programming language compatibility
- Type hierarchies and inheritance
- Extents
- Polymorphism and operator overloading