

Advanced Database Management Systems

Vasudevan T V

Objectives

- To study the concept of object oriented databases, distributed databases and deductive databases.
- To impart basic concepts of data warehousing , data accessing from databases and information systems.

Course Contents

- Object Databases (Module I)
- Distributed Databases (Module II)
- Deductive Databases (Module III)
- Advanced Database Concepts (Module IV)
- RDBMS - Examples (Module V)

Text Book

1. Elmasri and Navathe, Fundamentals of Database Systems,
Addison Wesley

Reference Books

1. Ramakrishnan R. & Gehrke J , Database Management Systems, 3rd Edition., McGraw Hill.
2. Connolly and Begg, Database systems, 3rd Edition, Pearson Education, 2003
3. O'neil P. & O'neil E, Database Principles, Programming and Performance, 2nd Edition., Harcourt Asia (Morgan Kaufman).
4. Silberschatz, Korth H. F. & Sudarshan S, Database System Concepts, Tata McGraw Hill.

Module I

Object Databases

- Object Oriented Database Systems
- Object Relational Database Systems
- Object Definition Language
- Object Query Language
- Object Database Conceptual Design
- CORBA Standard for Distributed Objects

Module I

Introduction

Traditional data models such as relational, network and hierarchical were successful in developing databases for traditional business applications.

The common features of these applications are

- Uniformity of Data Items
- Record Orientation
- Small Data items
- Atomic Fields

Introduction

When database technology evolved, certain new applications came which do not satisfy one or more of these features.

Some of these applications are

- Computer Aided Design (CAD)
- Computer Aided Software Engineering (CASE)
- Multimedia Databases

Introduction

- Computer Aided Design (CAD)

A CAD database stores data related to an engineering design.

It includes

- components of the structure being designed
- the interrelationships of components
- older versions of designs

Introduction

- Computer Aided Software Engineering (CASE)
A CASE database stores data needed to assist software developers.

It includes

- source code
- dependencies among various software modules
- definitions and uses of variables
- development history of the software system

Introduction

- Multimedia Databases

A multimedia database contains

- text
- image
- audio
- video
- animation

Introduction

- Complex data types are needed to support such applications
- They are not available in traditional DBMSs
- For that databases which are based on objects were developed

Introduction

Object database systems are classified into two types:

- Object - Oriented Database Systems
- Object - Relational Database Systems

Object - Oriented Database Systems

- They were proposed as an alternative to relational database systems
- Their approach is highly influenced by object oriented programming languages
- They are developed based on a standard called Object Data Model (ODM)
- This standard was developed by the Object Database Management Group (ODMG)
- The query language used here is Object Query Language (OQL), which is equivalent to SQL in relational model
- O2 and ObjectStore are two examples of OODBMSs

Object - Relational Database Systems

- They extend the relational model by adding object-oriented features
- They provide a bridge between relational and object oriented paradigms
- The SQL:1999 standard extends SQL to support the object-relational model
- Many RDBMS vendors such as Oracle have extended their functionality to support object-relational data models

Overview of Object - Oriented Concepts

- Objects in a program exist only during its execution and after it they are destroyed
- Hence they are called transient objects
- In object oriented databases they persist beyond program termination. They can be accessed later by other programs
- Such objects are called persistent objects

Overview of Object - Oriented Concepts

- There is a unique system generated identifier for each object
- This can be compared with a primary key in relational model
- OO Model also supports encapsulation, inheritance and polymorphism

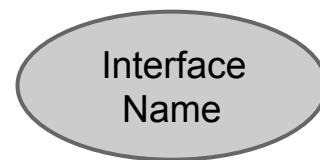
The Object Definition Language (ODL)

- Object Definition Language is used to define databases in Object Oriented Model
- This can be compared with Data Definition Language (DDL) in relational model

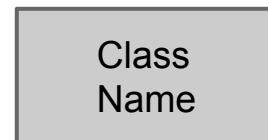
The Object Definition Language (ODL)

(a) Graphical Notation for representing ODL schemas

Interface



Class



Relationships



1 : 1



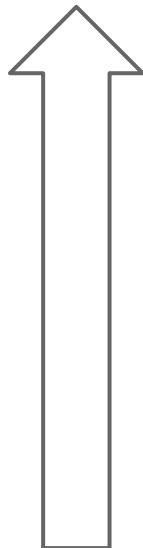
1 : N



M : N

The Object Definition Language (ODL)

Inheritance

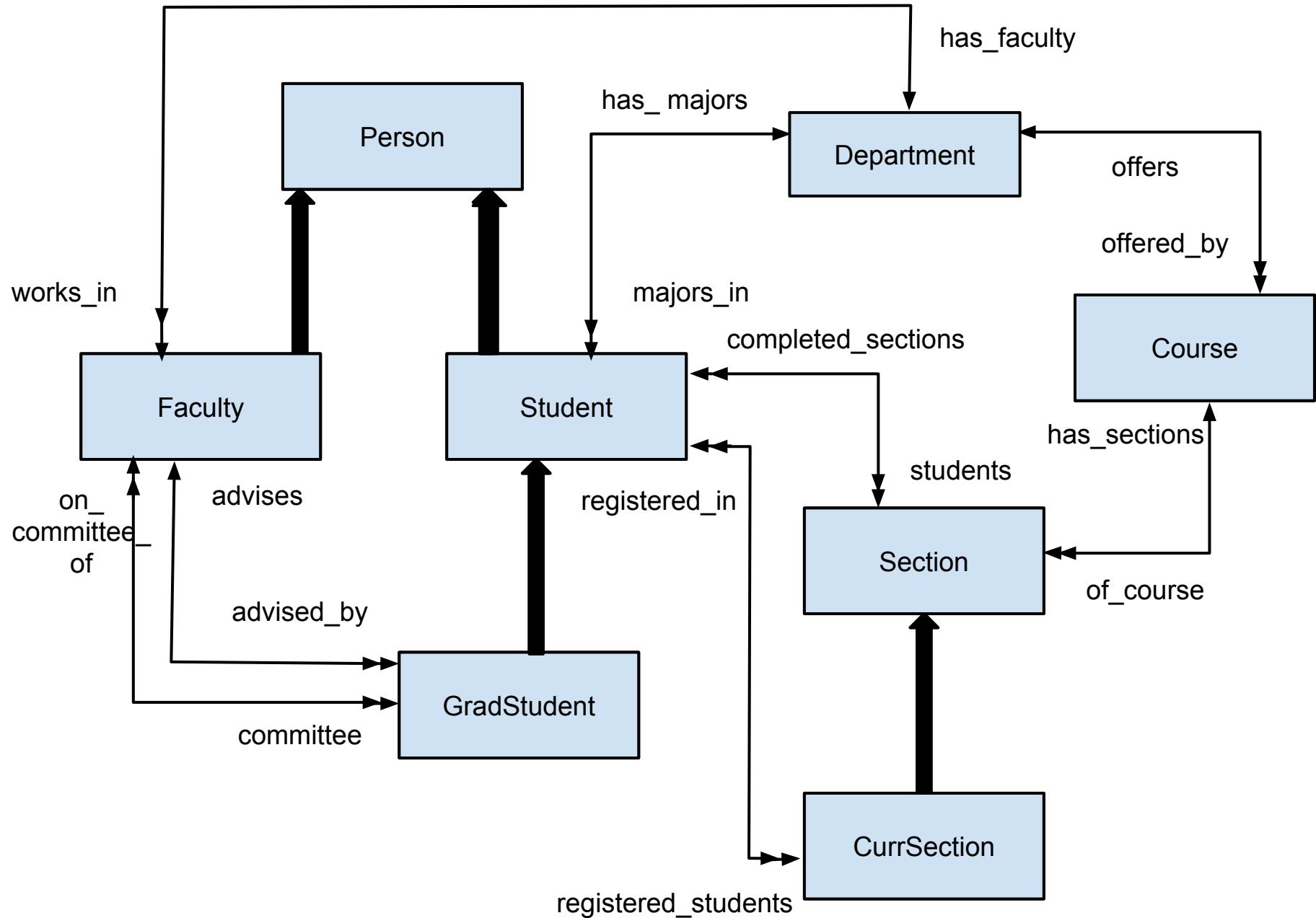


Interface
Inheritance
Using
": "



Class
Inheritance
Using
extends

Graphical Object Database Schema for Part of The UNIVERSITY Database



The Object Definition Language (ODL)

```
class Person
( extent persons
  key ssn )
{
  attribute struct Pname { string fname,
                           string mname,
                           string lname
                         } name;
  attribute string ssn;
  attribute date birthdate;
  attribute enum Gender { M, F } sex;
```

The Object Definition Language (ODL)

```
attribute struct Address
{ short aptno, string aptname,
  string street, string city,
  ▾ string state, short pincode } address;
short age( );
};

};
```

The Object Definition Language (ODL)

```
class Faculty extends Person
( extent faculty )
{
    attribute string rank;
    attribute float salary;
    attribute string phone;
    relationship Department works_in inverse
        Department :: has_faculty;
    relationship set < GradStudent > advises
        inverse GradStudent :: advised_by ;
```

The Object Definition Language (ODL)

```
relationship set < GradStudent > on_committee_of  
inverse GradStudent :: committee;
```



```
void give_raise ( in float raise);  
void promote ( in string new_rank);  
};
```

The Object Definition Language (ODL)

```
class Grade
(extent grades)
{
    attribute enum GradeValues {A, B, C, D, E, F}
    grade;
    relationship Section section inverse Section :: 
    students;
    relationship Student student inverse Student :: 
    completed_sections;
};
```

The Object Definition Language (ODL)

```
class Student extends Person
( extent students)
{
    attribute string class;
    attribute Department minors_in;
    relationship Department majors_in inverse
        Department :: has_majors ;
    relationship set < Grade > completed_sections inverse
        Grade :: student;
    relationship set < Curr Section > registered_in inverse
        CurrSection :: registered_students;
    void change_major (in string dname ) raises
        (dname_not_valid) ;
    float gpa ( );
```

The Object Definition Language (ODL)

```
void register ( in short secno ) raises (section _not _ valid);
void assign_grade ( in short secno, in GradeValue grade)
    raises (section_not_valid, grade_not_valid);
}
```



```
class Degree
{
    attribute string college;
    attribute string degree;
    attribute string year;
};
```

The Object Definition Language (ODL)

```
class GradStudent extends Student
( extent grad _ students)
{
    attribute set< Degree > degrees;
    relationship Faculty advisor inverse Faculty :: advises;
    relationship set < Faculty > committee inverse
        Faculty :: on _ committee _ of;
    void assign _ advisor ( in string lname , in string fname )
        raises ( faculty _ not _ valid);
    void assign _ committee _ member( in string lname, in string
        fname ) raises ( faculty _ not _ valid);
};
```

The Object Definition Language (ODL)

```
class Department
( extent departments
  key dname )
{
  attribute string dname;
  attribute string dphone;
  attribute Faculty hod;
  relationship set < Faculty > has_faculty inverse
    Faculty :: works_in;
  relationship set < Student > has_majors inverse
    Student :: majors_in;
  relationship set < Course > offers inverse
    Course :: offered_by;
};
```

The Object Definition Language (ODL)

```
class Course
(
    extent courses
    key cno
)
{
    attribute string cname;
    attribute string cno;
    attribute string description;
    relationship set <Section> has_sections inverse
        Section :: of_course;
    relationship Department offered_by inverse
        Department :: offers;
};
```

The Object Definition Language (ODL)

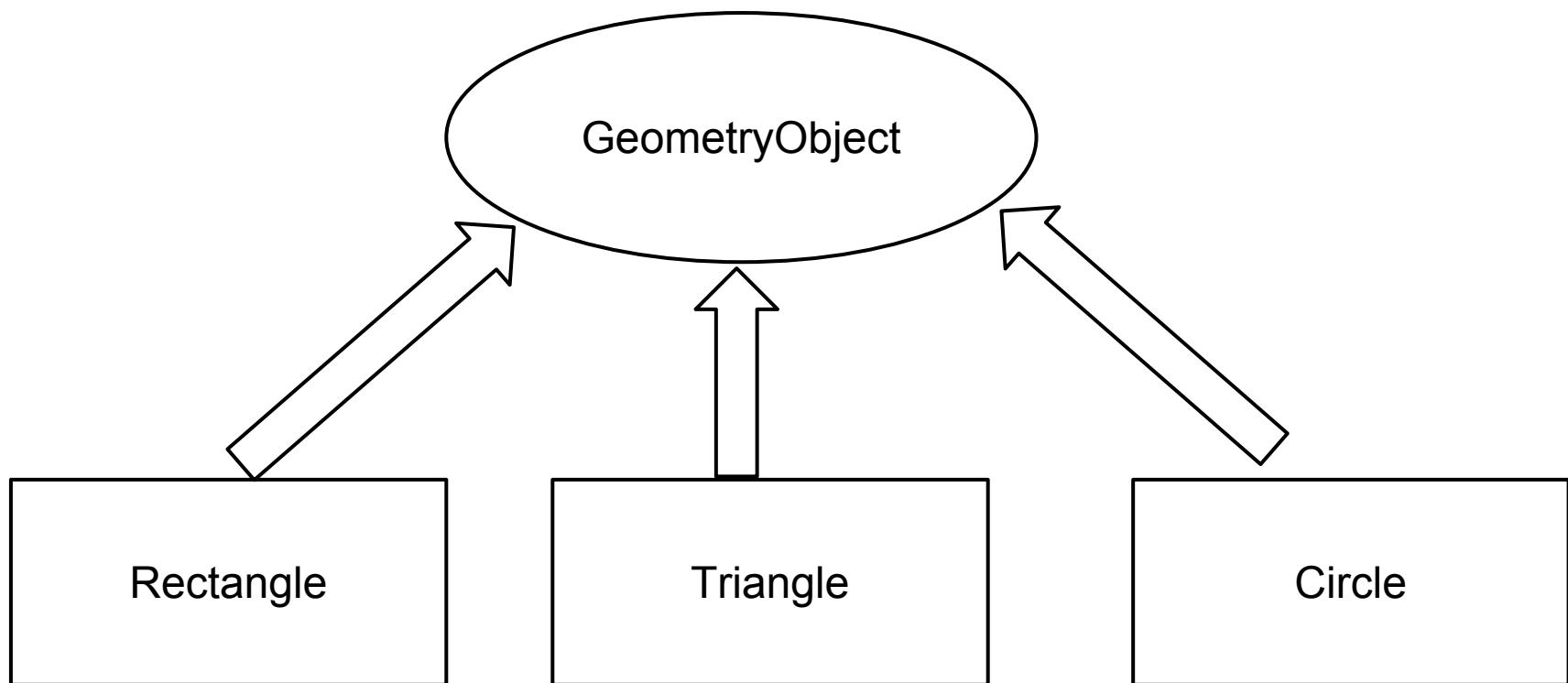
```
class Section
(extent sections)
{
    attribute short secno;
    attribute string year;
    attribute enum Semester { Odd, Even } sem;
    relationship set < Grade > students inverse
        Grade :: section ;
    relationship Course of_course inverse
        Course :: has_sections;
};
```

The Object Definition Language (ODL)

```
class CurrSection extends Section
(extent current_sections)
{
    relationship set <Student> registered_students
    inverse Student :: registered_in;
    void register_student ( in string ssn )
        raises ( student_not_valid, section_not_valid, section_full);
}
```

The Object Definition Language (ODL)

An Illustration of Interface Inheritance via " : "
Graphical Schema Representation



The Object Definition Language (ODL)

```
interface GeometryObject
{
    attribute enum Shape{ Rectangle, Triangle, Circle} shape;
    attribute struct Point { short x, short y } reference_ Point;
    float perimeter( );
    float area( );
    void translate( in short x_translation, in short y_translation);
    void rotate(in float angle_of_rotation);
};
```

The Object Definition Language (ODL)

```
class Rectangle : GeometryObject
(extent rectangles)
{
    attribute struct Point { short x, short y } reference_Point;
    attribute short length;
    attribute short height;
    attribute float orientation_angle;
}
```

The Object Definition Language (ODL)

```
class Triangle : GeometryObject
(extent triangles)
{
    attribute struct Point { short x, short y } reference_Point;
    attribute short side_1;
    attribute short side_2;
    attribute float side1_side2_angle;
    attribute float side1_orientation_angle;
};

class Circle : GeometryObject
(extent circles)
{
    attribute struct Point { short x, short y } reference_Point;
    attribute short radius;
};
```

The Object Definition Language(ODL)

Class Inheritance

1. done by using
extends
2. all features can be
inherited
3. Multiple
inheritance is not
allowed

Interface Inheritance

1. done by using :
2. only operations are
inherited
3. Multiple inheritance
can be done

The Object Query Language(OQL)

- OQL is the query language proposed for the ODMG object model
- It can be embedded into the object oriented programming languages such as C++, Small Talk and Java
- The OQL syntax for queries is similar to the syntax for SQL, with additional object oriented features

The Object Query Language(OQL)

Query

Retrieve the names of all faculty members who are having the rank Professor

```
select f.name  
from f in faculty  
where f.rank = 'Professor';
```

- Database Entry Point
 - For each query an entry point to the database is needed which can be any named persistent object
 - Usually the entry point is the name of the extent of a class

The Object Query Language(OQL)

Query

Retrieve the names of all faculty members who are having the rank Professor

```
select f.name  
from f in faculty  
where f.rank = 'Professor';
```

- Iterator Variable
 - Whenever a collection of persistent objects is referenced in an OQL query we should define an iterator variable that ranges over each object in the collection

The Object Query Language(OQL)

- There are three syntactic options for specifying iterator variables

f in faculty

faculty f

faculty as f

We will use the first construct in our examples

The Object Query Language(OQL)

Path Expressions

- The following are examples of path expressions which are also valid queries in OQL

csdepartment.hod;

// It returns a reference to the hod of the csdepartment

csdepartment.hod.rank;

// It returns the rank of the hod of the csdepartment

csdepartment.has_faculty;

// It returns a set of references to all faculty objects that are related to the csdepartment

The Object Query Language(OQL)

- ```
select f.rank
from f in csdepartment. has_faculty;
```

// This query will return the ranks of computer science faculty

# The Object Query Language(OQL)

- Duplicates can be eliminated using distinct keyword

```
select distinct f.rank
from f in csdepartment.has_faculty;
```

- The query

```
csdepartment.hod.advises;
```

will return the collection of graduate students that are advised by the hod of the computer science department

# The Object Query Language(OQL)

## Query

Retrieve the last and first names of graduate students that are advised by the hod of the computer science department along with their degrees

```
select struct(name : struct (last_name : s.name.lname,
 first_name : s.name.fname) ,
 degrees : (select struct (deg : d.degree,
 yr : d.year,
 college: d.college)
 from d in s.degrees)
 from s in csdepartment.hod.advises;
```

# The Object Query Language(OQL)

## Query

Retrieve the grade point average of all senior students majoring in computer science, with the result ordered by gpa, and within that by the last and first name

## OQL Query 1

```
select struct(last_name : s.name.lname,
 first_name : s.name.fname,
 gpa : s.gpa)
 from s in csdepartment.has_majors
 where s.class = 'senior'
 order by gpa desc, last_name asc, first_name asc;
```

# The Object Query Language(OQL)

## Query

Retrieve the grade point average of all senior students majoring in computer science, with the result ordered by gpa, and within that by the last and first name

## OQL Query 2

```
select struct(last_name : s.name.lname,
 first_name : s.name.fname,
 gpa : s.gpa)
from s in students
where s.majors_in.dname = 'Computer Science' and
s.class = 'senior'
order by gpa desc, last_name asc, first_name asc;
```

# The Object Query Language(OQL)

## Specifying Views as Named Queries

### Query

Create a View that defines a named query has\_minors to retrieve the student objects minoring in a given department

```
define has_minors(deptname) as
select s
from s in students
where s.minors_in.dname = deptname;
```

The user can now utilise the above view to write queries such as

```
has_minors('Computer Science');
```

# The Object Query Language(OQL)

- element operator

It ensures that there is **only one element** corresponding to the result. This operator returns that element. If the result is empty or it contains more than one element the operator will raise an exception

## Query

```
element (select d
 from d in departments
 where d.dname = 'Computer Science'
) ;
```

Since a department name is unique across all departments, the result should be one department

# The Object Query Language(OQL)

## Collection Operators

- They are operators applied on a collection of values
  - Aggregate Operators
  - Membership Operator
  - Quantification Operators

# The Object Query Language(OQL)

## Aggregate Operators

- **min**
  - Finds minimum value among a set of values
- **max**
  - Finds maximum value among a set of values
- **sum**
  - Finds the total of a set of values
- **count**
  - Counts a set of values
- **avg**
  - Finds the average of a set of values

# The Object Query Language(OQL)

## Aggregate Operators

- The operator count returns an integer type
- Other operators return the same type as the type of the collection

# The Object Query Language(OQL)

## Aggregate Operators

### Query

```
count (s in has_minors ('Computer Science'));
```

This query will return the number of students minoring in computer science

# The Object Query Language(OQL)

## Aggregate Operators

### Query

```
avg (select s.gpa
 from s in students
 where s.majors_in.dname = 'Computer Science'
 and s.class = 'senior');
```

This query will find out the average gpa of all seniors majoring in computer science

# The Object Query Language(OQL)

## Aggregate Operators

### Query

```
select d.dname
from d in departments
where count(d.has_majors) > 100
```

This query will display the names of all departments which are having more than 100 major students

# The Object Query Language(OQL)

- membership operator
  - in
- quantification operators
  - for all ( universal )
  - exists ( existential )

# The Object Query Language(OQL)

## Membership and Quantification Operators

- Let 'v' be a variable, 'c' a collection expression, 'b' a boolean expression and 'e' an element of a collection
- $(e \text{ in } c)$  returns true if e is a member of c
- $(\text{for all } v \text{ in } c : b)$  returns true if all the elements of collection c satisfy b
- $(\text{exists } v \text{ in } c : b)$  returns true if at least one element in c satisfy b

# The Object Query Language(OQL)

## Membership and Quantification Operators

### Query

```
select s.name.lname , s.name.fname
from s in students
where 'ADBMS' in
(select c cname
 from c in
 s.completed_sections.section.of_course);
```

This query will return the last names and first names of all students who have completed the ADBMS course

# The Object Query Language(OQL)

## Membership and Quantification Operators

### Query

```
Jeremy in has_minors ('Computer Science');
```

This query will check whether Jeremy is a computer science minor or not

# The Object Query Language(OQL)

## Membership and Quantification Operators

### Query

```
for all g in
 (select s
 from s in grad_students
 where s.majors_in.dname = 'Computer Science')
 : g.advisor in csdepartment . has_faculty;
```

This query will return true if all computer science graduate students are advised by computer science faculty

# The Object Query Language(OQL)

## Membership and Quantification Operators

### Query

```
exists g in
 (select s
 from s in grad_students
 where s.majors_in.dname = 'Computer Science')
 : g.gpa = 4;
```

This query will return true if any computer science graduate student have a gpa value of 4

# The Object Query Language(OQL)

## Ordered(Indexed) Collection Expressions

### Query

```
first (select struct (faculty : f.name.lname,
 salary : f.salary)
 from f in faculty
 order by f.salary desc);
```

- In the above query faculty with the highest salary will be displayed

# The Object Query Language(OQL)

## Ordered(Indexed) Collection Expressions

### Query

```
(select struct (last_name : s.name.lname,
 first_name : s.name.fname
 gpa : s.gpa)
 from s in csdepartment.has_majors
 order by gpa desc) [0:2];
```

This will retrieve the top three computer science major students based on gpa

# The Object Query Language(OQL)

- group by clause

## Query

```
select struct (deptname ,
 number_of_majors : count (Partition))
from s in students
group by deptname : s.majors_in.dname ;
```

This query will retrieve the number of major students in each department

# The Object Query Language(OQL)

- having clause

## Query

```
select deptname, avg_gpa : avg (select p.s.gpa
 from p in partition)
from s in students
group by deptname : s.majors_in.dname
having count (partition) > 100;
```

This query will retrieve the average gpa of major students for each department having more than 100 major students

# Object Database Conceptual Design

- Differences between Object Database (ODB) Design and Relational Database (RDB) Design
- Algorithm for Mapping an EER Schema to an ODB Schema

# Object Database Conceptual Design

## Differences between ODB design and RDB design

- Representation of Relationships
- Achieving Inheritance
- Specification of Operations

# Object Database Conceptual Design

## Representation of Relationships in ODB design

- Relationships are handled by having relationship attributes or reference attributes that include OID references to related objects
- Both single references and collection of references are allowed
- References for a binary relationship can be declared in a single direction or in both directions, depending on the type of access needed

# Object Database Conceptual Design

## Representation of Relationships in RDB design

- Relationships between tuples are specified by attributes with matching values
- They can be considered as value references
- They are specified via foreign keys, which are values of primary key attributes repeated in tuples of the referencing relation
- As an example, branch-name in account relation references branch-name in branch relation

# Object Database Conceptual Design

## Achieving Inheritance in ODB design and RDB design

- In ODB, Inheritance is achieved by using constructs such as derived ( : ) and EXTENDS
- In basic RDB model , there is no construct for providing inheritance
- However, constructs for inheritance are available in extended relational model and object relational model

# Object Database Conceptual Design

## Specifying operations in ODB design and RDB design

- In ODB design, it is necessary to specify operations since they are part of the class specification
- In RDB design, it may be delayed as it is not strictly required until the implementation phase

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 1

- Create an ODL class for EER entity type or subclass
- Multivalued attributes are declared using the set ( chosen if duplicates are not allowed), bag ( chosen if duplicates are allowed ), or list (chosen if values are ordered ) constructors
- Composite attributes are specified using a struct declaration
- Declare an extent for each class, and key specification if needed

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 2

- Add relationship properties or reference attributes to the ODL classes for every associated binary relationship
- If references are needed in both directions in a binary relationship, we can use relationship properties that are inverse references of one another
- Otherwise, we can use a reference attribute in the referencing relation whose type is the referenced class name
- Depending upon the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single valued or multivalued

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 3

- Include appropriate operations for each class
- These are not available from the EER schema and must be added based on the original requirements
- The constructor method should check for any constraint that must be satisfied when a new object is created
- Similarly, the destructor method should check for any constraint that must be satisfied when an object is deleted
- We can also specify constraint checks as part of other operations

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 4

- An ODL class that corresponds to a subclass in the EER schema inherits( via EXTENDS ) the type and methods of its superclass in the ODL schema
- Its specific attributes, relationship references and operations are specified as discussed in steps 1, 2 and 3

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 5

- Weak entity types can be mapped in the same way as regular entity types
- However, weak entity types that do not participate in any relationships except their identifying relationship, can be specified as composite multivalued attributes of the owner entity type

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 6

- Categories ( Union Types ) in EER schema are difficult to map to ODL
- However, this can be done by declaring a class to represent the category and defining 1:1 relationships between the category and each of its superclasses
- Another option is to use a Union Type, if it is available

# Object Database Conceptual Design

## Mapping an EER Schema to an ODB Schema

### Step 7

- An n-ary relationship with degree  $n > 2$  can be mapped into a separate class , with appropriate references to each participating class
- We can also use this option for a binary relationship containing relationship attributes

# Overview of the CORBA Standard For Distributed Objects

- CORBA ( Common Object Request Broker Architecture ) is an object management standard
- It allows objects to communicate in a distributed heterogeneous environment
- It provides transparency across network, operating system and programming language boundaries

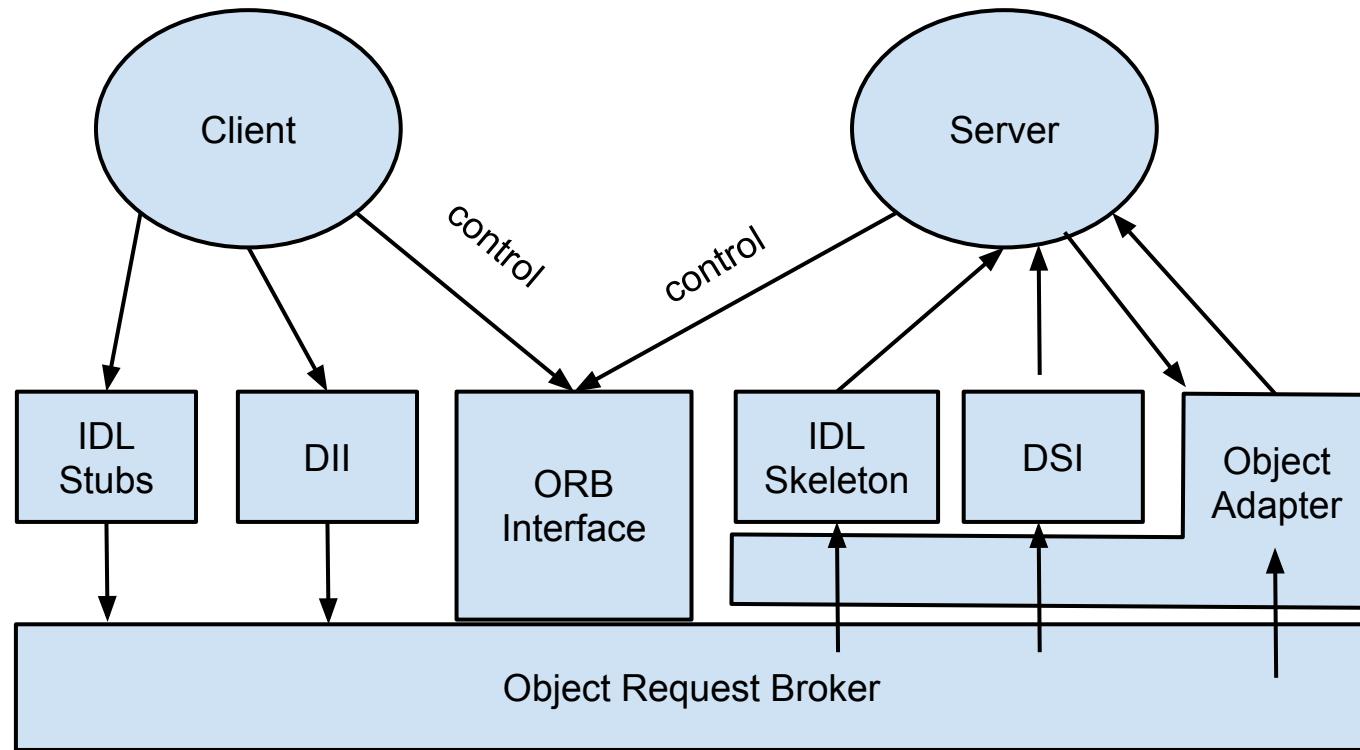
# Overview of the CORBA Standard For Distributed Objects

- It is the responsibility of the Object Request Broker (ORB) to provide the transparency across network, operating system and programming language boundaries
- It receives method invocations from one object , called the client, and delivers them to the appropriate target object, called the server
- The interface of a CORBA object is specified using Interface Definition Language (IDL)
- It is a programming language independent specification of the interface

# Overview of the CORBA Standard For Distributed Objects

- It describes only the functionality and not the implementation of an object
- The methods specified in an interface definition can be implemented in and invoked from a programming language that provides CORBA bindings such as C++, ADA, SmallTalk and Java
- This IDL specification is converted into the target programming language by the IDL compiler

# Overview of the CORBA Standard For Distributed Objects



CORBA 2.0 ORB Architecture

# Overview of the CORBA Standard For Distributed Objects

- An IDL compiler generates three files
  - header file
  - client source file
  - server source file

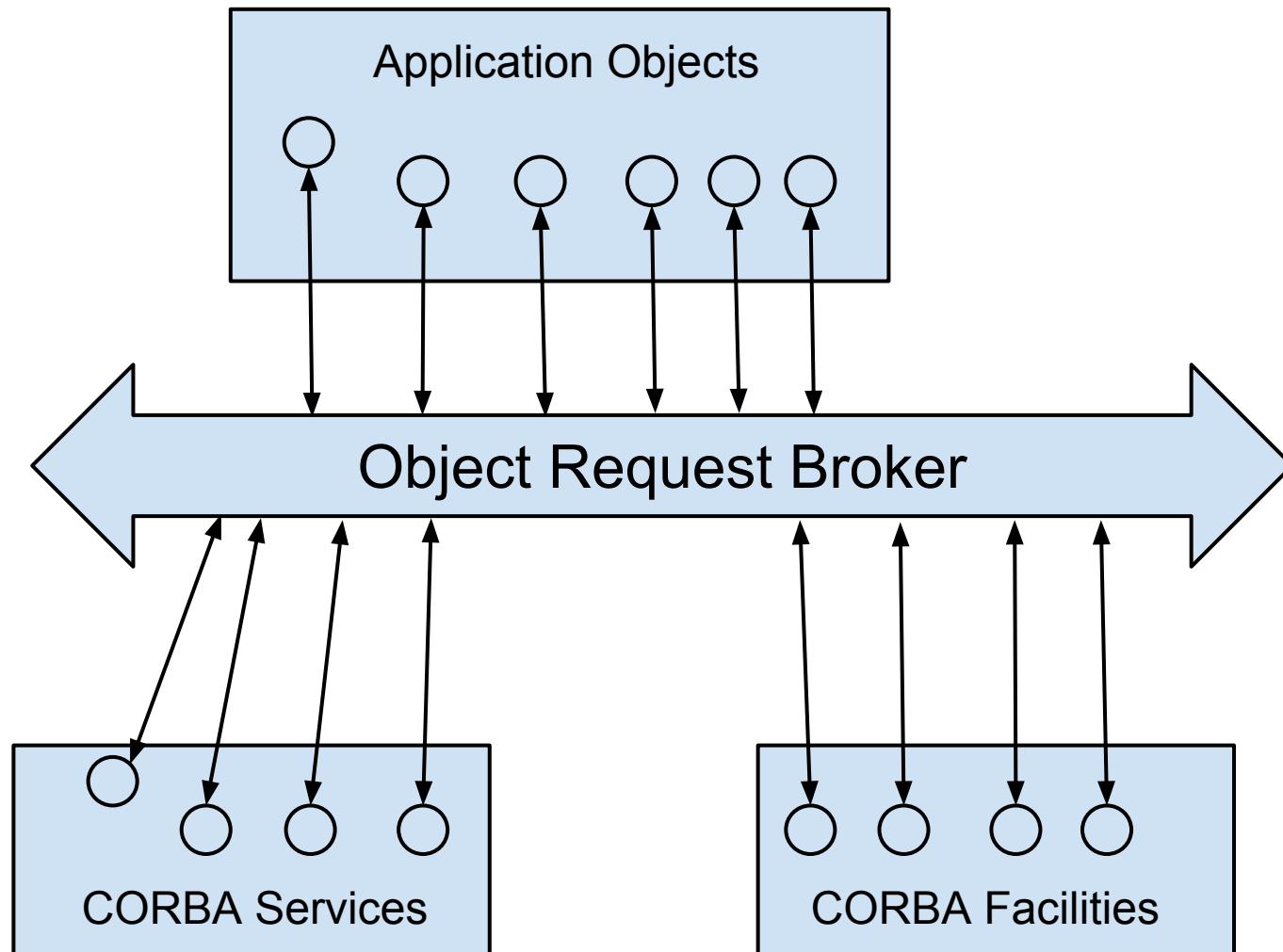
# Overview of the CORBA Standard For Distributed Objects

- The header file defines the programming language specific view of the IDL interface definition which is included in both the server and the client
- The client source file, called the stub code, is included in the source code of the client to transmit requests to the server
- The server source file, called the skeleton code, is included in the source code of the server to accept requests from a client
- The programmer writing the client implementation uses the header and stub code
- The programmer writing the server implementation uses the header and skeleton code

# Overview of the CORBA Standard For Distributed Objects

- The Dynamic Invocation Interface (DII) allows a client to discover objects at runtime to invoke their methods and to receive results from these invocations
- The Dynamic Skeleton Interface (DSI) delivers these dynamic requests to the target object
- The Object Adapter is responsible for registering object implementations
- It includes mapping from the name of the server object to the name of the executable code of the object implementation
- It is also responsible for invoking methods, either statically or dynamically

# Overview of the CORBA Standard For Distributed Objects



**CORBA Object Management Architecture**

# Overview of the CORBA Standard For Distributed Objects

- The Object Management Architecture (OMA) is built on top of the core CORBA architecture
- The OMA provides the following services to support distributed applications
  - CORBA services
  - CORBA facilities

# Overview of the CORBA Standard For Distributed Objects

- CORBA services provide system level services to objects, such as
  - naming services
  - event services
- CORBA facilities provide higher-level services for application objects
  - horizontal facilities which span application domains
  - vertical facilities which are specific to an application

# **Module II**

# **Distributed Databases**

- Distributed Database Concepts
- Advantages of Distributed Databases
- Additional Functions of Distributed Databases
- Data Fragmentation, Replication and Allocation
- Types of Distributed Database Systems
- Query Processing and Concurrency Control in Distributed Databases

# Module II

# Distributed Database Concepts

- Distributed databases bring the advantages of distributed computing to the database domain
- A distributed computing system consists of
  - A number of processing elements
  - They may not be homogeneous
  - They are interconnected by a computer network
  - They cooperate in performing certain assigned tasks

# Distributed Database Concepts

Its main advantages are

- An unmanageable problem is divided into smaller parts and can be solved efficiently in a coordinated manner
- More computing power is obtained to solve a complex task
- Each processing element can be managed independently

# Distributed Database Concepts

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

# Parallel Vs Distributed Technology

Multiprocessor architectures can be classified into different types

- Shared Memory ( Shared Everything ) ( Tightly Coupled ) Architecture
- Shared Disk ( Loosely Coupled ) Architecture
- Shared Nothing Architecture

# Parallel Vs Distributed Technology

- Shared Memory Architecture
  - Multiple processors share secondary storage
  - They also share primary memory

# Parallel Vs Distributed Technology

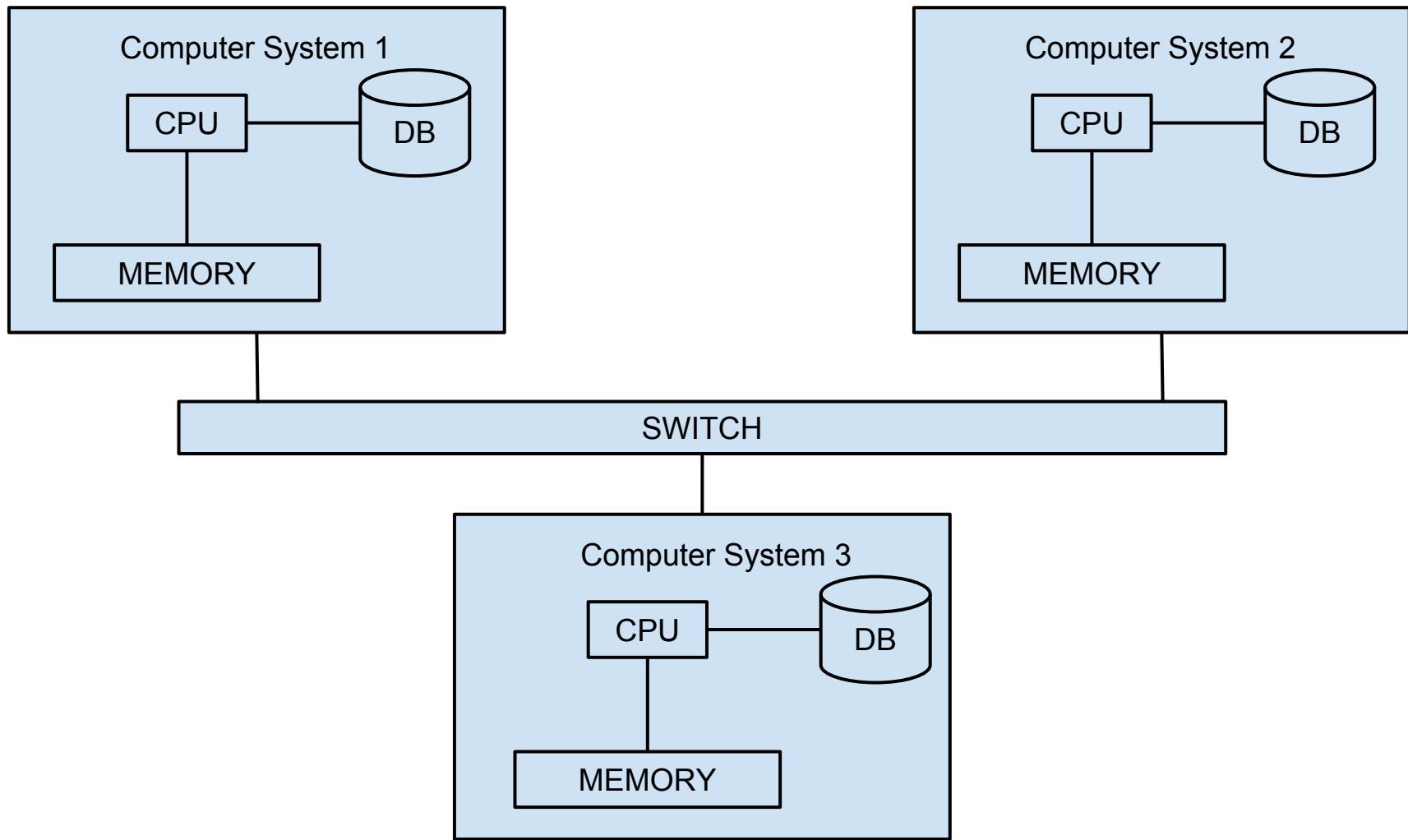
- Shared Disk Architecture
  - Multiple processors share secondary storage
  - But, each has their own primary memory

# Parallel Vs Distributed Technology

- Shared Nothing Architecture
  - Every processor has its own secondary storage
  - Also, each has their own primary memory
  - They communicate over a high speed interconnection network

# Parallel Vs Distributed Technology

## Shared Nothing Architecture

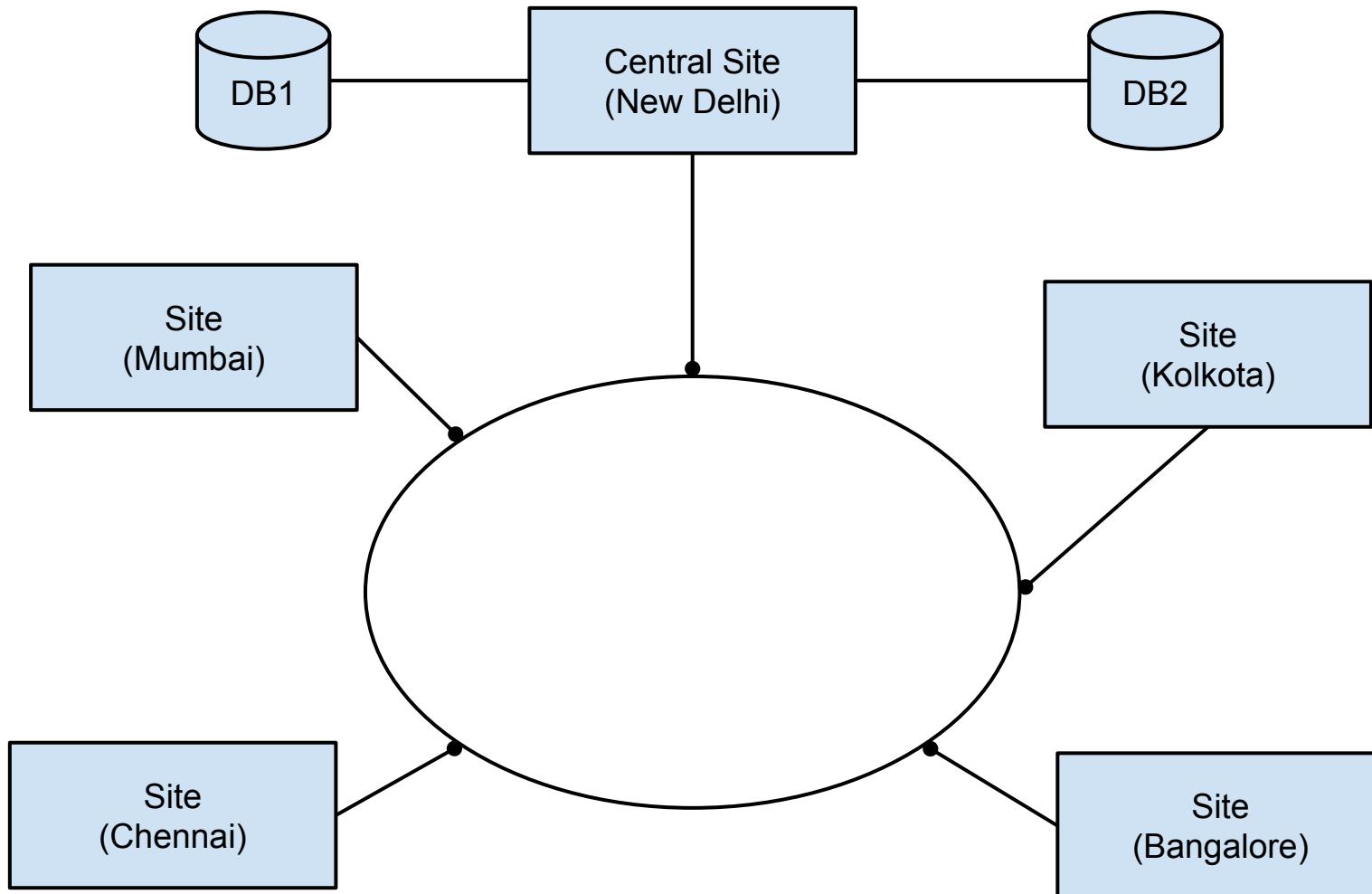


# Parallel Vs Distributed Technology

- They are called parallel architecture since they use parallel processor technology
- Even Though shared nothing architecture resembles a distributed environment, major differences exist between them
- In shared nothing architecture, there is symmetry and homogeneity of nodes
- But in a distributed environment, heterogeneity of hardware and operating system at every node is common

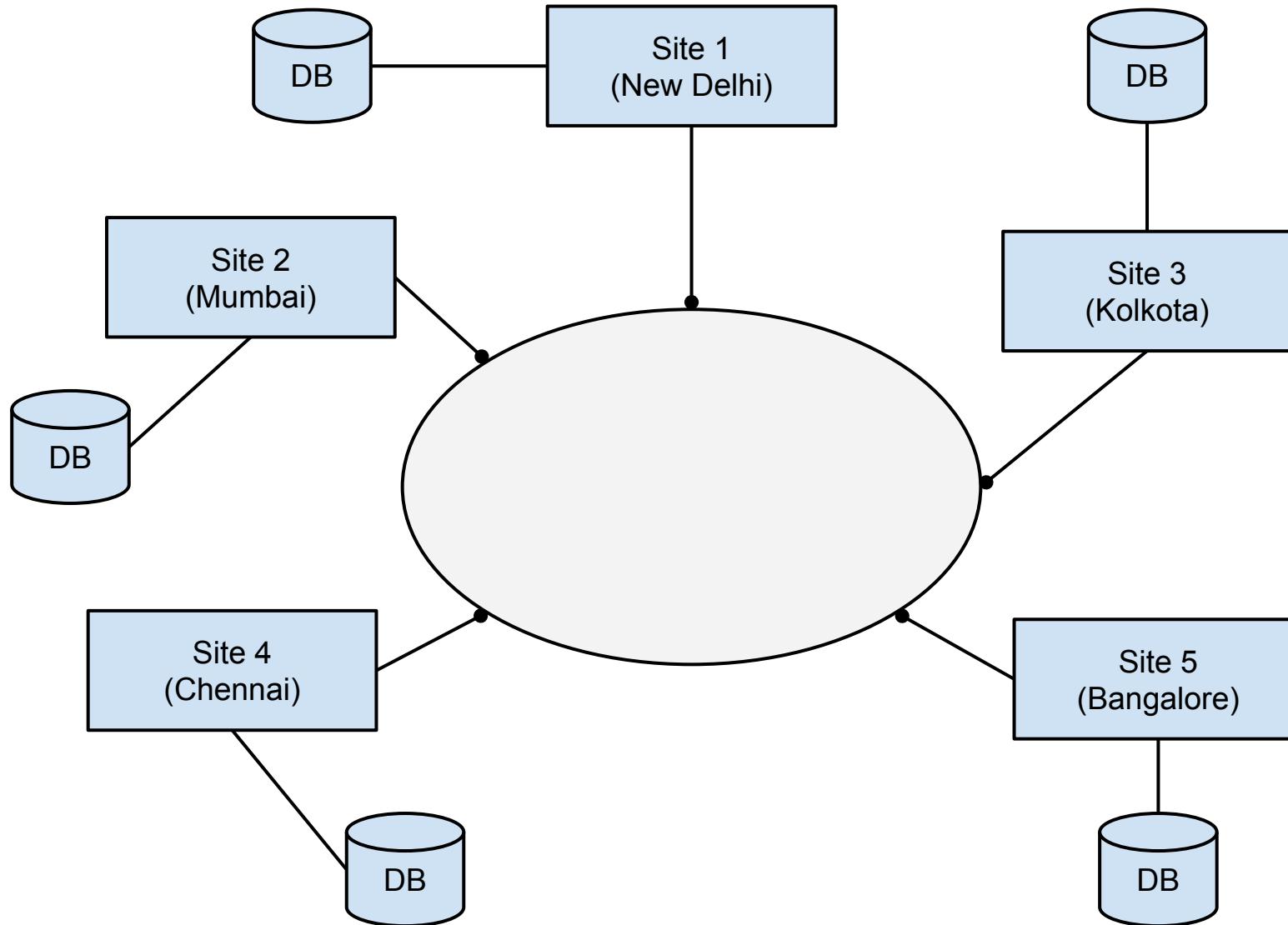
# Parallel Vs Distributed Technology

## Centralised Database Architecture



# Parallel Vs Distributed Technology

## Distributed Database Architecture



# Advantages of Distributed Databases

- Management of distributed data with different levels of transparency
- Increased reliability and availability
- Improved performance
- Easier expansion

# Advantages of Distributed Databases

- Management of distributed data with different levels of transparency
  - A database management system is said to be distribution transparent if details about the location of each file is hidden from the user
    - distribution or network transparency
    - replication transparency
    - fragmentation transparency

# Advantages of Distributed Databases

- Distribution or Network Transparency
  - This refers to the freedom for the user about the operational details of the network
    - Location Transparency
    - Naming Transparency

# Advantages of Distributed Databases

- Location Transparency

It refers to the fact that the command used to perform a task is independent of the location of data and the location from which the command was issued

- Naming Transparency

It implies that once a name is specified , the named objects can be accessed unambiguously without additional specification

# Advantages of Distributed Databases

- Replication Transparency
  - Copies of data may be stored at multiple sites for better availability, performance and reliability
  - Replication transparency makes the user unaware of the existence of copies

# Advantages of Distributed Databases

- Fragmentation Transparency

Fragmentation transparency makes the user unaware of the existence of fragments

- Horizontal Fragmentation
- Vertical Fragmentation

# Advantages of Distributed Databases

- Horizontal Fragmentation

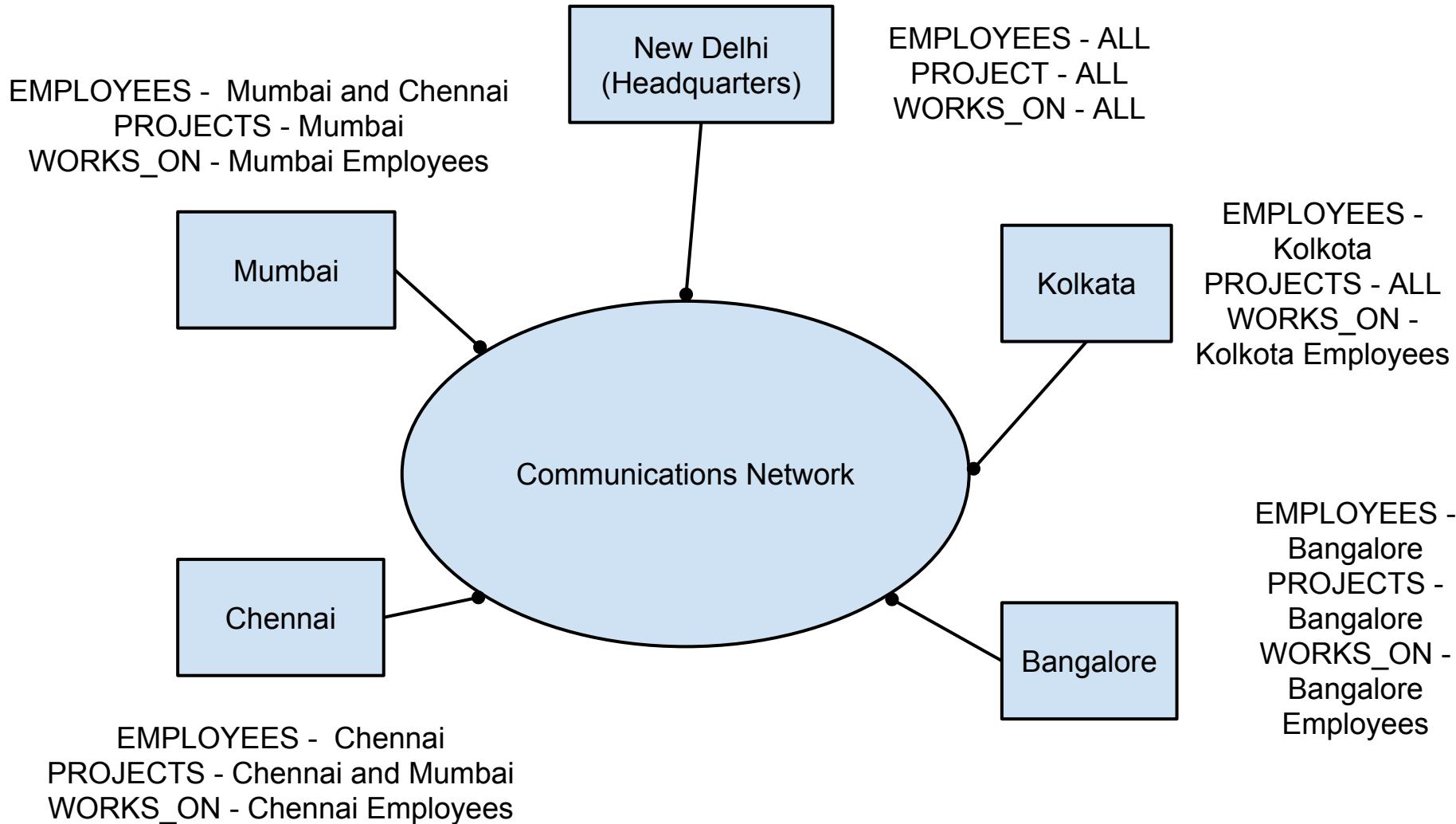
It distributes a relation into a set of tuples ( rows )

- Vertical Fragmentation

It distributes a relation into subrelations where each sub relation is defined by a subset of the columns of the original relation

# Advantages of Distributed Databases

## Data Distribution and Replication among Distributed Databases



# Advantages of Distributed Databases

- Increased Reliability and Availability
  - Reliability is defined as the ability of a system to run successfully ( not down ) at a certain time point
  - Availability is defined as the ability of a system to make itself available continuously during a time interval

# Advantages of Distributed Databases

- Increased Reliability and Availability
  - In a centralised system the failure at one site ( central site ) make the entire data unavailable to the user
  - In a distributed system even if one site fails, we can access data from others
  - This improves both reliability and availability
  - Further improvement can be achieved by judiciously replicating the data and software at more than one site

# Advantages of Distributed Databases

- Improved Performance
  - A distributed DBMS fragments the database by keeping the data closer to where it is needed most
  - This is called data localisation
  - This will reduce access delays involved in wide area networks
  - Local queries and transactions accessing data at a single site have better performance because of the smaller local databases

# Advantages of Distributed Databases

- Improved Performance
  - Besides each site has smaller number of transactions on local databases compared to those in a centralised database
  - Moreover inter query and intra query parallelism can be achieved
  - Inter query parallelism is achieved by executing multiple queries at different sites in parallel
  - Intra query parallelism is achieved by breaking up a query into a number of sub queries that execute in parallel

# Advantages of Distributed Databases

- Easier Expansion

In a distributed environment, expansion of the system can be done easily by

- adding more data
- increasing database size
- adding more processors

The transparencies discussed above lead to a compromise between ease of use and the overhead cost of providing transparency

# **Additional Functions of Distributed Databases**

- Keeping track of data
- Distributed query processing
- Distributed transaction management
- Replicated data management
- Distributed database recovery
- Security
- Distributed directory ( catalogue ) management

# Additional Functions of Distributed Databases

- Keeping track of data

The ability to keep track of the data distribution, fragmentation and replication

- Distributed query processing

The ability to access remote sites and transmit queries and data among the various sites via a communication network

# Additional Functions of Distributed Databases

- Distributed transaction management
  - The ability to manage transactions that access data from more than one site
  - To synchronise the access to distributed data
  - To maintain the integrity of the overall database

# Additional Functions of Distributed Databases

- Replicated data management
  - The ability to decide which copy of a replicated data item to access
  - To maintain the consistency of copies of a replicated data item

# **Additional Functions of Distributed Databases**

- Distributed database recovery  
The ability to recover from individual site crashes and failures of communication links
- Security  
The ability to ensure security of data and manage authorisation / access privileges of users
- Distributed directory ( catalogue ) management  
A directory contains information ( metadata ) about data in the database. The directory may be global for the entire DDB or local for each site

# Data Fragmentation, Replication and Allocation

- Data fragmentation is the process by which a database is divided into several logical units called fragments
- Data replication is the method by which we store different copies of data at different sites
- Data allocation is the process of allocating fragments or replicas for storage at different sites
- These techniques are used during distributed database design
- The information concerning these techniques is stored in a global directory that is accessed by the DDBS applications

# Data Fragmentation

- Data fragmentation is done in such a way that we can reconstruct the original relation from the fragments
- This can be done by applying either union operation or natural join operation on the fragments
- There are three types of fragmentation
  - Horizontal fragmentation
  - Vertical fragmentation
  - Mixed ( Hybrid ) fragmentation

# Horizontal Fragmentation

- Here the relation  $r$  is partitioned into several fragments  
 $r_1, r_2, r_3, \dots, r_n$
- Each tuple of  $r$  must belong to at least one of the fragments
- A fragment can be defined by applying selection predicate on the global relation  $r$

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

- We can reconstruct the global relation by taking the union of all fragments

$$r = r_1 \cup r_2 \cup r_3 \cup \dots \cup r_n$$

# Horizontal Fragmentation

account relation

| branch_name | account_number | balance |
|-------------|----------------|---------|
| Thrissur    | A - 305        | 500     |
| Thrissur    | A - 226        | 336     |
| Kuttippuram | A - 177        | 205     |
| Kuttippuram | A - 402        | 10000   |
| Thrissur    | A - 155        | 62      |
| Kuttippuram | A - 408        | 1123    |
| Kuttippuram | A - 639        | 750     |

# Horizontal Fragmentation

$\text{account}_1 = \sigma_{\text{branch\_name} = \text{"Thrissur}} (\text{ account })$

| <b>branch_name</b> | <b>account_number</b> | <b>balance</b> |
|--------------------|-----------------------|----------------|
| Thrissur           | A - 305               | 500            |
| Thrissur           | A - 226               | 336            |
| Thrissur           | A - 155               | 62             |

# Horizontal Fragmentation

$\text{account}_2 = \sigma_{\text{branch\_name} = \text{"Kuttippuram"}} (\text{account})$

| <b>branch_name</b> | <b>account_number</b> | <b>balance</b> |
|--------------------|-----------------------|----------------|
| Kuttippuram        | A - 177               | 205            |
| Kuttippuram        | A - 402               | 10000          |
| Kuttippuram        | A - 408               | 1123           |
| Kuttippuram        | A - 639               | 750            |

# Vertical Fragmentation

- It is done by decomposing a relation into smaller relations where each relation consists of subsets of attributes of the original relation
- Vertical fragmentation is more meaningful for databases which are not highly normalised
- Vertical fragmentation of  $r(R)$  involves definition of several subsets of attributes  $R_1, R_2, R_3, \dots, R_n$  of the schema  $R$  such that

$$R = R_1 \cup R_2 \cup R_3 \cup \dots \cup R_n$$

- Each fragment  $r_i$  of  $r$  is defined by

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

# Vertical Fragmentation

- The fragmentation should be done in such a way that we are able to reconstruct the original relation  $r$  from the fragments by taking natural join operation
- One way of ensuring that original relation is reconstructed from the fragments is to include the primary key of  $r$  in every  $r_i$
- More generally any superkey can be used

# Vertical Fragmentation

- It is often convenient to add a special attribute called tuple-id to the relation schema R
- It is used to distinguish a tuple from others
- The physical or logical address of a tuple can be used as a tuple-id
- We can make this tuple-id as the primary key of relation schema R

# Vertical Fragmentation

deposit

| branch_name | account_no | customer_name | balance |
|-------------|------------|---------------|---------|
| Thrissur    | A - 305    | Alice         | 500     |
| Thrissur    | A - 226    | Ahmed         | 336     |
| Kuttippuram | A - 177    | Ahmed         | 205     |
| Kuttippuram | A - 402    | Rajesh        | 10000   |
| Thrissur    | A - 155    | Rajesh        | 62      |
| Kuttippuram | A - 408    | Rajesh        | 1123    |
| Kuttippuram | A - 639    | Ramesh        | 750     |

# Vertical Fragmentation

'deposit' (after adding tuple-id )

| branch_name | account_no | customer_name | balance | tuple_id |
|-------------|------------|---------------|---------|----------|
| Thrissur    | A - 305    | Alice         | 500     | 1        |
| Thrissur    | A - 226    | Ahmed         | 336     | 2        |
| Kuttippuram | A - 177    | Ahmed         | 205     | 3        |
| Kuttippuram | A - 402    | Rajesh        | 10000   | 4        |
| Thrissur    | A - 155    | Rajesh        | 62      | 5        |
| Kuttippuram | A - 408    | Rajesh        | 1123    | 6        |
| Kuttippuram | A - 639    | Ramesh        | 750     | 7        |

# Vertical Fragmentation

deposit<sub>1</sub> (after vertical fragmentation )

| branch_name | Customer_name | tuple_id |
|-------------|---------------|----------|
| Thrissur    | Alice         | 1        |
| Thrissur    | Ahmed         | 2        |
| Kuttippuram | Ahmed         | 3        |
| Kuttippuram | Rajesh        | 4        |
| Thrissur    | Rajesh        | 5        |
| Kuttippuram | Rajesh        | 6        |
| Kuttippuram | Ramesh        | 7        |

# Vertical Fragmentation

deposit<sub>2</sub>(after vertical fragmentation )

| account_no | balance | tuple_id |
|------------|---------|----------|
| A - 305    | 500     | 1        |
| A - 226    | 336     | 2        |
| A - 177    | 205     | 3        |
| A - 402    | 10000   | 4        |
| A - 155    | 62      | 5        |
| A - 408    | 1123    | 6        |
| A - 639    | 750     | 7        |

# Mixed ( Hybrid ) Fragmentation

- It is a combination of horizontal fragmentation and vertical fragmentation
- We have divided the deposit relation vertically into two fragments  $\text{deposit}_1$  and  $\text{deposit}_2$
- We can now further divide  $\text{deposit}_1$  using the horizontal fragmentation scheme, into the following two fragments

$$\text{deposit}_{1a} = \sigma_{\text{branch\_name} = \text{"Thrissur}} (\text{deposit}_1)$$
$$\text{deposit}_{1b} = \sigma_{\text{branch\_name} = \text{"Kuttippuram}} (\text{deposit}_1)$$

- Thus the relation deposit is divided into three fragments:

$$\text{deposit}_{1a}, \text{deposit}_{1b} \text{ and } \text{deposit}_2$$

# Mixed Fragmentation

deposit<sub>1a</sub>

| branch_name | Customer_name | tuple_id |
|-------------|---------------|----------|
| Thrissur    | Alice         | 1        |
| Thrissur    | Ahmed         | 2        |
| Thrissur    | Rajesh        | 5        |

# Mixed Fragmentation

deposit<sub>1b</sub>

| <b>branch_name</b> | <b>Customer_name</b> | <b>tuple_id</b> |
|--------------------|----------------------|-----------------|
| Kuttippuram        | Ahmed                | 3               |
| Kuttippuram        | Rajesh               | 4               |
| Kuttippuram        | Rajesh               | 6               |
| Kuttippuram        | Ramesh               | 7               |

# Data Replication

- It is the process of keeping different copies of data at different sites
- There are three types of data replication
  - Full Replication
  - No Replication
  - Partial Replication

# Data Replication

- Full Replication

Here the whole database is stored at every site in the distributed system

- advantages
  - It improves the performance of global queries, since the result of such a query can be accessed from a local site
  - It improves availability of data, since the system will continue to run even if only one site is up

# Data Replication

- Full Replication
  - disadvantages
    - It can slow down update operations heavily, since they are to be performed on every copy of data
    - It also makes concurrency control and recovery techniques more expensive

# Data Replication

- No Replication

Here each fragment will be stored at exactly one site

- Partial Replication

In this case, some of the fragments may replicated while others may not

# Data Allocation (Data Distribution)

- Each fragment must be assigned to a particular site in the distributed network. This is called data allocation
- The choice of sites and degree of replication required depend on factors such as
  - availability required
  - types and frequencies of transactions at various sites

# Data Allocation (Data Distribution)

- If high availability is required, most transactions are read only and transactions can be submitted at any site, a fully replicated database is the best choice
- If transactions are directed towards a particular site, then we can limit data allocation to that site
- Data that is accessed at multiple sites can be replicated at those sites
- If many updates are performed, it will be useful to limit replication

# Types of Distributed Database Systems

Based on the degree of homogeneity of DDBMS software used there are two types of distributed database systems

- Homogeneous DDBMS
- Heterogeneous DDBMS

# Types of Distributed Database Systems

- Homogeneous DDBMS

If all servers and clients use identical software, such a DDBMS is called homogeneous DDBMS

- Heterogeneous DDBMS

If all servers and clients use different software, such a DDBMS is called heterogeneous DDBMS

# Types of Distributed Database Systems

Based on the degree of local autonomy of individual sites we can classify distributed database systems

- A DDBMS is said to have local autonomy if local data can be directly accessed by local users

# Types of Distributed Database Systems

Based on the degree of local autonomy of individual sites there are three types of distributed database systems

- centralised DDBMS
- federated database management system
- multidatabase system

# Types of Distributed Database Systems

- centralised DDBMS

Here, there is a single conceptual schema through which all data is accessed
- federated database management system
  - has local autonomy
  - There is a global schema which is shared
- multidatabase system
  - has local autonomy
  - it interactively creates global schema as needed by the application

# **Types of Distributed Database Systems**

Design issues related with a heterogeneous Federated DBMS are given below

- Differences in data models
- Differences in constraints
- Differences in query languages
- Semantic Heterogeneity
- To provide different types of autonomies to component databases

# Types of Distributed Database Systems

- Differences in data models
  - In a heterogeneous FDBS, one server may be a relational DBMS, another a network DBMS and a third an object DBMS
  - To deal with them uniformly via a single global schema and to process them in a single language is challenging

# Types of Distributed Database Systems

- Differences in constraints
  - In the relational model, relationships are represented as referential integrity constraints
  - In an object database, relationships are indicated using inverse references
  - Hence, the global schema must deal with differences and conflicts in constraints

# Types of Distributed Database Systems

- Differences in query languages
  - Even with the same data model, the languages and their versions can vary
  - For example, SQL has multiple versions such as SQL-92, SQL-99 and SQL-2003
  - Each system has its own set of data types, comparison operators, string manipulation features etc.

# Types of Distributed Database Systems

- Semantic Heterogeneity
  - It occurs when there are differences in the meaning of the same or related data
  - For example, two customer account databases in USA and Japan will have different sets of attributes about customer accounts required by the accounting practices
  - Hence customer and account relations in these databases may have some common and some distinct information

# Types of Distributed Database Systems

- To provide different types of autonomies to component databases
  - Communication Autonomy
  - Execution Autonomy
  - Association Autonomy

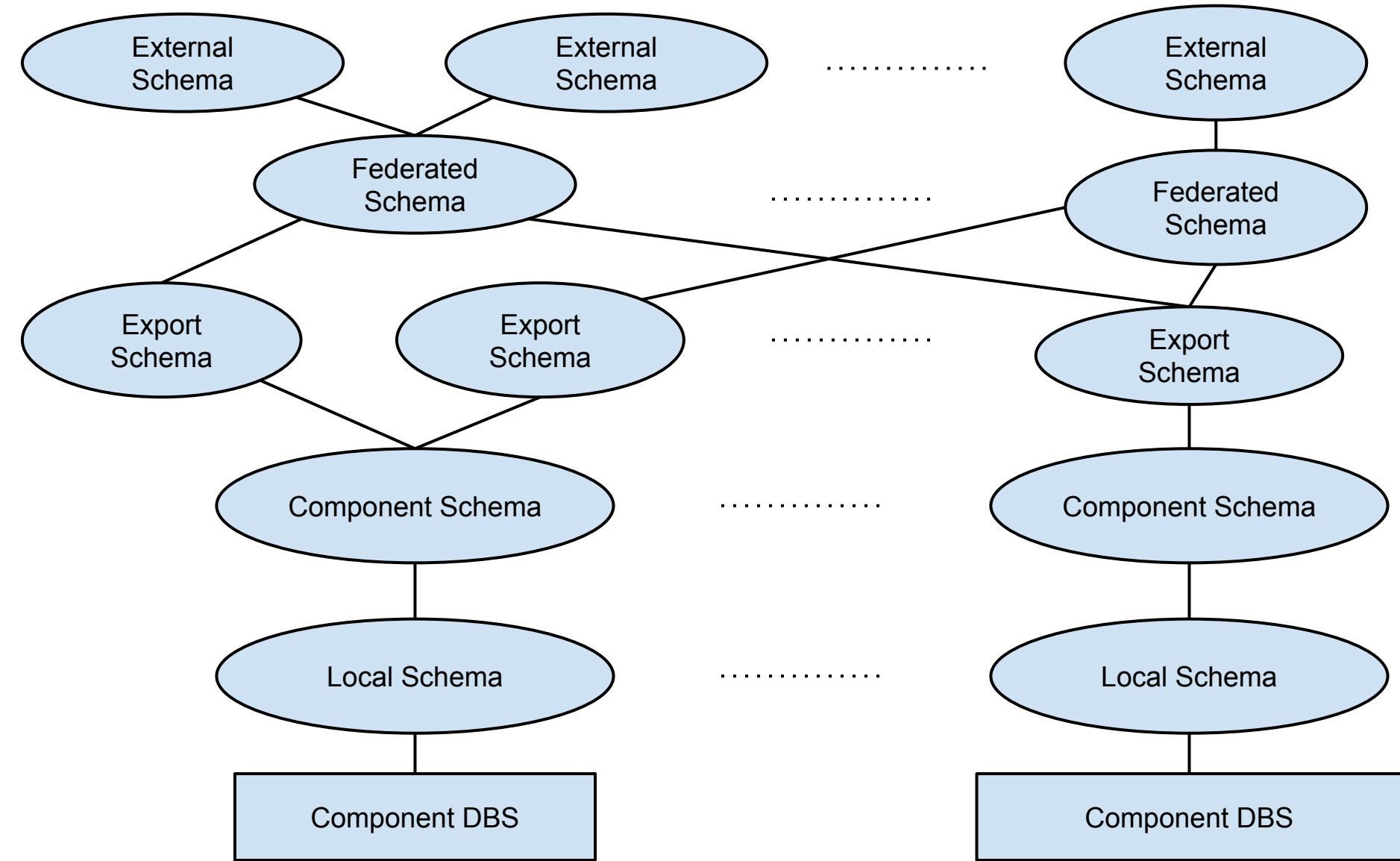
# Types of Distributed Database Systems

- Communication Autonomy
  - The ability to decide whether to communicate with another component DBS
- Execution Autonomy
  - The ability to execute local operations without interference from external operations of other component DBSs

# Types of Distributed Database Systems

- Association Autonomy
  - The ability to decide whether and how much to share its data and operations with other component DBSs
- The major challenge in designing FDBs is to let component DBSs to interoperate while still providing the above types of autonomies to them

# FDBS - Five Level Schema Architecture



# FDBS - Five Level Schema Architecture

- The local schema is the conceptual schema (full database definition) of a component database
- Component schema is derived by translating the local schema into a canonical data model or common data model ( CDM ) for the FDBS.
- The export schema represents the subset of a component schema that is available to the FDBS
- The federated schema is the global schema or view , which is the result of integrating all the shareable export schemas
- The external schemas define the schema for a user group or an application

# Query Processing in Distributed Databases

- Data transfer costs of distributed query processing
- Distributed Query Processing Using Semijoin
- Query and Update Decomposition

# Query Processing in Distributed Databases

- Data transfer costs of distributed query processing
  - These costs may not be very high if the sites are connected via a high performance network
  - They become quite significant in other types of networks
  - Hence DDBMS query optimisation algorithms consider the goal of reducing the amount of data transfer to reduce the total data transfer costs

# Query Processing in Distributed Databases

- Example

Site 1

EMPLOYEE

| FNAME | MINIT | LNAME | <u>SSN</u> | BDATE | ADDRESS | SEX | SALARY |
|-------|-------|-------|------------|-------|---------|-----|--------|
|-------|-------|-------|------------|-------|---------|-----|--------|

15 bytes

15 bytes 9 bytes

|          |     |
|----------|-----|
| SUPERSSN | DNO |
|----------|-----|

4 bytes

10000 records

record length 100 bytes

# Query Processing in Distributed Databases

## Site 2

### DEPARTMENT

| DNAME | <u>DNUMBER</u> | MGRSSN | MGRSTARTDATE |
|-------|----------------|--------|--------------|
|-------|----------------|--------|--------------|

0 bytes

4 bytes

9 bytes

100 records

record length 35 bytes

# Query Processing in Distributed Databases

## Query 1

- " For each employee, retrieve the employee name and the name of the department for which the employee works "
- submitted at site 3 which is the result site

# Query Processing in Distributed Databases

## Query 1

- " For each employee, retrieve the employee name and the name of the department for which the employee works "

## Query 1 in Relational Algebra

$$\Pi_{\text{FNAME}, \text{LNAME}, \text{DNAME}} (\text{EMPLOYEE} \bowtie_{\text{DNO} = \text{DNUMBER}} \text{DEPARTMENT})$$

# Query Processing in Distributed Databases

## Strategy 1

- Transfer both EMPLOYEE and DEPARTMENT relations to the result site , and perform the join at site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 1

- Transfer both EMPLOYEE and DEPARTMENT relations to the result site , and perform the join at site 3
- Total data transfer needed is

$$1,000,000 + 3500 = 1,003,500 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 2

- Transfer the EMPLOYEE relation to site 2 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 2

- Transfer the EMPLOYEE relation to site 2 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

$$1,000,000 + 4,00,000 = 1,400,000 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

$$3,500 + 4,00,000 = ,4,03,500 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is  
$$3,500 + 4,00,000 = ,4,03,500 \text{ bytes}$$
- Optimum strategy

# Query Processing in Distributed Databases

## Query 2

- " For each department, retrieve the department name and the name of the department manager "

## Query 2 in Relational Algebra

$$\Pi_{\text{DNAME}, \text{FNAME}, \text{LNAME}} (\text{EMPLOYEE} \bowtie_{\text{SSN} = \text{MGRSSN}} \text{DEPARTMENT})$$

# Query Processing in Distributed Databases

## Strategy 1

- Transfer both EMPLOYEE and DEPARTMENT relations to the result site , and perform the join at site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 1

- Transfer both EMPLOYEE and DEPARTMENT relations to the result site , and perform the join at site 3
- Total data transfer needed is

$$1,000,000 + 3500 = 1,003,500 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 2

- Transfer the EMPLOYEE relation to site 2 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 2

- Transfer the EMPLOYEE relation to site 2 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

$$1,000,000 + 4,000 = 1,004,000 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

$$3,500 + 4,000 = 7,500 \text{ bytes}$$

# Query Processing in Distributed Databases

## Strategy 3

- Transfer the DEPARTMENT relation to site 1 , perform the join there, and transfer the result to site 3
- Total data transfer needed is

$$3,500 + 4,000 = 7,500 \text{ bytes}$$

- Optimum strategy

# Query Processing in Distributed Databases

## Assumption

- The result site is site 2

# Query Processing in Distributed Databases

## Strategy 1

- Transfer EMPLOYEE relation to the site 2 , perform the join there, and present the result to the user
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 1

- Transfer EMPLOYEE relation to the site 2 , perform the join there, and present the result to the user
- Total data transfer needed is  
1,000,000 bytes ( for both queries )

# Query Processing in Distributed Databases

## Strategy 2

- Transfer DEPARTMENT relation to the site 1 , perform the join there, and transfer back the result to the site 2
- Total data transfer needed is

# Query Processing in Distributed Databases

## Strategy 2

- Transfer DEPARTMENT relation to the site 1 , perform the join there, and transfer back the result to the site 2
- Total data transfer needed is

$$3,500 + 4,00,000 = 4,03,500 \text{ bytes (query 1)}$$

# Query Processing in Distributed Databases

## Strategy 2

- Transfer DEPARTMENT relation to the site 1 , perform the join there, and transfer back the result to the site 2
- Total data transfer needed is

$$3,500 + 4,00,000 = 4,03,500 \text{ bytes (query 1)}$$

$$3,500 + 4,000 = 7,500 \text{ bytes (query 2)}$$

# Query Processing in Distributed Databases

## Strategy 2

- Transfer DEPARTMENT relation to the site 1 , perform the join there, and transfer back the result to the site 2
- Total data transfer needed is

$$3,500 + 4,00,000 = 4,03,500 \text{ bytes (query 1)}$$

$$3,500 + 4,000 = 7,500 \text{ bytes (query 2)}$$

- Optimum Strategy

# Query Processing in Distributed Databases

- Distributed query processing using semijoin
  - The idea behind using semijoin operation is to reduce the number of tuples in a relation before transferring it to another site

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

The following are the different steps in this operation  
( R semijoin S )

- We send the joining column of S to the site where R is located. This column is then joined with R
- The attributes required in the result, are projected out and sent back to the original site and joined with S

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

Illustration of this strategy for the above queries

step 1

- Project the join attributes of DEPARTMENT at site 2, and transfer them to site 1
- For the first query, we transfer

$F = \Pi_{DNUMBER} (\text{DEPARTMENT})$  whose size is  $4 * 100 = 400$  bytes

- For the second query, we transfer

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

Illustration of this strategy for the above queries

step 2

- Join the transferred file with the EMPLOYEE relation at site 1 and transfer the required attributes from the resulting file to site 2
- For the first query, we transfer

$$R = \Pi_{DNO, FNAME, LNAME} (F \bowtie_{DNUMBER = DNO} EMPLOYEE)$$

whose size is  $34 * 10,000 = 3,40,000$  bytes

- For the second query, we transfer

$$R' = \Pi_{MGRSSN, FNAME, LNAME} (F' \bowtie_{MGRSSN = SSN} EMPLOYEE)$$

whose size is  $39 * 100 = 3900$  bytes

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

Illustration of this strategy for the above queries

step 3

- Join the transferred file  $R$  or  $R'$  with DEPARTMENT, and present the result to the user at site 2

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

Illustration of this strategy for the above queries

- Using this strategy, we transfer 3,40,400 bytes for query 1 and 4800 bytes for query 2
- We can see that this method reduces the amount of data transfer

# Query Processing in Distributed Databases

- Distributed query processing using semijoin

## SemiJoin Operation - Defn

- A semijoin operation between R and S is indicated by

$$R \times_{A=B} S$$

where A and B are domain compatible attributes of R and S

- The semijoin operation is not commutative

$$R \times_S \neq S \times_R$$

# Query Processing in Distributed Databases

- Query and Update Decomposition
  - In a DDBMS, data is distributed across various fragments
  - Each site holds a set of fragments
  - A user can submit a query at any of the sites

# Query Processing in Distributed Databases

- Query and Update Decomposition
    - For queries, a query decomposition module must breakup or decompose a query into subqueries that can be executed at the individual sites
    - In addition, a strategy for combining the results of the subqueries to form the query result is generated
- For updates, the DDBMS is responsible for maintaining consistency among replicated data items by using one of the distributed concurrency control algorithms

# Query Processing in Distributed Databases

- Query and Update Decomposition
  - To determine which replicas include the data items referenced in a query, the DDBMS refers to the fragmentation, replication and distribution information stored in the DDBMS catalogue
  - For vertical fragmentation, the attribute list for each fragment is kept in the catalogue
  - For horizontal fragmentation, a condition called guard is kept for each fragment

# Query Processing in Distributed Databases

- Query and Update Decomposition
  - This is basically a selection condition that specifies which tuples exist in the fragment
  - It is called a guard because only those tuples that satisfy this condition are allowed to be stored in the fragment
  - For mixed fragmentation, both the attribute list and the guard condition are kept in the catalogue

# Query Processing in Distributed Databases

- Example - Fragments

Site 1

## EMPLOYEE

|          |       |       |            |       |         |     |        |
|----------|-------|-------|------------|-------|---------|-----|--------|
| FNAME    | MINIT | LNAME | <u>SSN</u> | BDATE | ADDRESS | SEX | SALARY |
| SUPERSSN |       | DNO   |            |       |         |     |        |

## DEPARTMENT

|       |                |        |              |
|-------|----------------|--------|--------------|
| DNAME | <u>DNUMBER</u> | MGRSSN | MGRSTARTDATE |
|-------|----------------|--------|--------------|

# Query Processing in Distributed Databases

- Example - Fragments

Site 1

DEPT\_LOCATIONS

| <u>DNUMBER</u> | <u>DLOCATION</u> |
|----------------|------------------|
|                |                  |

WORKS\_ON

| ESSN | <u>PNO</u> | HOURS |
|------|------------|-------|
|      |            |       |

PROJECT

| PNAME | <u>PNUMBER</u> | PLOCATION | DNUM |
|-------|----------------|-----------|------|
|       |                |           |      |

attribute list : \* (all attributes )

guard condition : TRUE (all tuples )

# Query Processing in Distributed Databases

- Example - Fragments

Site 2

EMPD5

attribute list : FNAME, MINIT, LNAME, SSN,  
SALARY, SUPERSSN, DNO

guard condition : DNO = 5

# Query Processing in Distributed Databases

- Example - Fragments

## Site 2

### DEP5

attribute list : \*

guard condition : DNUMBER = 5

### DEP5\_LOCS

attribute list : \*

guard condition : DNUMBER = 5

# Query Processing in Distributed Databases

- Example - Fragments

## Site 2

### PROJS5

attribute list : \*

guard condition : DNUM = 5

### WORKS\_ON5

attribute list : \*

guard condition : ESSN IN (  $\Pi_{\text{SSN}}$  ( EMPD5 ) ) OR  
PNO IN (  $\Pi_{\text{PNUMBER}}$  ( PROJS5 ) )

# Query Processing in Distributed Databases

- Example - Fragments

Site 3

EMPD4

attribute list : FNAME, MINIT, LNAME, SSN,  
SALARY, SUPERSSN, DNO

guard condition : DNO = 4

# Query Processing in Distributed Databases

- Example - Fragments

Site 3

DEP4

attribute list : \*

guard condition : DNUMBER = 4

DEP4\_LOCS

attribute list : \*

guard condition : DNUMBER = 4

# Query Processing in Distributed Databases

- Example - Fragments

## Site 3

### PROJS4

attribute list : \*

guard condition : DNUM = 4

### WORKS\_ON4

attribute list : \*

guard condition : ESSN IN (  $\Pi_{\text{SSN}} (\text{EMPD4})$  ) OR  
PNO IN (  $\Pi_{\text{PNUMBER}} (\text{PROJS4})$  )

# Query Processing in Distributed Databases

- When the DDBMS decomposes an update request, it can determine which fragments must be updated by examining their guard conditions
- A user request to insert a new EMPLOYEE tuple  
< 'Rajesh', 'P', 'Raj', '123456789', '22-Apr-74' , ' Anjali, Punkunnam, Thrissur' , M, 50000, '987654321', 4 >  
would be decomposed by the DDBMS into two insert requests
- The first inserts the above tuple in the EMPLOYEE fragment at site 1
- The second inserts the projected tuple  
< 'Rajesh', 'P', 'Raj', '123456789', 50000, '987654321', 4 >  
in the EMPD4 fragment at site 3

# Query Processing in Distributed Databases

## Query

Retrieve the names and hours per week for each employee who works on some project controlled by department 5

## Query in SQL

```
SELECT FNAME, LNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE DNUM = 5 AND PNUMBER = PNO AND
 ESSN = SSN
```

- suppose that query is submitted at site 2 and result is needed there

# Query Processing in Distributed Databases

## Strategy 1

- The DDBMS can determine about query decomposition by looking at query conditions and guard conditions
- Here we can see that tuples corresponding to the conditions  $DNUM = 5$  AND  $PNUMBER = PNO$  reside at site 2

# Query Processing in Distributed Databases

## Strategy 1

- Hence we can decompose the query into the following relational algebra queries

$$T1 \leftarrow \Pi_{ESSN} (\text{PROJS5} \bowtie_{PNUMBER = PNO} \text{WORKS\_ON5})$$
$$T2 \leftarrow \Pi_{ESSN,FNAME,LNAME} ( T1 \bowtie_{ESSN = SSN} \text{EMPLOYEE} )$$
$$\text{RESULT} \leftarrow \Pi_{FNAME, LNAME, HOURS} ( T2 \bowtie_{ } \text{WORKS\_ON5} )$$

# Query Processing in Distributed Databases

## Strategy 1

- This decomposition can be used to execute the above query using the semijoin strategy
- Subquery T1 is executed at site 2, and the projected column ESSN can be sent to site 1
- Subquery T2 can then be executed at site 1, and the result can be sent back to site 2
- The final query result is calculated and displayed at site 2

# Query Processing in Distributed Databases

## Strategy 2

- Send the query itself to site 1, where all the database tuples are located
  - The query is executed locally there
  - The result is calculated and sent back to site 2
- 
- The query optimiser will calculate the costs of both strategies and will decide the one with the lower cost estimate

# Concurrency Control For Distributed Databases

- These techniques extend the locking technique used in centralised databases
- Once a transaction obtains a read lock on a data item, it can access any copy of that data item
- For a write lock, after updating a data item, the DDBMS is responsible for updating all copies of the data item before releasing the lock

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
- Distributed Concurrency Control Based On Voting

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Here a particular copy of a data item is designated as a distinguished copy
  - All the locking and unlocking requests are associated with this copy, and are sent to the site which contains this copy
  - This site acts as the coordinator site for concurrency control on that item

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Site Technique
  - Primary Site With Backup Site
  - Primary Copy Technique

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Site Technique
    - Here all distinguished copies of various data items are stored in a single site called primary site
    - All locking and unlocking requests are sent to this site

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
    - Primary Site Technique
- advantage
- It is not overly complex, as it is a simple extension of the centralised approach

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Site Technique

## disadvantages

- It might overload the site as all requests are made to this site
- The failure of the primary site paralyses the system, since all locking information is kept there

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Site With Backup Site
    - Here all locking information is kept at the primary site as well as the backup site  
In case of a primary site failure, the backup site takes over as primary site, and a new backup site is chosen

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
    - Primary Site With Backup Site
- advantage
- It simplifies the recovery process in case of primary site failure

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
    - Primary Site With Backup Site
- disadvantages
- It slows down the process of acquiring locks, since locking information need to be stored at both primary site and the backup site
  - overloading

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Copy Technique
    - Here distinguished copies of various data items are stored in different sites
    - It will improve reliability and availability, since failure of one site will affect only those transactions which access data items whose distinguished copies are stored in this site

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On A Distinguished Copy Of A Data Item
  - Primary Copy Technique
    - By using backup sites we can further improve reliability and availability

# Concurrency Control For Distributed Databases

- Choosing A New Coordinator Site in Case Of Failure
  - In the primary site technique with no backup, if the coordinator site fails, all transactions are aborted and restarted and a new coordinator site is chosen based on a process called election
  - The election algorithm is a complex one
  - Here the site which gets the majority of votes from other sites is chosen as the new coordinator site

# Concurrency Control For Distributed Databases

- Choosing A New Coordinator Site in Case Of Failure
  - If there is a backup site, it will become the coordinator site
  - It can choose the backup site from other sites
  - All transactions will be suspended before this process, and they will resume after it is over
  - If both primary and backup sites fail, we can use the election method to choose the new coordinator site

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On Voting
  - In this method there is no distinguished copy for data items
  - Each copy of a data item holds its own lock
  - Hence, for a transaction to lock a data item, it has to get approval from majority of copies of that data item

# Concurrency Control For Distributed Databases

- Distributed Concurrency Control Based On Voting
  - After getting the lock it informs other copies that it has been granted the lock
  - If a transaction does not get a majority of votes to grant a lock within a certain timeout period, it cancels the request and informs all sites about the cancellation

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- Full scale DDBMSs having all the functionalities we have discussed have not been developed
- Instead, distributed database applications are being developed in the context of client server architecture

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- In a typical DDBMS, software modules are divided into three levels
  - The server software
  - The client software
  - The communications software

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- The server software
  - It is responsible for local data management at a site, much like centralised DBMS software

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- The client software
  - It processes all requests that require access to more than one site
  - It handles all user interfaces
  - It ensures consistency of replicated copies of data items
  - It hides the details of data distribution from the user

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- The communications software
  - It provides the communication primitives that are used by the client to transmit command and data among the various sites as needed

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- The server
  - transaction server
  - database processor (DP)
  - back-end machine
- The client
  - application processor ( AP )
  - front-end machine

# An OverView Of Client Server Architecture and Its Relationship To Distributed Databases

- Interaction between client and server during the processing of a query
  - The client parses a user query and decomposes it into a number of independent site queries. Each site query is sent to the appropriate server site
  - Each server processes the local query and sends the resulting relation to the client site
  - The client site combines the results of the subqueries to produce the result of the originally submitted query

# Module III

## Deductive Databases

- Introduction
- Prolog Notation
- Datalog Notation
- Evaluation of Datalog Programs
- Evaluation of Non-recursive Datalog Queries
- Recursive Query Processing in Datalog
- Examples of Deductive Database Systems

# Deductive Databases - Introduction

- A deductive database system has the capabilities to define (deductive) rules
- Rules can deduce or infer additional information from the facts that are stored in the database
- The model used for deductive databases is closely related to the relational data model
- It is also related to the field of logic programming

# Deductive Databases - Introduction

- Facts are specified in a manner similar to the way relations are specified
- Rules are somewhat similar to relational views
- The main difference between rules and views is that rules may involve recursion
- Hence they can yield information which cannot be defined using views

# Deductive Databases - Introduction

- Prolog and Datalog are two programming languages used in deductive databases
- Datalog is more suitable for database applications
- Deductive Object Oriented Databases (DOODs) can incorporate object oriented features

# Prolog Notation

- In Prolog, every information corresponding to a deductive database is represented using a predicate
- A predicate has an implicit meaning, which is suggested by the predicate name, and a fixed number of arguments
- If the arguments are all constant values, the predicate simply states that a certain fact is true
- If, on the other hand, the predicate has variables as arguments, it is either considered as a query or as a rule or constraint

# Prolog Notation

- Constant names start with lower case letters
- Variable names start with an upper case letter

## examples

supervise ( franklin, john )                              ( fact )

superior ( X,Y) : - supervise ( X, Y)    ( rule )

superior ( james, Y) ?                                      (query)

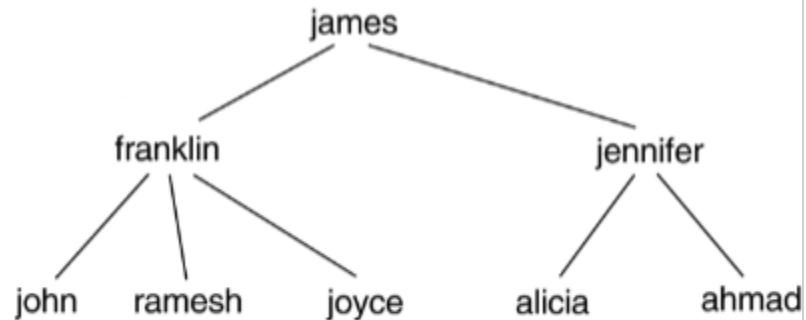
# (a) Prolog notation. (b) The supervisory tree.

(a)

Facts

```
supervise(franklin,john).
supervise(franklin,ramesh).
supervise(franklin,joyce).
supervise(jennifer,alicia).
supervise(jennifer,ahmad).
supervise(james,franklin).
supervise(james,jennifer).
...
```

(b)



Rules

```
superior(X,Y) :- supervise(X,Y).
superior(X,Y) :- supervise(X,Z), superior(Z,Y).
subordinate(X,Y) :- superior(Y,X).
```

Queries

```
superior(james,Y)?
superior(james,joyce)?
```

# Prolog Notation

- The actual data is stored in the relation SUPERVISE with two attributes whose schema is  
 $\text{SUPERVISE}(\text{supervisor}, \text{supervisee})$
- A predicate with constants as arguments is said to be ground or an instantiated predicate
- A rule is of the form  
 $\text{head} : - \text{body}$
- A head usually contains a single predicate
- The body can contain one or more predicates

# Prolog Notation

## Some Derived Facts

superior (franklin, john)

superior (franklin, ramesh)

superior (franklin, joyce)

superior (jennifer, alicia)

superior (jennifer, ahmed)

superior (james, franklin)

superior (james, jennifer)

subordinate(john, franklin)

subordinate(ramesh, franklin)

subordinate(joyce, franklin)

subordinate(alicia, jennifer)

subordinate(ahmed, jennifer)

subordinate(franklin, james)

subordinate(jennifer, james)

# Prolog Notation

Answer to query 1

Y = franklin, jennifer, john, ramesh, joyce, alicia, ahmad

Answer to query 2

True

# Datalog Notation

- A program is built from basic objects called atomic formulas
- Atomic formulas are literals of the form

$$P( a_1, a_2, a_3 \dots a_n )$$

- P is the predicate name
- $a_1, a_2, a_3 \dots a_n$  are the arguments for the predicate
- The number of arguments n is called the arity or degree of P
- The arguments can be either constant values or variable names
- The convention for naming constants and variables is same as that used in Prolog

# Datalog Notation

- We can use built-in predicates
- Binary comparison predicates ( over ordered domains )

< ( less )

<= ( less\_ or \_ equal )

> ( greater )

>= ( greater \_ or \_ equal )

- Comparison predicates ( over ordered / unordered domains)

= ( equal )

/ = ( not \_ equal )

# Datalog Notation

- They can be used like other predicates as

less ( X, 3 )

- They can also be represented using infix notation as

X < 3

- Built-in predicates can be used in Prolog also
- A literal which is preceded by **not** is called a negative literal
- Others are positive literals
- Atomic formulas are converted into Clausal Form

# Clausal Form and Horn Clauses

- A clausal form has the following characteristics
- All variables in the formula are implicitly quantified by the universal quantifier ( for all )
- The formula is made up of a number of clauses
- Each clause is composed of a number of literals connected by OR logical connectives only
- Hence, each clause is a disjunction of literals
- The clauses themselves are connected by AND logical connectives only, to form a formula
- Hence, the clausal form of a formula is a conjunction of clauses
- Any formula can be converted into clausal form

# Clausal Form and Horn Clauses

- Consider a clause of the form

$$\text{not} ( P_1 ) \text{ OR not} ( P_2 ) \text{ OR ..... not} ( P_n ) \text{ OR} \\ Q_1 \text{ OR } Q_2 \text{ OR ..... OR } Q_m$$

- This clause has **n negative literals** and **m positive literals**
- The equivalent logical formula is

$$P_1 \text{ AND } P_2 \text{ AND ..... AND } P_n \Rightarrow Q_1 \text{ OR } Q_2 \text{ OR .... OR } Q_m$$

# Clausal Form and Horn Clauses

- A **Horn clause** is one which contain **at most one positive literal**

$\text{not} ( P_1 ) \text{ OR } \text{not} ( P_2 ) \text{ OR } \dots \text{ not} ( P_n ) \text{ OR } Q$  ( form 1)

$\text{not} ( P_1 ) \text{ OR } \text{not} ( P_2 ) \text{ OR } \dots \text{ not} ( P_n )$  ( form 2)

- Transformed clause

$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \rightarrow Q$  ( form 1)

$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \rightarrow$  ( form 2)

# Clausal Form and Horn Clauses

- Datalog Rule

$Q : - P_1, P_2 \dots, P_n$  ( form 1)

$P_1, P_2 \dots, P_n$  ( form 2)

- This datalog expression can be considered as an **integrity constraint**, where all the predicates must be true to satisfy the query **(form 2)**

# Clausal Form and Horn Clauses

- A prolog / datalog system has an internal inference engine that can be used to process and compute the results of queries
- Prolog inference engine returns one result to the query at a time and must be prompted to give additional results
- Datalog inference engine returns a set of results at a time

# Interpretation of Rules

- There are two methods for interpreting the theoretical meaning of rules
  - Proof - theoretic interpretation
  - Model - theoretic interpretation

# Interpretation of Rules

- Proof-theoretic interpretation
  - Here, we consider the facts and rules to be true statements or axioms
  - Ground axioms contain no variables
  - Facts are ground axioms that are given to be true
  - Rules are called deductive axioms, since they can be used to derive new facts
  - Using this method, we can prove whether a certain fact is true or not

# Interpretation of Rules

- Proof-theoretic interpretation
  - To prove the fact `superior(james, ahmad)`

|                                                                 |                           |
|-----------------------------------------------------------------|---------------------------|
| 1. <code>superior(X,Y) :- supervise(X,Y).</code>                | (rule 1)                  |
| 2. <code>superior(X,Y) :- supervise(X,Z), superior(Z,Y).</code> | (rule 2)                  |
| 3. <code>supervise(jennifer,ahmad).</code>                      | (ground axiom, given)     |
| 4. <code>supervise(james,jennifer).</code>                      | (ground axiom, given)     |
| 5. <code>superior(jennifer,ahmad).</code>                       | (apply rule 1 on 3)       |
| 6. <code>superior(james,ahmad).</code>                          | (apply rule 2 on 4 and 5) |

# Interpretation of Rules

- Model-theoretic interpretation
  - Here, we specify the combinations of arguments that make the predicate true
  - Then, we state that all other combinations make the predicate false
  - This is done for every predicate

# Model theoretic interpretation to prove **superior(james,ahmad)**

## Rules

```
superior(X,Y) :- supervise(X,Y).
superior(X,Y) :- supervise(X,Z), superior(Z,Y).
```

## Interpretation

### *Known Facts:*

supervise(franklin,john) is **true**.  
supervise(franklin,ramesh) is **true**.  
supervise(franklin,joyce) is **true**.  
supervise(jennifer,alicia) is **true**.  
supervise(jennifer,ahmad) is **true**.  
supervise(james,franklin) is **true**.  
supervise(james,jennifer) is **true**.  
supervise(X,Y) is **false** for all other possible (X,Y) combinations.

### *Derived Facts:*

superior(franklin,john) is **true**.  
superior(franklin,ramesh) is **true**.  
superior(franklin,joyce) is **true**.  
superior(jennifer,alicia) is **true**.  
superior(jennifer,ahmad) is **true**.  
superior(james,franklin) is **true**.  
superior(james,jennifer) is **true**.  
superior(james,john) is **true**.  
superior(james,ramesh) is **true**.  
superior(james,joyce) is **true**.  
superior(james,alicia) is **true**.  
superior(james,ahmad) is **true**.  
superior(X,Y) is **false** for all other possible (X,Y) combinations.

# Basic Inference Mechanisms for Logic Programs

- Methods for evaluating a query based on proof-theoretic interpretation of rules
  - Bottom-up inference mechanisms ( forward chaining )
  - Top-down inference mechanisms(backward chaining)

# Basic Inference Mechanisms for Logic Programs

- Bottom-up inference mechanisms ( forward chaining )
  - The inference starts from the given facts
  - Generate additional facts by applying rules
  - These facts are matched to the goal of a query
  - The inference moves forward toward the goal

# Basic Inference Mechanisms for Logic Programs

- Bottom-up inference mechanisms ( forward chaining )
  - To evaluate the query superior ( james, Y ) ?
  - The basic facts are compared with the query to find a match
  - Since all the basic facts are for the supervise predicate, we have to look for the derived facts by applying rules
  - From the derived facts the computed answers to this query are

Y = franklin, Y = jennifer, Y = john, Y = ramesh

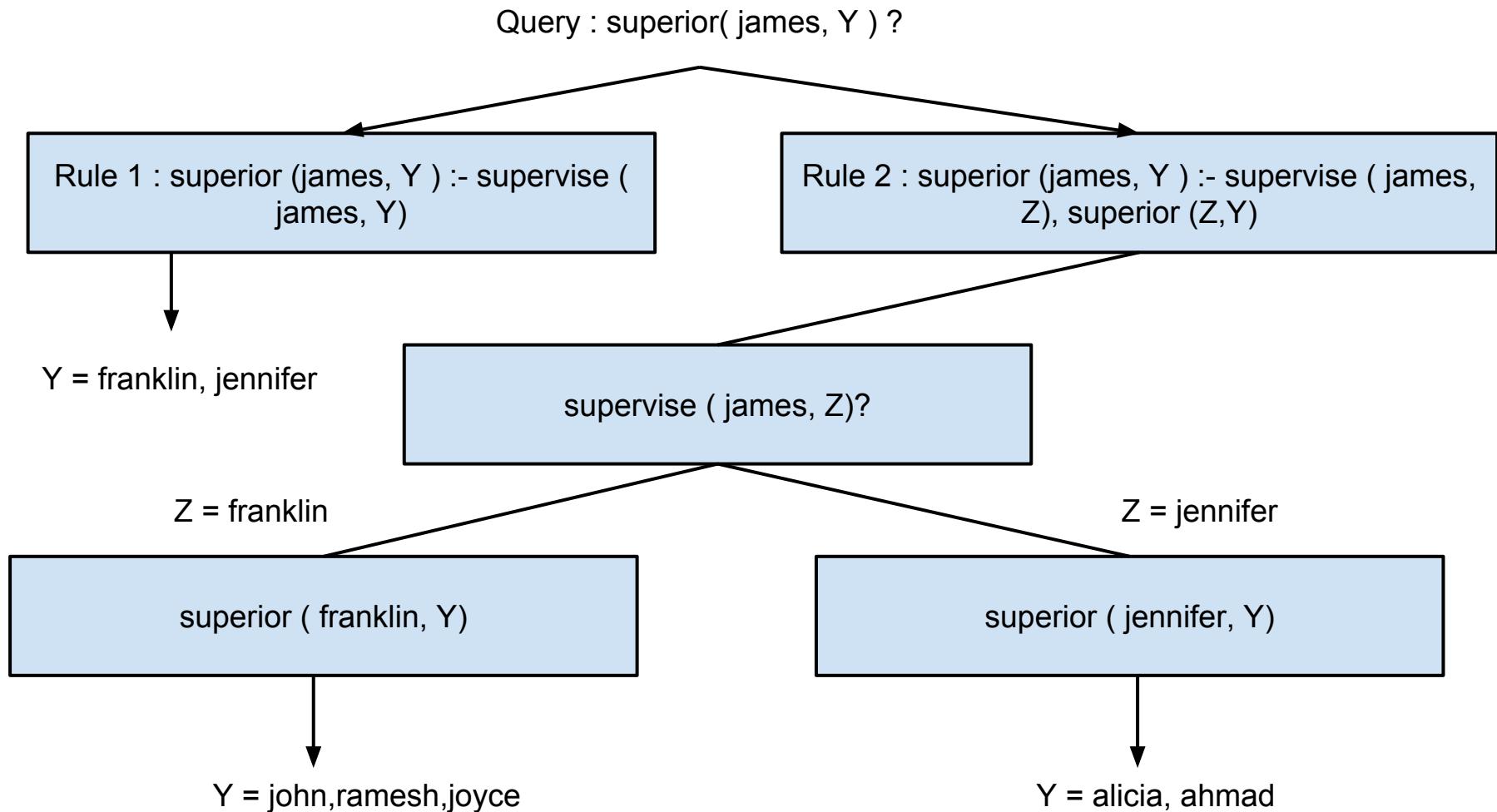
Y = joyce, Y = alicia, Y = ahmad

# Basic Inference Mechanisms for Logic Programs

- Top-down inference mechanisms ( backward chaining )
  - The inference starts from the goal of a query
  - We try to find facts that make the goal true
  - The inference moves backward from the goal towards facts that satisfy the goal

# Basic Inference Mechanisms for Logic Programs

- Top-down inference mechanisms ( backward chaining )



# Fact predicates for part of the database.

```
employee(john).
employee(franklin).
employee(alicia).
employee(jennifer).
employee(ramesh).
employee(joyce).
employee(ahmad).
employee(james).

salary(john,30000).
salary(franklin,40000).
salary(alicia,25000).
salary(jennifer,43000).
salary(ramesh,38000).
salary(joyce,25000).
salary(ahmad,25000).
salary(james,55000).

department(john,research).
department(franklin,research).
department(alicia,administration).
department(jennifer,administration).
department(ramesh,research).
department(joyce,research).
department(ahmad,administration).
department(james,headquarters).

supervise(franklin,john).
supervise(franklin,ramesh).
supervise(franklin,joyce).
supervise(jennifer,alicia).
supervise(jennifer,ahmad).
supervise(james,franklin).
supervise(james,jennifer).

male(john).
male(franklin).
male(ramesh).
male(ahmad).
male(james).

female(alicia).
female(jennifer).
female(joyce).

project(productx).
project(producty).
project(productz).
project(computerization).
project(reorganization).
project(newbenefits).

workson(john,productx,32).
workson(john,producty,8).
workson(ramesh,productz,40).
workson(joyce,productx,20).
workson(joyce,producty,20).
workson(franklin,producty,10).
workson(franklin,productz,10).
workson(franklin,computerization,10).
workson(franklin,reorganization,10).
workson(alicia,newbenefits,30).
workson(alicia,computerization,10).
workson(ahmad,computerization,35).
workson(ahmad,newbenefits,5).
workson(jennifer,newbenefits,20).
workson(jennifer,reorganization,15).
workson(james,reorganization,10).
```

# Rule-defined predicates.

superior(X,Y) :- supervise(X,Y).

superior(X,Y) :- supervise(X,Z), superior(Z,Y).

subordinate(X,Y) :- superior(Y,X).

supervisor(X) :- employee(X), supervise(X,Y).

over\_40K\_emp(X) :- employee(X), salary(X,Y), Y>=40000.

under\_40K\_supervisor(X) :- supervisor(X), not(over\_40\_K\_emp(X)).

main\_productx\_emp(X) :- employee(X), workson(X,productX,Y), Y>=20.

president(X) :- employee(X), not(supervise(Y,X)).

---

# Safety of Programs

- Defn- A program or a rule is said to be safe if it generates a finite set of facts

Consider the given below rules

1. `big_salary(Y) :- Y>60000` (not safe)
2. `big_salary(Y):-employee(X),salary(X,Y),Y>60000` (safe)
3. `has_something(X,Y) :- employee(X)` (not safe)

# Safety of Programs

## Defn –limited variable

A variable X is limited in a rule if

1. It appears in a regular(not built-in) predicate in the body of the rule
2. It appears in a predicate of the form  $X=c$  or  $c=X$  or  $(c1 \leq X \text{ and } X \leq c2)$  in the rule body where  $c1$ ,  $c2$  and  $c$  are constant values
3. It appears in a predicate of the form  $X=Y$  or  $Y=X$  in the rule body, where  $Y$  is a limited variable

A rule is said to be limited if all its variables are limited

# Predicates for illustrating relational operations

```
rel_one(A,B,C).
rel_two(D,E,F).
rel_three(G,H,I,J).
```

```
select_one_A_eq_c(X,Y,Z) :- rel_one(c,Y,Z).
select_one_B_less_5(X,Y,Z) :- rel_one(X,Y,Z), Y<5.
select_one_A_eq_c_and_B_less_5(X,Y,Z) :- rel_one(c,Y,Z), Y<5.
```

```
select_one_A_eq_c_or_B_less_5(X,Y,Z) :- rel_one(c,Y,Z).
select_one_A_eq_c_or_B_less_5(X,Y,Z) :- rel_one(X,Y,Z), Y<5.
```

```
project_three_on_G_H(W,X) :- rel_three(W,X,Y,Z).
```

```
union_one_two(X,Y,Z) :- rel_one(X,Y,Z).
union_one_two(X,Y,Z) :- rel_two(X,Y,Z).
```

```
intersect_one_two(X,Y,Z) :- rel_one(X,Y,Z), rel_two(X,Y,Z).
```

```
difference_two_one(X,Y,Z) :- rel_two(X,Y,Z), not(rel_one(X,Y,Z)).
```

```
cart_prod_one_three(T,U,V,W,X,Y,Z) :-
 rel_one(T,U,V), rel_three(W,X,Y,Z).
```

```
natural_join_one_three_C_eq_G(U,V,W,X,Y,Z) :-
 rel_one(U,V,W), rel_three(W,X,Y,Z).
```

---

# Evaluation of Non-recursive Datalog Queries

- We create a predicate dependency graph
- It keeps track of dependency among predicates in a deductive database

# Evaluation of Non-recursive Datalog Queries

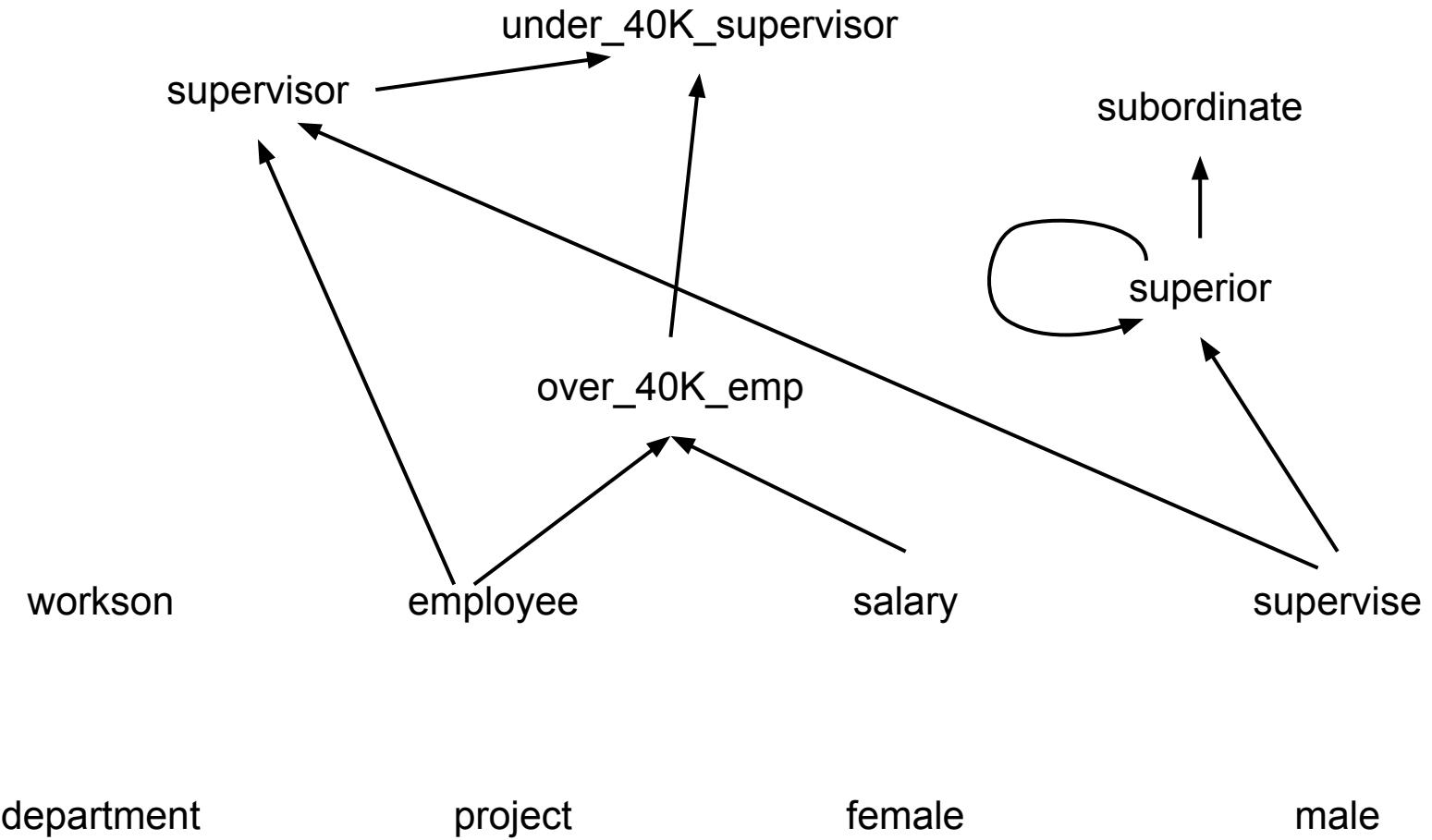
## Steps to create a predicate dependency graph

Each predicate is specified as a node in the predicate dependency graph

- Assume that predicate B is head of a rule ( LHS ) and predicate A is specified in the body ( RHS ) of a rule
- We say that B depends on A
- Draw a directed edge from A to B
- This indicates that, in order to compute the facts for predicate B ( rule head ), we must first compute the facts for all the predicate A ( rule body )

# Evaluation of Non-recursive Datalog Queries

Predicate Dependency Graph for The Organisation Database



# Evaluation of Non-recursive Datalog Queries

- If the dependency graph has no cycles, we call the rule set non recursive
- If there is at least one cycle, the rule set is called recursive
- Here, **superior** is a recursively defined predicate
- As **subordinate** predicate depends on **superior**, it also requires recursion in computing its result

# Evaluation of Non-recursive Datalog Queries

- The inference algorithm creates a relational expression for computing the result of the query
- The relational expression can be generated for the datalog query  $Q = P ( \text{arg 1}, \text{arg 2}, \dots, \text{arg n} )$  as follows

# Evaluation of Non-recursive Datalog Queries

## step 1

- Locate all rules  $S$  whose head involves the predicate  $P$
- If there are no such rules, then  $P$  is a fact defined predicate corresponding to some database relation  $R_p$
- In this case, one of the following expressions is returned and the algorithm is terminated
- We use the notation  $\$ i$  to refer to the name of the  $i^{\text{th}}$  attribute of relation  $R_p$ 
  - (a) If all arguments are distinct variables, the relational expression returned is  $R_p$

# Evaluation of Non-recursive Datalog Queries

## step 1

(b) If some arguments are constants or if the same variable appears in more than one argument position, then expression returned is

SELECT < condition > (  $R_p$  )

where the selection < condition > is a conjunctive condition made up of a number of simple conditions connected by AND, and constructed as follows.

( i ) if a constant c appears as argument i, include a simple condition ( \$ i = c ) in the conjunction

# Evaluation of Non-recursive Datalog Queries

## step 1

- ( ii ) if the same variable appears in both argument locations j and k , include a condition ( \$ j = \$ k) in the conjunction
- ( c ) For an argument that is not present in any predicate, a unary relation containing values that satisfy all conditions is constructed. Since the rule is assumed to be safe, this unary relation must be finite

# Evaluation of Non-recursive Datalog Queries

## step 2

- At this point, one or more rules  $S_i$ ,  $i = 1, 2, 3, \dots, n$ ,  $n > 0$  exist with predicate  $p$  as their head.
- For each rule  $S_i$ , generate a relational expression as follows:
  - ( a ) Apply selection operations on the predicates in the RHS for each such rule, as discussed in step 1
  - ( b ) A natural join is constructed among the relations that corresponds to the predicates in the body of the rule  $S_i$  over the common variables. For arguments that gave rise to the unary relations in step 1 (c) , the corresponding relations are brought as members into

# Evaluation of Non-recursive Datalog Queries

## step 2

the natural join. Let the resulting relation from this join be  $R_s$

- (c) If any built - in predicate  $X \theta Y$  was defined over the arguments  $X$  and  $Y$  , the result of the join is subjected to an additional selection:

$$\text{SELECT}_{X \theta Y} ( R_s )$$

- (d) Repeat step 2(b) until no more built-in predicates apply

# Evaluation of Non-recursive Datalog Queries

## step 3

- Take the UNION of the expressions generated in step 2  
( If more than one rule exists with predicate p as its head )

# Concepts of Recursive Query Processing in Datalog

- Pure Evaluation Approach
- Rule Rewriting Approach

# **Concepts of Recursive Query Processing in Datalog**

- Pure Evaluation Approach

creating a query evaluation plan that produces an answer to the query

- Rule Rewriting Approach

# **Concepts of Recursive Query Processing in Datalog**

- Pure Evaluation Approach

creating a query evaluation plan that produces an answer to the query

- Rule Rewriting Approach

optimising the plan into a more efficient strategy

# Concepts of Recursive Query Processing in Datalog

- Consider the following recursive query

`ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y)`

- This rule is in conjunction with the rule

`ancestor(X,Y) :- parent(X,Y)`

# Concepts of Recursive Query Processing in Datalog

- A rule is said to be linearly recursive, if the recursive predicate appears once and only once in the RHS of the rule

ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y) (left linearly  
recursive )

ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y) (right linearly  
recursive)

ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y)  
(not linearly recursive)

# **Concepts of Recursive Query Processing in Datalog**

- It is believed that most real-life rules can be described as linear recursive rules
- Algorithms have been defined to execute linear sets of rules efficiently

# Concepts of Recursive Query Processing in Datalog

- A predicate whose relation is stored in the database is called an **extensional database (EDB) predicate**
- A predicate for which the corresponding relation is defined by logical rules is called an **intensional database (IDB) predicate**

# Concepts of Recursive Query Processing in Datalog

- For evaluating queries we convert datalog rules to datalog equations
- Consider the following rules

$\text{path}(X,Y) :- \text{edge}(X,Y)$

$\text{path}(X,Y) :- \text{path}(X,Z), \text{path}(Z,Y)$

- These rules are converted into a single datalog equation

$$P(X,Y) = E(X,Y) \cup \Pi_{X,Y} ( P(X,Z) \bowtie P(Z,Y) )$$

# Concepts of Recursive Query Processing in Datalog

## Example

- Suppose that the nodes are 3,4, 5 and  $E = \{ (3, 4), (4, 5) \}$

∴

$$P = \{ (3, 4), (4, 5) \} \cup \Pi_{X,Y} (\{ (3,4), (4,5), (3,5) \} \bowtie \{ (3,4), (4,5), (3,5) \})$$

$$= \{ (3,4), (4,5), (3,5) \}$$

# Concepts of Recursive Query Processing in Datalog

- Naive Strategy
- Semi-naive Strategy

# Concepts of Recursive Query Processing in Datalog

- Naive Strategy
  - It is a **pure evaluation method**
  - It uses an iterative strategy
  - All rules are applied to a set of tuples at each iteration to deduce additional tuples
  - This is continued until no more tuples can be generated

# Concepts of Recursive Query Processing in Datalog

- Naive Strategy
  - Jacobi Strategy
  - Gauss- Seidel Strategy

# Concepts of Recursive Query Processing in Datalog

- Naive Strategy - Jacobi Strategy

## Datalog Rules

$\text{ancestor}(X,Y) :- \text{parent}(X,Y)$

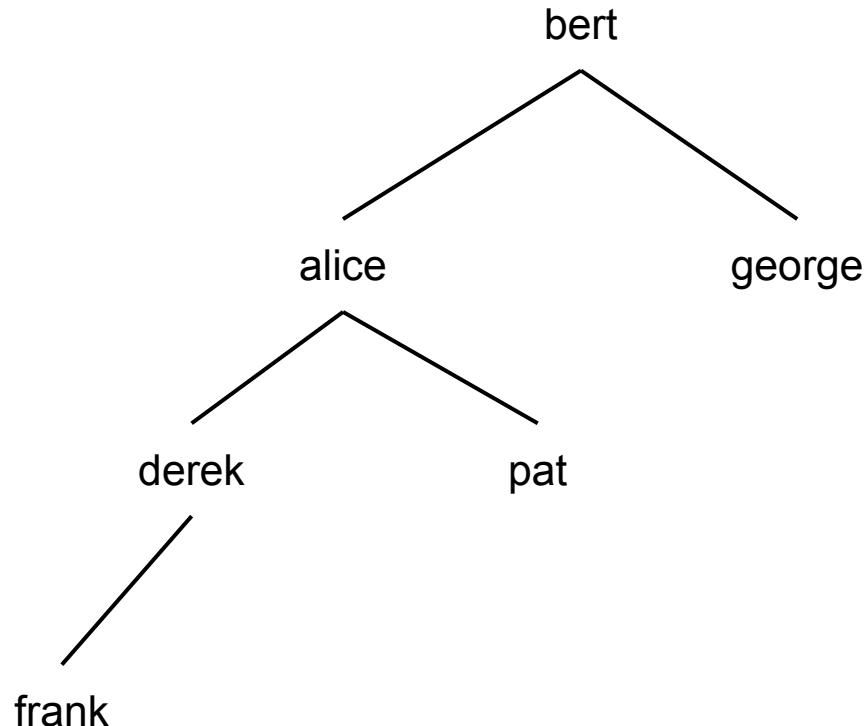
$\text{ancestor}(X,Y) :- \text{ancestor}(X,Z), \text{parent}(Z,Y)$

## Datalog Equation

$$A(X,Y) = P(X,Y) \cup \Pi_{X,Y} ( A(X,Z) \bowtie P(Z,Y) )$$

# Concepts of Recursive Query Processing in Datalog

Parent Tree - Example



# Concepts of Recursive Query Processing in Datalog

Here the EDB is

$$P = \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), (\text{derek}, \text{frank}) \}$$

# Concepts of Recursive Query Processing in Datalog

Here the EDB is

$$P = \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), (\text{derek}, \text{frank}) \}$$
$$A(1) = P = \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), (\text{derek}, \text{frank}) \}$$

# Concepts of Recursive Query Processing in Datalog

- The second iteration includes grandparents as ancestors besides parents

$$A(2) = P \cup \Pi_{X,Y}(A(1) \bowtie P)$$

= { (bert, alice), (bert, george), (alice,derek),  
( alice,pat), (derek, frank), (bert, derek),  
(bert, pat) , (alice, frank) }

# Concepts of Recursive Query Processing in Datalog

- The third iteration includes great grandparents as ancestors

$$A(3) = P \cup \Pi_{X,Y} ( A(2) \bowtie P )$$

= { (bert, alice), (bert, george), (alice,derek),  
( alice,pat), (derek, frank), (bert, derek),  
(bert, pat) , (alice, frank), (bert, frank) }

# Concepts of Recursive Query Processing in Datalog

- We can see that no more ancestors can be included
- Even after applying the rules, the next iteration yields the same tuples
- $A(4) = A(3)$  and the iterative process ends here

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy
  - It is an improvement on the naive strategy
  - It eliminates redundancy in calculations by reducing the number of join operations

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

Here the EDB is

$$P = \{ (bert, alice), (bert, george), (alice, derek), (alice, pat), (derek, frank) \}$$

- In addition to the relation A we use relation D for storing newly generated tuples in each iteration

$$D(1) = A(1) = P$$

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$D(2) = P \bowtie D(1)$$

$$\begin{aligned} D(2) &= \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), \\ &\quad (\text{derek}, \text{frank}) \} \bowtie \\ &\quad \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), \\ &\quad (\text{derek}, \text{frank}) \} \\ &= \{ (\text{bert}, \text{derek}), (\text{bert}, \text{pat}), (\text{alice}, \text{frank}) \} \end{aligned}$$

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$A(2) = A(1) \cup D(2)$$

= { (bert,alice),(bert,george),(alice,derek),(alice,pat),  
(derek, frank),(bert, derek),(bert, pat),(alice, frank) }

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$D(3) = P \bowtie D(2)$$

$$= \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), \\ (\text{derek}, \text{frank}) \} \bowtie$$

$$\{ (\text{bert}, \text{derek}), (\text{bert}, \text{pat}), (\text{alice}, \text{frank}) \}$$

$$= \{ (\text{bert}, \text{frank}) \}$$

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$A(3) = A(2) \cup D(3)$$

$= \{(bert,alice), (bert,george), (alice,derek), (alice,pat),$   
 $(derek,frank), (bert,derek), (bert,pat), (alice,frank), \textcolor{blue}{(bert,frank)}\}$

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$D(4) = P \bowtie D(3)$$

$$= \{ (\text{bert}, \text{alice}), (\text{bert}, \text{george}), (\text{alice}, \text{derek}), (\text{alice}, \text{pat}), \\ (\text{derek}, \text{frank}) \} \bowtie$$

$$\{ (\text{bert}, \text{frank}) \}$$

$$= \emptyset$$

# Concepts of Recursive Query Processing in Datalog

- Semi-Naive Strategy - For the above example

$$A(4) = A(3) \cup D(4)$$

$$= A(3) \cup \{ \Phi \}$$

$$= A(3)$$

- The evaluation ends here

# Concepts of Recursive Query Processing in Datalog

- The Magic Set Rule Rewriting Technique
  - It combines the advantages of both bottom-up and top-down evaluation
  - It was originally developed for evaluating recursive queries
  - It can be used for non recursive queries also
  - Further details of this technique are beyond our scope

# Concepts of Recursive Query Processing in Datalog

- Stratified Negation

A program is said to have **stratified negation** if it contains **rules with negative literals** where **no recursion is present**

# Concepts of Recursive Query Processing in Datalog

- Stratified Negation

## example-program

```
ancestor(X,Y) :- parent(X,Y)
```

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)
```

```
nocyc(X, Y) :- ancestor(X,Y), not(ancestor(Y,X))
```

# Concepts of Recursive Query Processing in Datalog

- Stratified Negation

advantage

Such programs can be efficiently evaluated

# Deductive Database Systems

- The LDL System
- NAIL!
- The CORAL System

# Deductive Database Systems

- The LDL System
  - Logic Data Language (LDL)
  - developed at MCC  
( Microelectronics and Computer technology Corporation)
  - uses a modified version of Prolog Language
  - supports complex terms through the use of function symbols called **functors**

# Deductive Database Systems

- The LDL System

## Defn - Employee Record

Employee ( Name (Rajesh),  
Job(Teacher),  
Education({(Govt HS School,Thrissur, 2000),  
(College(St Thomas College, Thrissur, BSc  
Physics), 2003),  
(College(MESCE, MCA), 2006)})  
)

# Deductive Database Systems

- NAIL!

- Not Another Implementation of Logic
- Developed at Stanford University
- Uses a language called GLUE

# Deductive Database Systems

- The CORAL System
  - Developed at the University of Wisconsin
  - Combines the important features of SQL and Prolog

# Deductive Database Systems

- The CORAL System

## Example - Rule

`budget ( Dname, sum ( <Sal> ) ) :- dept (Dname, Ename, Sal)`

# **Module IV**

- Data Warehousing
- Data Mining
- Databases on the World Wide Web
- Multimedia Databases and Mobile Databases
- Geographic Information Systems
- Digital Libraries

# Data Warehousing

- It is an infrastructure for storing historical data about an organisation
- These data can be used for decision making
- Post retrieval processing ( reporting ) is just as complex as the retrieval itself

# Data Warehousing

## Normal Database

- stores operational data
- regularly updated
- operational queries (transactions) are performed

## Data Warehouse

- stores historical data
- updated only over certain periods of time
- analytical queries are performed

# Data Warehousing

- Operational Data (OLTP Applications )
  - Data that “works”
  - Frequent updates and queries
  - Normalised for efficient search and updates  
(minimise update anomalies)
  - Fragmented and local relevance
  - Point queries : queries accessing individual tuples

# Data Warehousing

- Historical Data (OLAP Applications )
  - Data that “tells”
  - Very infrequent updates
  - Integrated data set with global relevance
  - Analytical queries that require huge amount of aggregation
  - Performance issues mainly in query response time ( not in updates )

# Data Warehousing

## Operational Queries - Examples

1. Retrieve the salary of Mr John
2. What is the average account balance of Mr Jennifer in July?
3. How many tickets have been sold from Thrissur Railway Station in September?

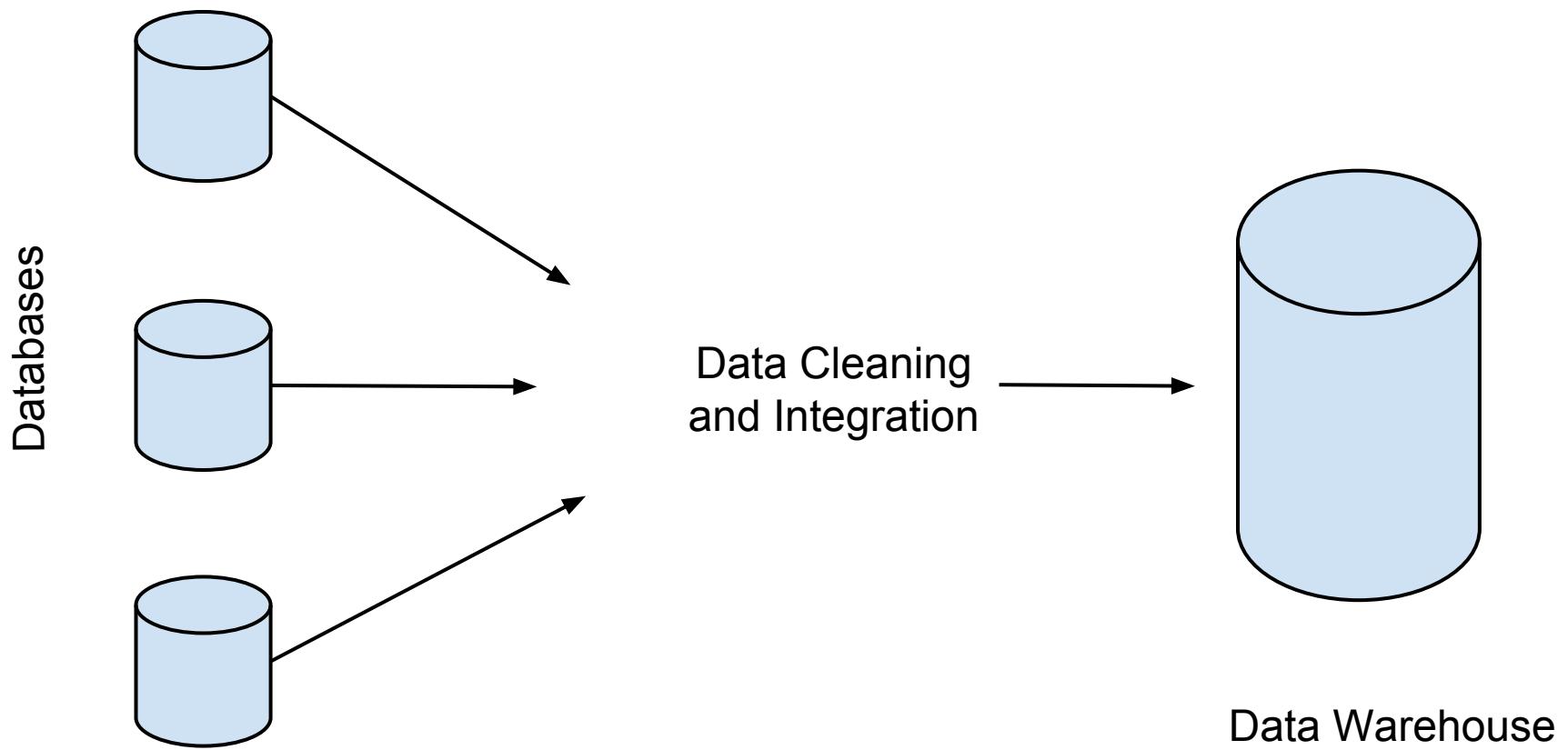
# Data Warehousing

## Analytical Queries - Examples

1. How does the attrition rate of employees of different ages change in a company during the last 10 years?
2. Retrieve the investment pattern of customers of various ages during the last 20 years
3. Compare the number of people travelling in express trains and passenger trains in various zones during the last 5 years

# Data Warehousing

Data Warehousing - Simple Illustration



# Data Warehousing

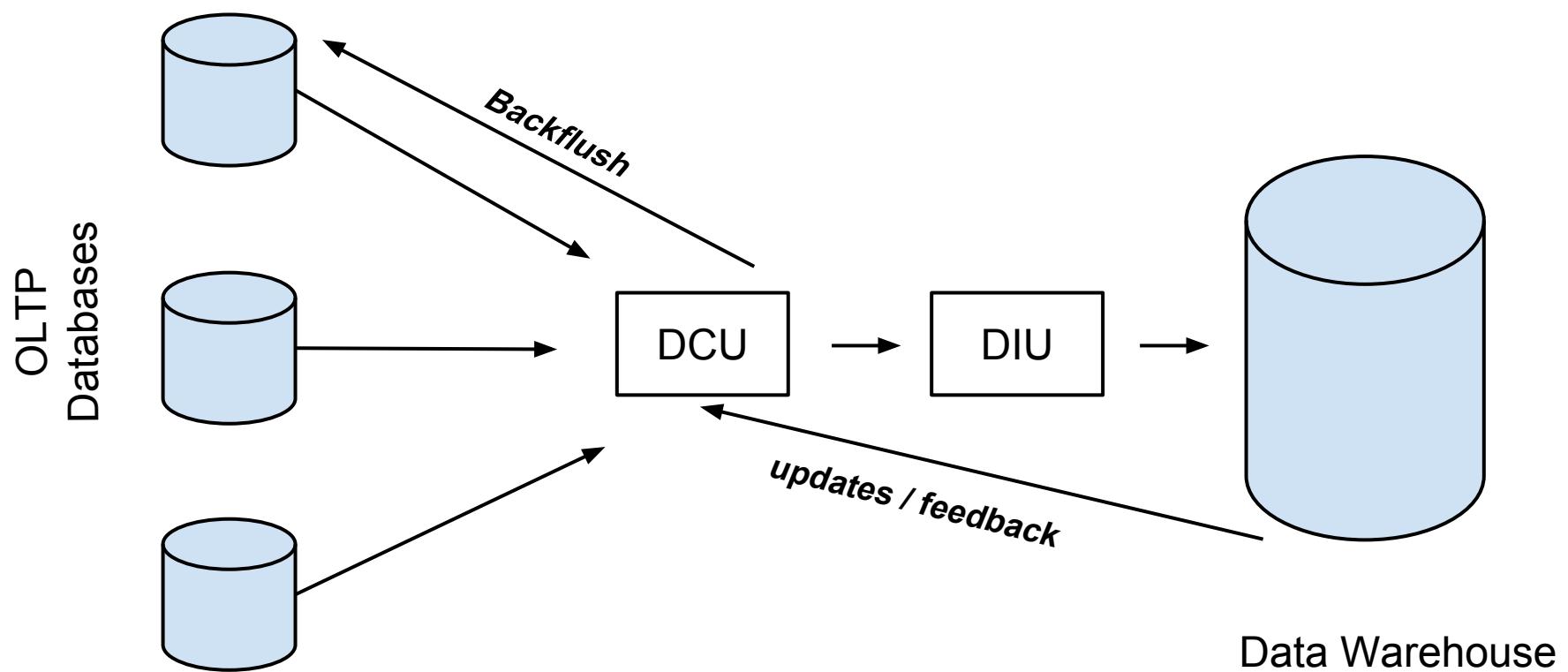
- The operational data in different databases are cleaned and then integrated before storing them in a data warehouse
- Data cleaning is the process of removing duplicate values and inconsistencies from operational data

# Data Warehousing

- Data warehouse is seen as a collection of data marts
- Data mart is a collection of historical data about a particular OLTP unit that feeds into the data warehouse
- Data marts are seen as small data warehouses where OLAP queries with respect to that segment are performed

# Data Warehousing

Data Warehousing - Another Illustration



# Data Warehousing

- The process of returning cleaned data to the source for updating is called backflushing.

# Data Warehousing

- Dirty Data
  - Lack of Standardisation
    - Multiple encodings(Unicode / ASCII), languages
    - Abbreviations
      - “Mahatma Gandhi Road” and “MG Road” are the same
    - Semantic equivalence
      - “Chennai” and “Madras” are the same
    - Multiple Measurement Systems
      - 1.6 kms is same as 1 mile

# Data Warehousing

- Dirty Data
  - Missing, Spurious, and Duplicate Data
    - Missing age field of an employee
    - Spurious (incorrectly entered) sales values
    - Duplication of data sets across OLTP units
    - Semantic duplication ( B L Sundar appearing in another data set as L Balasundar )

# Data Warehousing

- Dirty Data
  - Inconsistencies
    - Incorrect use of codes
      - Use of M/F in addition to 0/1 for Gender
    - Codes with inconsistent or outdated meaning
      - Travel eligibility “C” means eligibility to travel by IIIrd Class which no longer exists
    - Inconsistent duplicate data
      - Two data sets are found to belong to the same person but have different addresses

# Data Warehousing

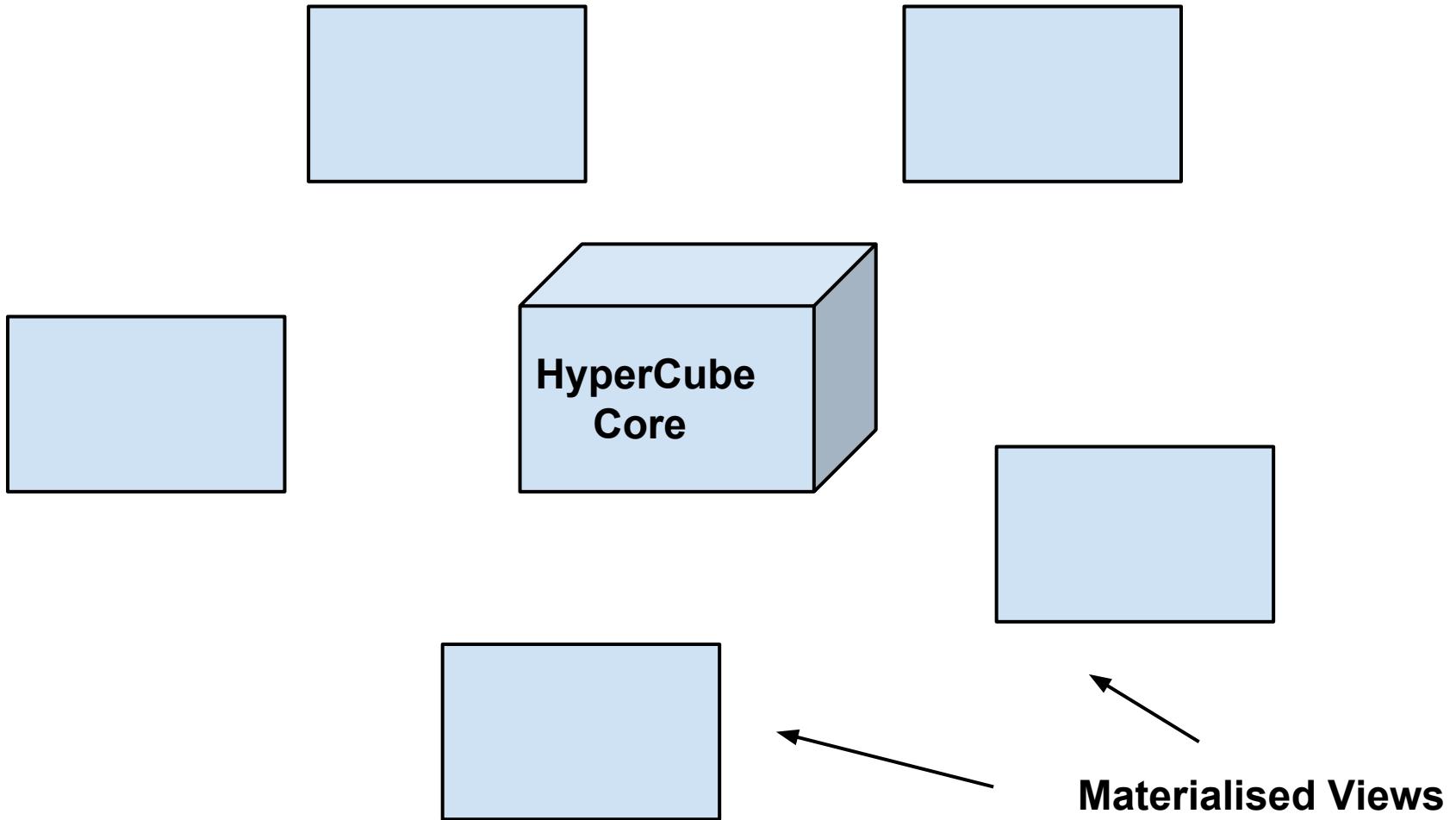
- Dirty Data
  - Inconsistencies
    - Inconsistent associations
      - Sales figures provided by the marketing department do not add up to the total sales figures by the retail units
    - Semantic Inconsistencies
      - Feb 31st
    - Referential inconsistency ( Rs 10 lakh sales reported from a unit which has been closed down)

# Data Warehousing

- Issues in Data Cleaning
  - Cannot be fully automated
  - Requires considerable knowledge that is beyond the purview of warehouse ( metrics, geography, govt policies )
  - Complexity increases with increase in data sources
  - Complexity increases with increase in history span taken for cleaning

# Data Warehousing

## A Typical Warehouse

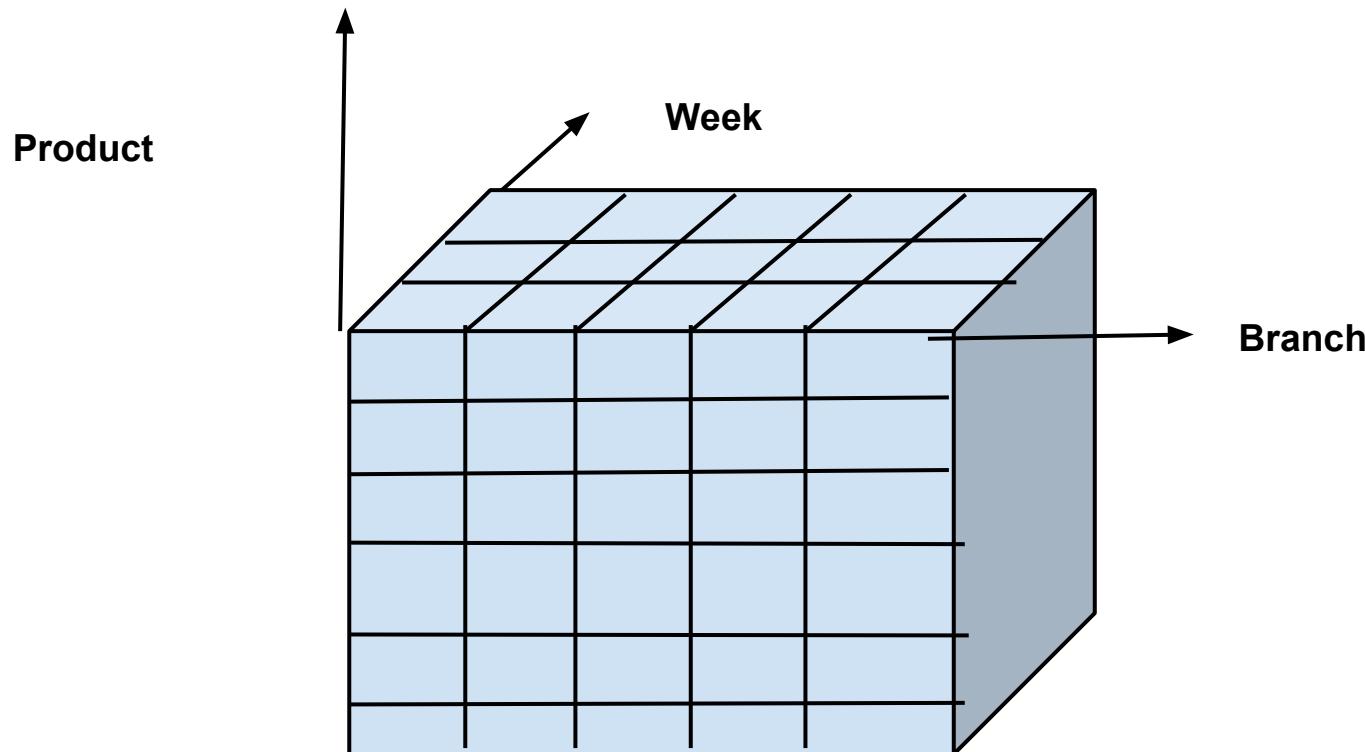


# Data Warehousing

- Hypercube Core
  - Manages the atomic data elements
  - Global schematic structure for the entire warehouse
  - Based on the multidimensional data model
- Materialised Views
  - Physical views for faster aggregate query answering
  - Denormalisation of the core

# Data Warehousing

## The Sales HyperCube



# Data Warehousing

- The Sales Hypercube
  - “Sales” is the fact
  - “Branch”, Product” and “week” are dimensions

# Data Warehousing

- Operations on Hyper Cubes
  - Pivoting: Choosing ( rotating the cube on a pivot) a set of dimensions for display
  - Slicing-dicing: Select some subset of the cube
  - Roll up: Aggregate a dimension to a larger dimension ( Roll up weeks into months )
  - Drill down: Open an aggregated dimension into reveal details ( Open up months to reveal weak by weak information )

# Data Warehousing

- Implementation of HyperCubes

- Relational OLAP (ROLAP)

- Maintain the data cube in a set of RDBMS tables

- eg: True ROLAP from Microstrategy Inc

- (<http://www.microstrategy.com>)

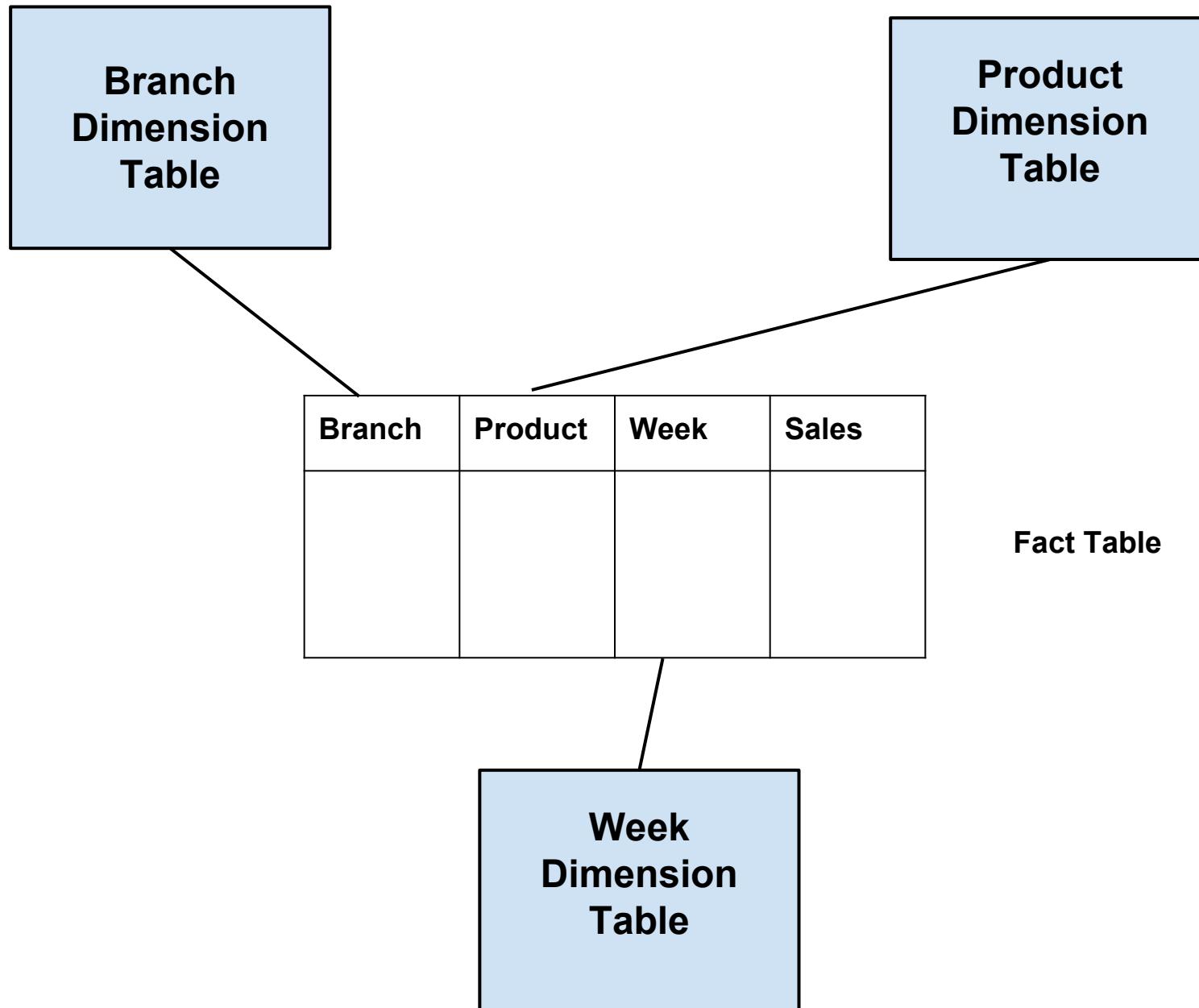
- Multidimensional OLAP (MOLAP)

- Use a separate storage model for multidimensional data

- eg: Arbor Essbase

- (<http://www.arborsoft.com>)

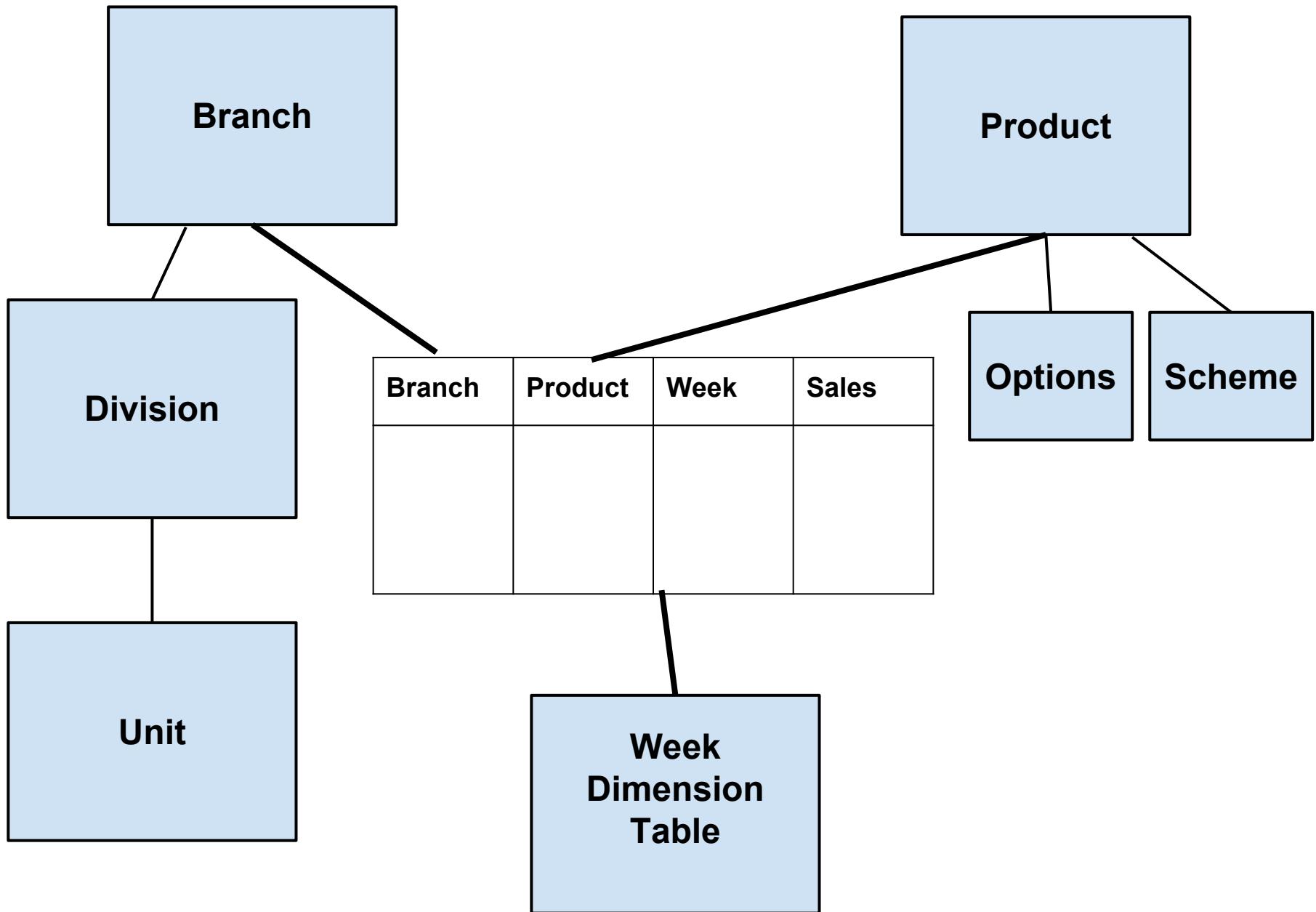
# Physical Models : Star Schema



# Data Warehousing

- Star Schema
  - Features
    - Central Fact Table
    - Set of supporting dimension tables
    - Denormalised data storage in dimension tables
  - Advantages
    - Simple to comprehend and design
    - Small meta data
    - Quick query responses
  - Limitations
    - Not robust towards changes
    - Enormous amount of redundancy in dimension table data

# Physical Models : Snowflake Schema



# Data Warehousing

- Snowflake Schema

- Features

- Central Fact Table
    - Normalised dimension tables storing atomic data units

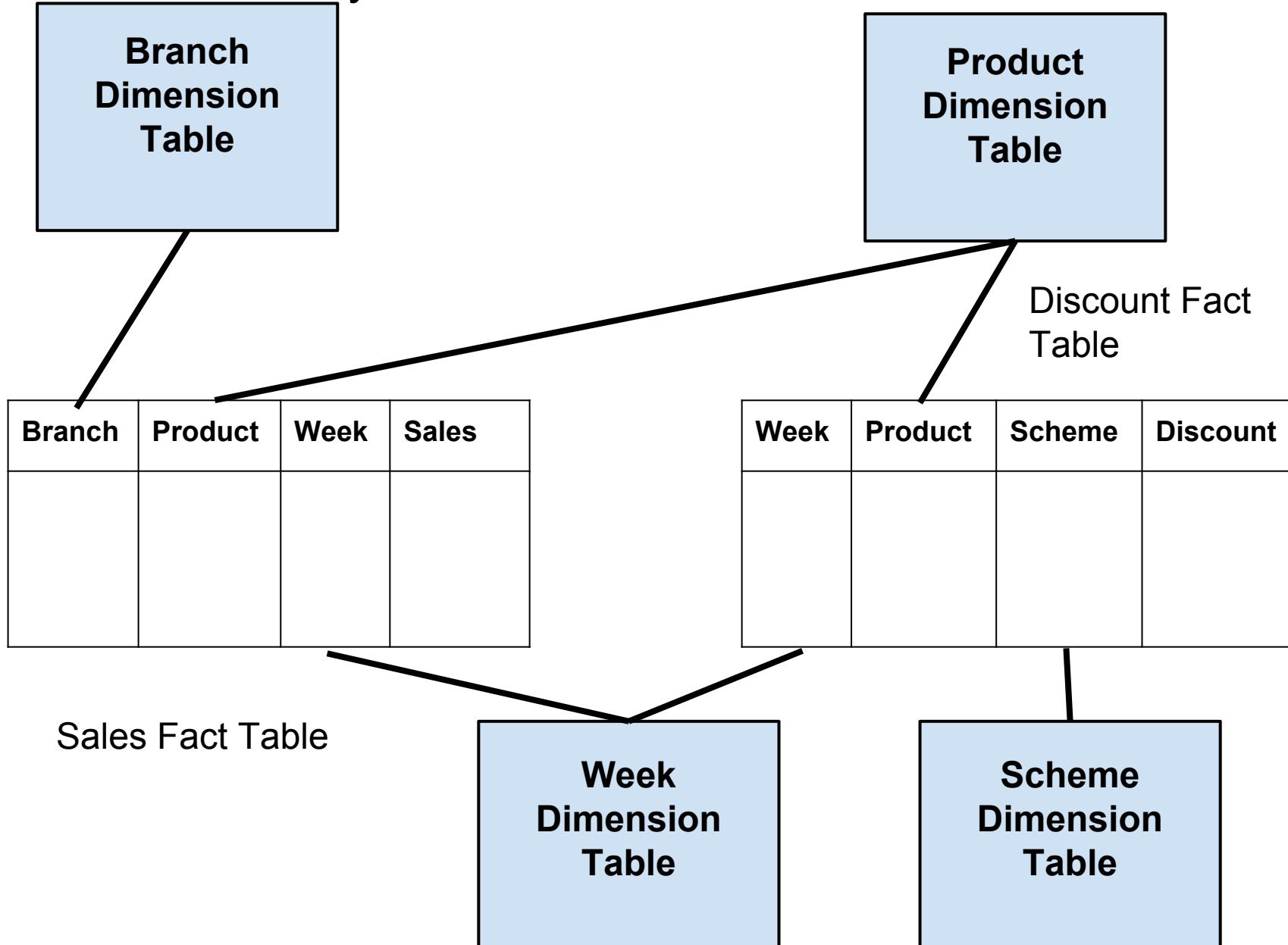
- Advantages

- Easy updation
    - Faster query responses

- Limitations

- Large amount of metadata
    - May result in too many tables
    - Harder to comprehend manually

# Physical Models : Constellation



# Data Warehousing

- Constellation
  - Most commonly used architecture
  - Used when multiple fact tables are needed
  - Usually has a “main” fact table and several “auxiliary” fact tables which are *summary tables* or *materialised views* over the main fact table
  - Helps in faster query answering for frequently asked queries
  - Costlier to update than snowflake

# Data Warehousing

- Multi dimensional Indexes
  - Bitmap Indexes
    - Used on fields that are sparse  
( eg : Gender, Grade etc.)
    - A bit vector enumerates all possible values and sets corresponding bit for each data element
    - Useful for efficiently answering composite queries over multiple bit - vectored fields

# Data Warehousing

- Multi dimensional Indexes
  - Encoding Bitmap Indexes

**Grade = { A, B, C, D, E, F }**   **Subject = { DBMS, ADBMS, IP }**

**A = 000001**

**DBMS = 001**

**B = 000010**

**ADBMS = 010**

**C = 000100**

**IP = 100**

**D = 001000**

**Null = 000**

**E = 010000**

**F = 100000**

**Null = 000000**

**Student who has scored A Grade in  
DBMS and ADBMS  
= { 000001 && 001 && 010 }**

# Data Mining

- It is the process of retrieving hidden knowledge from databases that can't be retrieved using normal queries
- For retrieving them, special data mining algorithms are used

# Data Mining

## Query - Example

Is there any relationship between the performance of a student in subject A and subject B ?

# Data Mining

- Need of Data Mining
  - Strategic Decision Making
  - Wealth Generation
  - Analysing Trends
  - Security

# Data Mining

- Type of Data
  - Tabular ( eg: Transaction data )
    - Relational
    - Multidimensional
  - Spatial (eg: Remote sensing data )
  - Temporal ( eg: Log information )
    - Streaming (eg: multimedia, network traffic )
    - Spatio - temporal ( eg: GIS)

# Data Mining

- Type of Data
  - Tree ( eg: XML data )
  - Graphs ( eg: www, Biomolecular data )
  - Sequence ( eg: DNA, activity logs)
  - Text, Multimedia ...

# Data Mining

- Type of Interestingness
  - Frequency
  - Rarity
  - Correlation
  - Length of occurrence ( for sequence and temporal data )
  - Consistency
  - Repeating / Periodicity
  - “Abnormal” behaviour

# Data Mining

- Associations
    - An association is a rule of the form if X then Y  
It is denoted as  $X \rightarrow Y$
- example:
- If India wins in cricket, then sales of sweets go up

# Data Mining

- Interesting Item Sets
  - For any rule if  $X \rightarrow Y \rightarrow Y \rightarrow X$  , then  $X$  and  $Y$  are called an “ interesting item set ”  
example:  
People buying school uniforms in june also buy school bags  
( People buying school bags in june also buy school uniforms )

# Data Mining

- Support
  - The **support** for an item set S is the ratio of the number of occurrences of S, given all occurrences of all item sets
- Confidence
  - The **confidence** of a rule  $X \rightarrow Y$ , is the ratio of the number of occurrences of Y given X, among all other occurrences given X

# Data Mining

- Support and Confidence

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

Support for {Bag, Uniform}

Confidence for

Bag -> Uniform

# Data Mining

- Support and Confidence

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

$$\begin{aligned} \text{Support for } \{\text{Bag, Uniform}\} \\ = 5 / 10 = 0.5 \end{aligned}$$

# Data Mining

- Support and Confidence

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

Confidence for  
Bag -> Uniform  
 $= 5 / 8 = 0.625$

# Data Mining

- Mining for Frequent Item Sets

- The Apriori Algorithm

Given minimum required support  $s$  as interestingness criterion.

1. Search for all individual elements( 1 element item set)that have a minimum support of  $s$
2. Repeat
  - a. From the results of the previous search for  $i$  element item sets, search for all  $i+1$  element item sets that have a minimum support of  $s$
  - b. This becomes the set of all frequent  $i+1$  element item sets that are interesting
3. Until item set size reaches maximum

# Data Mining

- Mining for Frequent Item Sets
  - The Apriori Algorithm ( Example)

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

Let minimum support = 0.3  
Interesting 1 element item set = { (bag), (uniform), (crayons), (books), (pencil) }

# Data Mining

- Mining for Frequent Item Sets
  - The Apriori Algorithm ( Example)

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

Let minimum support = 0.3

Interesting 2 element item set  
= { (Bag, Uniform), (Bag, Crayons), (Bag, Books), (Bag, Pencil),(Uniform,Crayons), (Uniform,Pencil),(Books, Pencil)}

# Data Mining

- Mining for Frequent Item Sets
  - The Apriori Algorithm ( Example)

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

Let minimum support = 0.3

Interesting 3 element item set  
= { (Bag, Uniform, Crayons) }

# Data Mining

- Mining for Association Rules

Association rules are of the form

$A \rightarrow B$

which are directional

Association rule mining requires two thresholds

minsup and minconf

# Data Mining

- Mining for Association Rules Using apriori

## General Procedure

1. Use apriori to generate frequent itemsets of different sizes
2. At each iteration divide each frequent itemset X into two parts LHS and RHS. This represents a rule of the form LHS  $\rightarrow$  RHS
3. The confidence of such a rule is  
$$\text{support}(X) / \text{support}(LHS)$$
4. Discard all rules whose confidence is less than minconf

# Data Mining

- Mining for Association Rules Using apriori  
Example

The frequent item set { Bag, Uniform, Crayons } has a minimum support of 0.3

This can be divided into the following rules

{ Bag } -> { Uniform, Crayons }

{ Bag, Uniform } -> { Crayons }

{ Bag, Crayons } -> { Uniform }

{ Uniform } -> { Bag, Crayons }

{ Uniform,Crayons } -> { Bag }

{ Crayons } -> { Bag,Uniform }

# Data Mining

- Mining for Association Rules Using apriori
  - Example

Confidence of these rules are :

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

{ Bag } -> { Uniform, Crayons } 0.375

{ Bag, Uniform } -> { Crayons } 0.6

{ Bag, Crayons } -> { Uniform } 0.75

{ Uniform } -> { Bag, Crayons } 0.429

{ Uniform,Crayons } -> { Bag } 0.75

{ Crayons } -> { Bag,Uniform } 0.6

If minconf is 0.7 then we have discovered the following rules ...

# Data Mining

- Mining for Association Rules Using apriori
  - Example

|         |         |         |
|---------|---------|---------|
| Bag     | Uniform | Crayons |
| Books   | Bag     | Uniform |
| Bag     | Uniform | Pencil  |
| Bag     | Pencil  | Books   |
| Uniform | Crayons | Bag     |
| Bag     | Pencil  | Books   |
| Crayons | Uniform | Bag     |
| Books   | Crayons | Bag     |
| Uniform | Crayons | Pencil  |
| Pencil  | Uniform | Books   |

If minconf is 0.7 then we have discovered the following rules ...

People who buy a school bag and a set of crayons are likely to buy school uniform

People who buy a school uniform and a set of crayons are likely to buy a school bag

# Data Mining

- Generalised Association Rules
  - Since customers can buy any number of items in one transaction, the transaction relation would be in the form of a list of individual purchases

| Bill No | Date       | Item    |
|---------|------------|---------|
| 15563   | 02-10-2010 | Uniform |
| 15563   | 02-10-2010 | Pencil  |
| 15564   | 02-10-2010 | Books   |
| 15564   | 02-10-2010 | Bag     |

# Data Mining

- Generalised Association Rules

- A GROUP BY over **Bill No** would show frequent buying patterns across different customers
- A GROUP BY over **Date** would show frequent buying patterns across different days

| Bill No | Date       | Item    |
|---------|------------|---------|
| 15563   | 02-10-2010 | Uniform |
| 15563   | 02-10-2010 | Pencil  |
| 15564   | 02-10-2010 | Books   |
| 15564   | 02-10-2010 | Bag     |

# Data Mining

- Classification and Clustering
  - Classification maps each data element to one of a set of pre-determined classes based on the difference among data elements belonging to different classes
  - Clustering groups data elements into different groups based on the similarity between elements within a single group

# Data Mining

- Decision Tree Identification

| Outlook  | Temp | Play? |
|----------|------|-------|
| Sunny    | 30   | Yes   |
| Overcast | 15   | No    |
| Sunny    | 16   | Yes   |
| Cloudy   | 27   | Yes   |
| Overcast | 25   | Yes   |
| Overcast | 17   | No    |
| Cloudy   | 17   | No    |
| Cloudy   | 35   | Yes   |

Classification Problem

Weather -> Play (Yes, No)

# Data Mining

- Hunt's Method For Decision Tree Identification

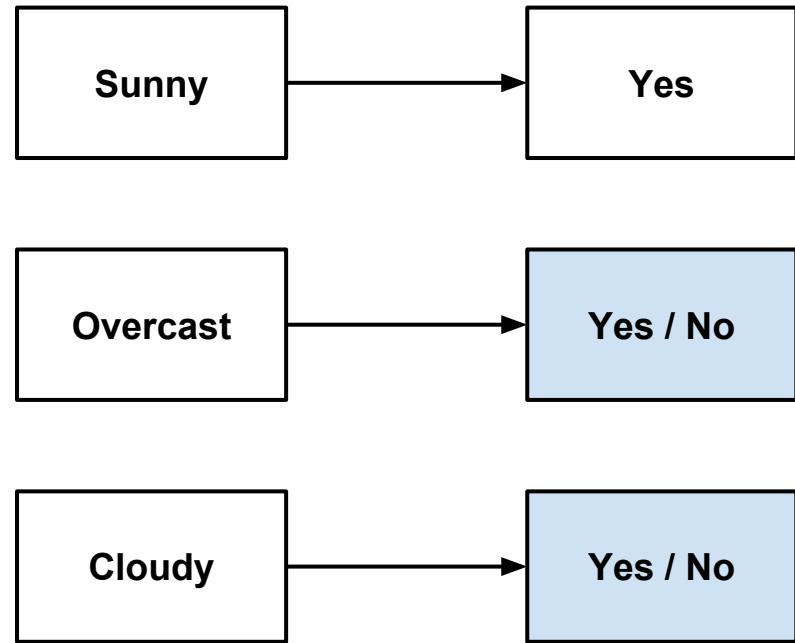
Given N element types and m decision classes

1. For  $i \leftarrow 1$  to  $N$  do
  1. Add element  $i$  to the  $i-1$  element item sets from the previous iteration
  2. Identify the set of decision classes for each item set
  3. If an item set has only one decision class, then that item set is done, remove that item set from subsequent iterations
2. Done

# Data Mining

- Decision Tree Identification

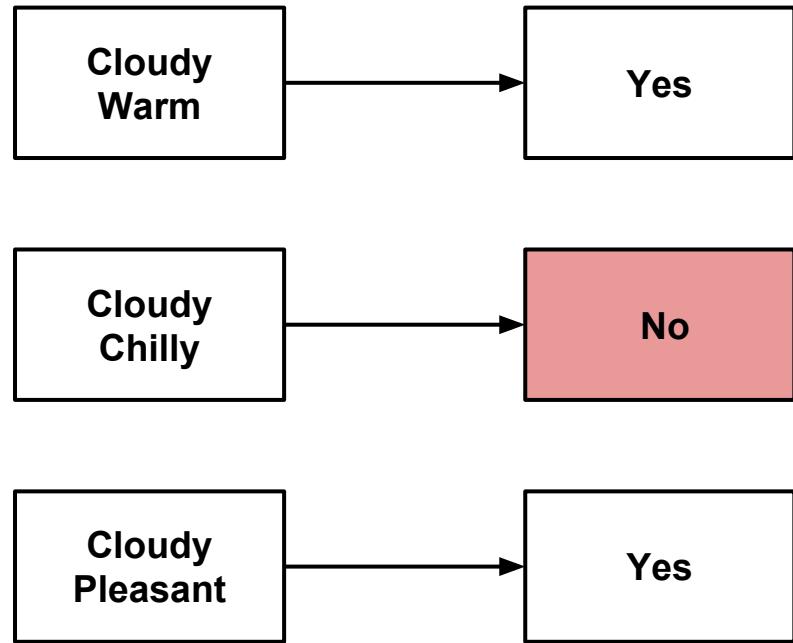
| Outlook  | Temp     | Play? |
|----------|----------|-------|
| Sunny    | Warm     | Yes   |
| Overcast | Chilly   | No    |
| Sunny    | Chilly   | Yes   |
| Cloudy   | Pleasant | Yes   |
| Overcast | Pleasant | Yes   |
| Overcast | Chilly   | No    |
| Cloudy   | Chilly   | No    |
| Cloudy   | Warm     | Yes   |



# Data Mining

- Decision Tree Identification

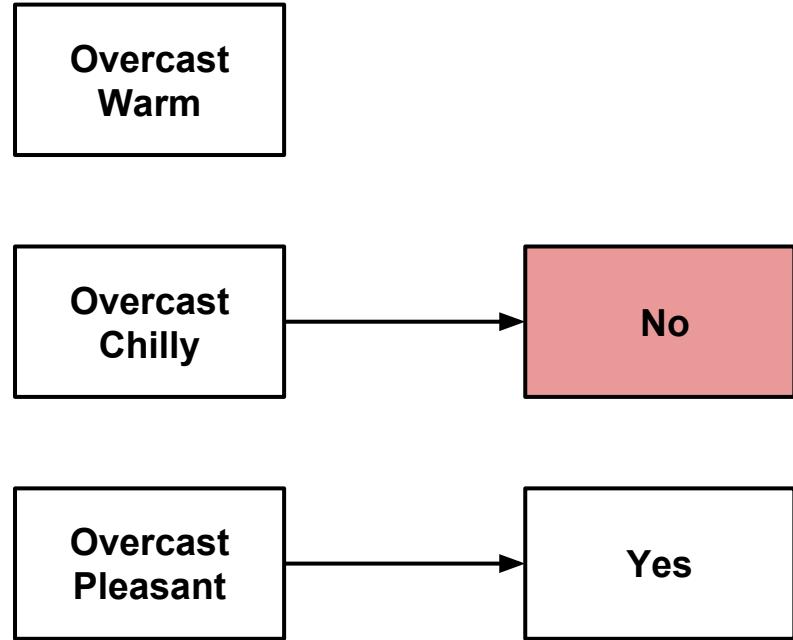
| Outlook  | Temp     | Play? |
|----------|----------|-------|
| Sunny    | Warm     | Yes   |
| Overcast | Chilly   | No    |
| Sunny    | Chilly   | Yes   |
| Cloudy   | Pleasant | Yes   |
| Overcast | Pleasant | Yes   |
| Overcast | Chilly   | No    |
| Cloudy   | Chilly   | No    |
| Cloudy   | Warm     | Yes   |



# Data Mining

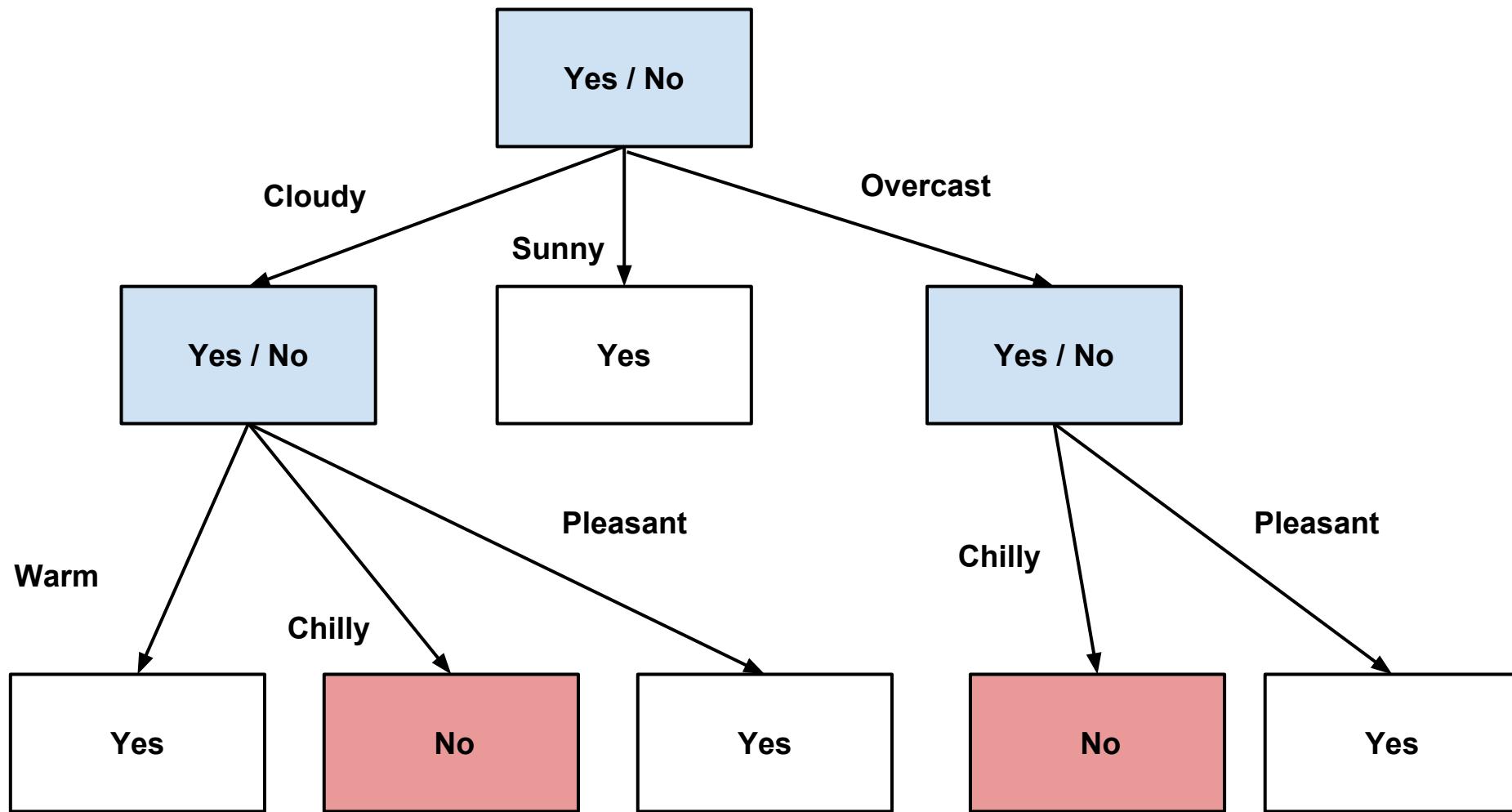
- Decision Tree Identification

| Outlook  | Temp     | Play? |
|----------|----------|-------|
| Sunny    | Warm     | Yes   |
| Overcast | Chilly   | No    |
| Sunny    | Chilly   | Yes   |
| Cloudy   | Pleasant | Yes   |
| Overcast | Pleasant | Yes   |
| Overcast | Chilly   | No    |
| Cloudy   | Chilly   | No    |
| Cloudy   | Warm     | Yes   |



# Data Mining

- Decision Tree Identification Example



# Data Mining

- Clustering Techniques

- Nearest Neighbour Clustering Algorithm

Given n elements  $x_1, x_2, \dots, x_n$  and threshold t

1.  $j \leftarrow 1, k \leftarrow 1, \text{Clusters} = \{ \}$
2. Repeat
  - a. Find the nearest neighbour of  $x_j$
  - b. Let the nearest neighbour be in cluster m
  - c. If distance to nearest neighbour  $> t$ , then  
create a new cluster and  $k \leftarrow k + 1$ ; else  
assign  $x_j$  to cluster m
  - d.  $j \leftarrow j + 1$
3. until  $j > n$

# Data Mining

- Clustering Techniques

- Iterative Partitional Clustering

Given  $n$  elements  $x_1, x_2, \dots, x_n$  and  $k$  clusters each with a centre

1. Assign each element to its closest cluster centre
2. After all assignments have been made, compute the cluster centroids for each of the cluster
3. Repeat the above two steps with the new centroids until the algorithm converges

# Data Mining

- Mining Sequence Data

- A sequence is a list of itemsets of finite length
- Example

{ pen, pencil, ink } { pencil, ink } { ink, eraser } { ruler, pencil }

The purchases of a single customer over time

- The order of items within an itemset does not matter; but the order of itemsets matter
- In a sequence each item has an index associated with it
- A k - sequence is a sequence of length k

# Data Mining

- Mining Sequence Data

- A subsequence is a sequence with some itemsets deleted
- A sequence  $S' = \{ a_1, a_2, \dots, a_m \}$  is said to be contained within another sequence  $S$ , if  $S$  contains a subsequence  $\{ b_1, b_2, \dots, b_m \}$  such that

$$a_1 \subseteq b_1, a_2 \subseteq b_2, \dots, a_n \subseteq b_n$$

- Hence  $\{ \text{pen} \} \{ \text{pencil} \} \{ \text{ruler, pencil} \}$

is contained within

$\{ \text{pen, pencil, ink} \} \{ \text{pencil, ink} \} \{ \text{ink, eraser} \}$   
 $\{ \text{ruler, pencil} \}$

# Data Mining

- Mining Sequence Data
  - Apriori algorithm for sequences
    1.  $L_1 \leftarrow$  Set of all interesting 1-sequences
    2.  $k \leftarrow 1$
    3. while  $L_k$  is not empty do
      - a. Generate all candidate  $k+1$  - sequences
      - b.  $L_{k+1} \leftarrow$  Set of all interesting  $k+1$  - sequences
    4. done

# Data Mining

- Mining Sequence Data
  - Generating Candidate Sequences

Given  $L_1, L_2, \dots, L_k$  candidate sequences of  $L_{k+1}$  are generated as follows :

*For each sequence  $s$  in  $L_k$ , concatenate  $s$  with all new 1-sequences found while generating  $L_{k-1}$*

# Data Mining

- Mining Sequence Data

Example

$\text{minsup} = 0.5$

a b c d e  
b d a e  
a e b d  
b e  
e a b d a  
a a a a  
b a a a  
e b d b  
a b b a b  
a b d e

Interesting 1-sequences

a  
b  
d  
e

Candidate 2-sequences

aa ab ad ae  
ba bb bd be  
da db dd de  
ea eb ed ee

# Data Mining

- Mining Sequence Data

Example

$\text{minsup} = 0.5$

a b c d e

b d a e

a e b d

b e

e a b d a

a a a a

b a a a

e b d b

a b b a b

a b d e

Interesting 2-sequences

ab bd

Candidate 3-sequences

aba abb abd abe

aab bab dab eab

bda bdb bdd bde

bbd dbd ebd

Interesting 3-sequences

{ }

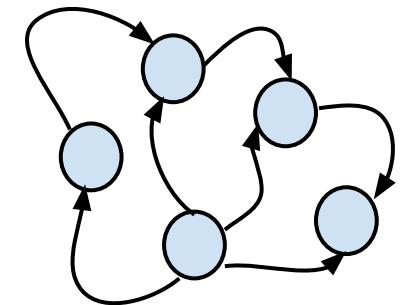
# Data Mining

- Mining Sequence Data

aabb  
ababcac  
abbac



**Input Set Of Sequences**

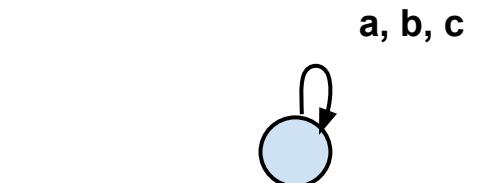


**Output State Diagram**

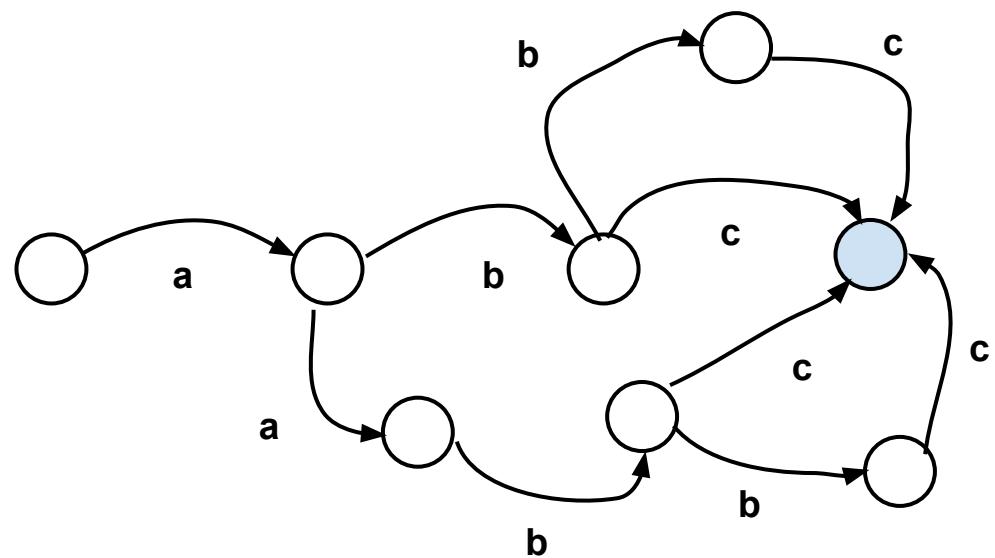
# Data Mining

- Mining Sequence Data

abc  
aabc  
aabbc  
abbc



Most General State Machine

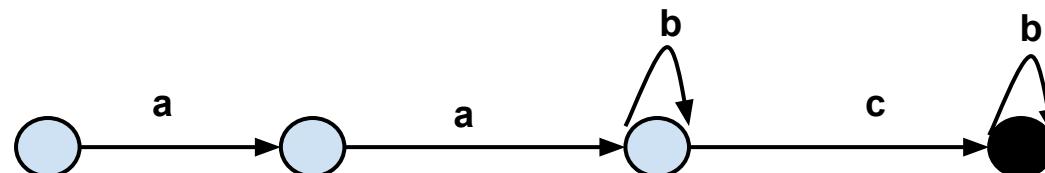
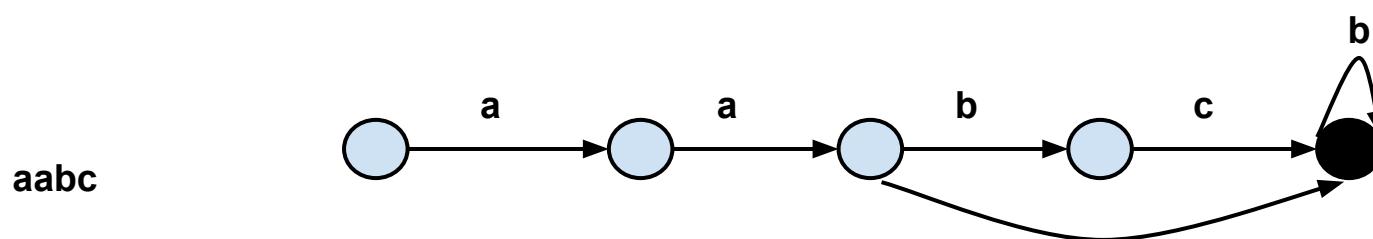
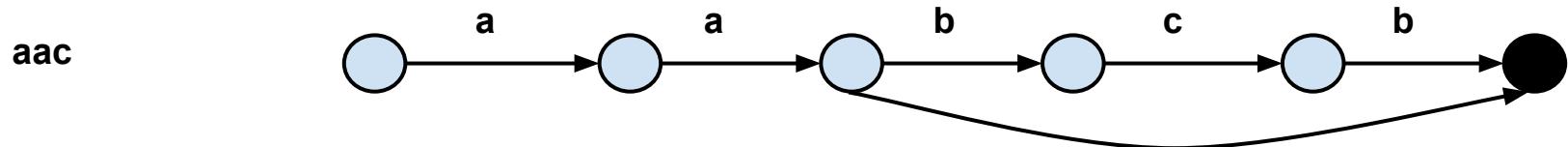
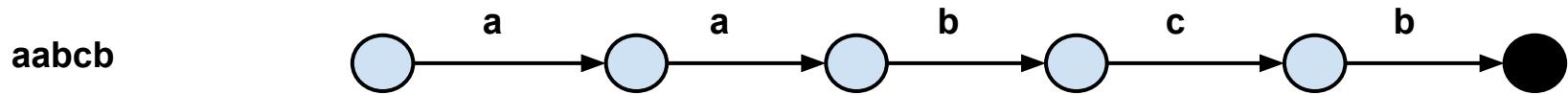


Most Specific State Machine

# Data Mining

- Mining Sequence Data

- Shortest Run Generalisation



# Data Mining

- Mining Streaming Data
  - Characteristics of Streaming Data
    - Large Data Sequence
    - No Storage
    - Often an infinite sequence
    - Examples : Stock Market Quotes, Streaming Audio / Video, Network Traffic

# Data Mining

- Mining Streaming Data
  - Running Mean

Let  $n$  = number of items read so far

$\text{avg}$  = average of items read so far

On reading the next number  $\text{num}$ ,

$$\text{avg} = (\text{n} * \text{avg} + \text{num}) / (\text{n} + 1)$$

$$\text{n} = \text{n} + 1$$

# Data Mining

- Marketing
  - used for analysing consumer behaviour based on buying patterns
- Finance
  - used for analysing the performance of finance investments like stocks, bonds, mutual funds etc.

# Data Mining

- Healthcare
  - analyse effectiveness of certain treatments,
  - analyse side effects of certain drugs
- Manufacturing
  - optimisation of resources like machines, manpower and materials
  - optimal design of products ( manufacture automobiles based on customer requirements)

# Databases on The World Wide Web

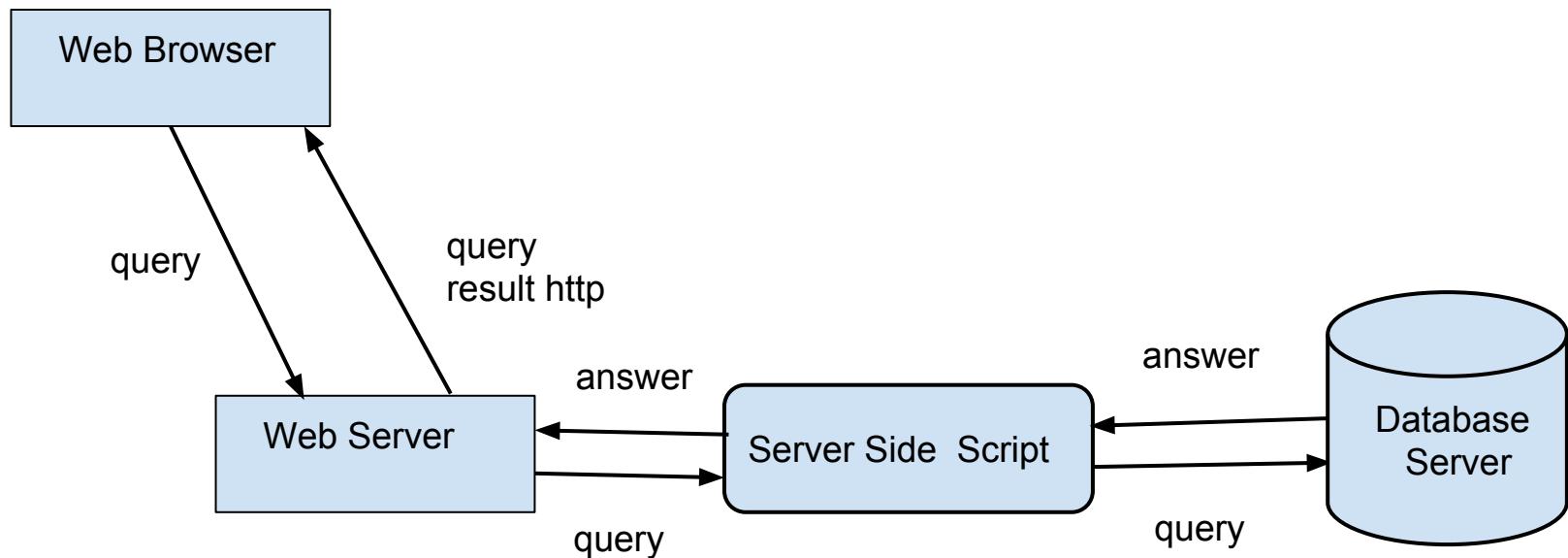
- The world wide web(W W W) is a distributed information system where information is stored on computers designated as web servers
- The information is stored in files encoded using Hyper Text Markup Language (HTML)
- This information is accessed using a web browser, which is a program that communicates with the server

# Databases on The World Wide Web

- For this, a web browser uses a URL(Uniform Resource Locator) which provides the complete path name of a file
- A URL always begins with a hypertext transport protocol (http), which is the protocol used by the web browsers

# Databases on The World Wide Web

Database Access On The Web



# Databases on The World Wide Web

- Some of the server side scripting languages are
  - ASP.NET
  - JSP
  - JavaScript
  - Perl CGI
  - PHP
  - R
  - Python

# Multimedia Databases

- They store multimedia data such as text, image, animation, audio and video
- They are stored in databases if the number of multimedia objects is large
- If the number is relatively small, they can be stored in filesystems

# Multimedia Databases

- One approach to building a database for such multimedia objects is to use databases for storing the descriptive attributes and for keeping track of the files in which the multimedia objects are stored.
- However, storing multimedia outside the database makes it harder to provide database functionality, such as indexing on the basis of actual multimedia data content. It is therefore desirable to store the data themselves in the database.

# Multimedia Databases

- Several issues are needed to be addressed if they are to be stored in a database
- The database must support large objects since multimedia data such as video can occupy upto several GBs of storage

# Multimedia Databases

- Similarity-based retrieval is needed in multimedia databases
- For example, in a database that stores fingerprint images we need to identify an image that matches a particular image

# Multimedia Databases

- The data delivery in the case of audio and video must proceed at a guaranteed steady rate
- Otherwise there will be gaps in the supplied data which is not desirable

# Multimedia Databases

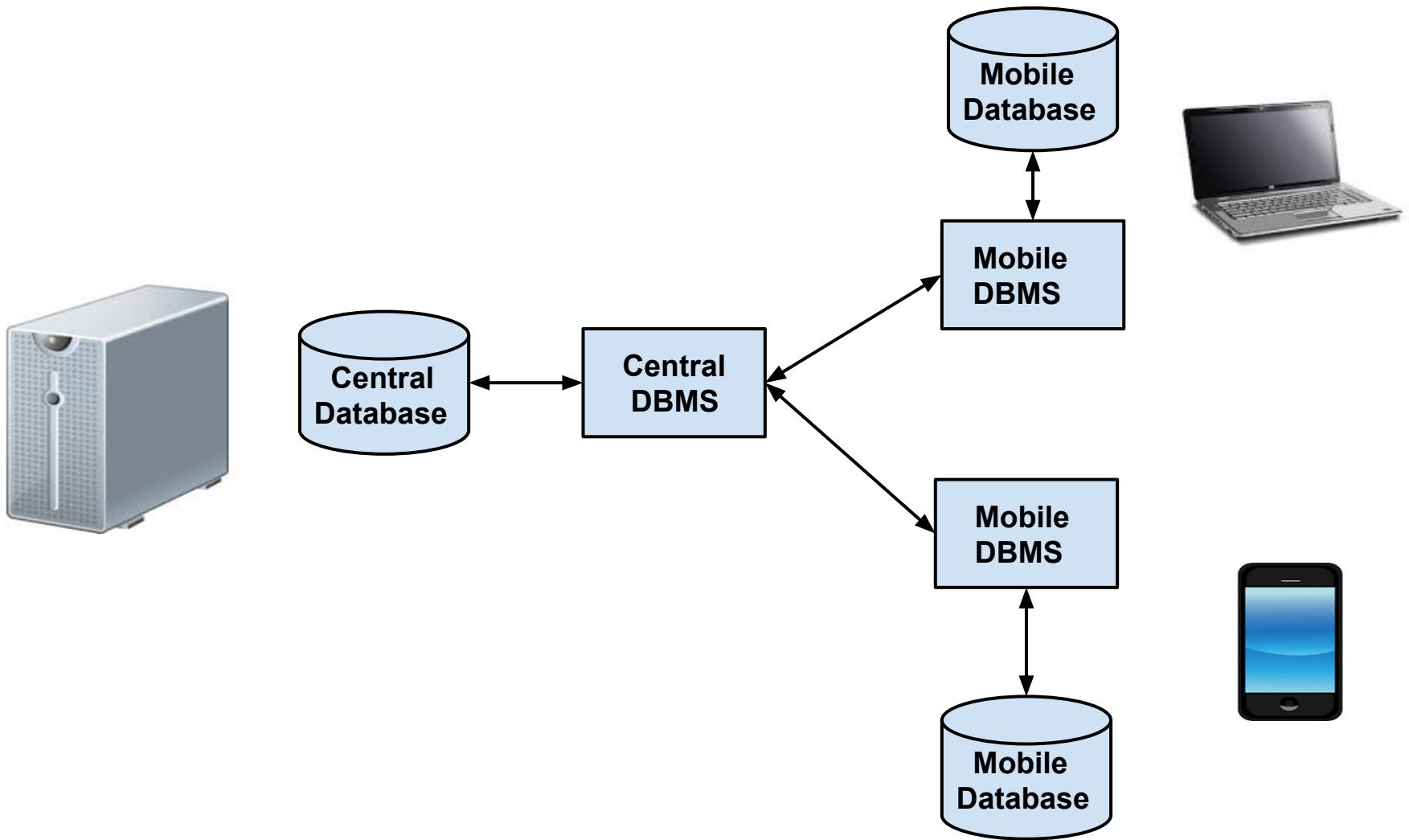
- Applications

- Digital Libraries
- Video On Demand
- Music Database
- Geographic Information Systems
- Tele Medicine

# Mobile Databases

- A mobile database is a portable database that is capable of communicating with the corporate server from remote sites enabling it to access shared data.
- The communication link can be
  - Wireless
  - Internet

# Mobile Databases



# Mobile Databases

- Examples
  - Sybase SQL Anywhere
  - Oracle Lite
  - Microsoft SQL Server Compact
  - SQLite
  - IBM Mobile Database

# Mobile Databases

- Applications
  - Business

Salespersons can update sales and customer data on the move
  - Media

Reporters can update news from anywhere
  - Health Sector

Physicians can store and retrieve information while making their rounds

Used by doctors to retrieve vital patient history and treatment information in battlefields and remote accident locations

# Mobile Databases

- Advantages
  - Location Flexibility
  - Saves Time
  - Enhanced Productivity

# Mobile Databases

- Limitations
  - Limited wireless bandwidth / communication speed
  - Limited energy source ( battery speed )
  - Less secured
  - Vulnerable to physical activities
  - Hard to make theft proof

# Mobile Databases

- Data Classification
  - Location Dependent Data
  - Location Independent Data

# Mobile Databases

- Data Classification
  - Location Dependent Data

Here the value of the location determines the correct value of data

Examples : City Tax, City Area

# Mobile Databases

- Data Classification
  - Location Independent Data

Here the value of the location does not affect the value of data

Examples : Customer Number, Account Number

# Mobile Databases

- Query Types
  - Location Dependent Query
  - Location Independent Query

# Mobile Databases

- Query Types
  - Location Dependent Query

The answer to the query is dependent on the geographical location of the query

example: How far is the Kuttippuram Railway Station from here?

Here we have to do continuous monitoring of the latitude and longitude of the origin of the query using GPS

# Mobile Databases

- Query Types
  - Location Independent Query

The geographical location does not affect the answer to the query

example: List the order history of a particular customer during the last one year

# **Geographic Information Systems (GIS)**

- They are spatial databases used to store geographic information, such as maps
- They handle
  - Raster Data
  - Vector Data

# Geographic Information Systems (GIS)

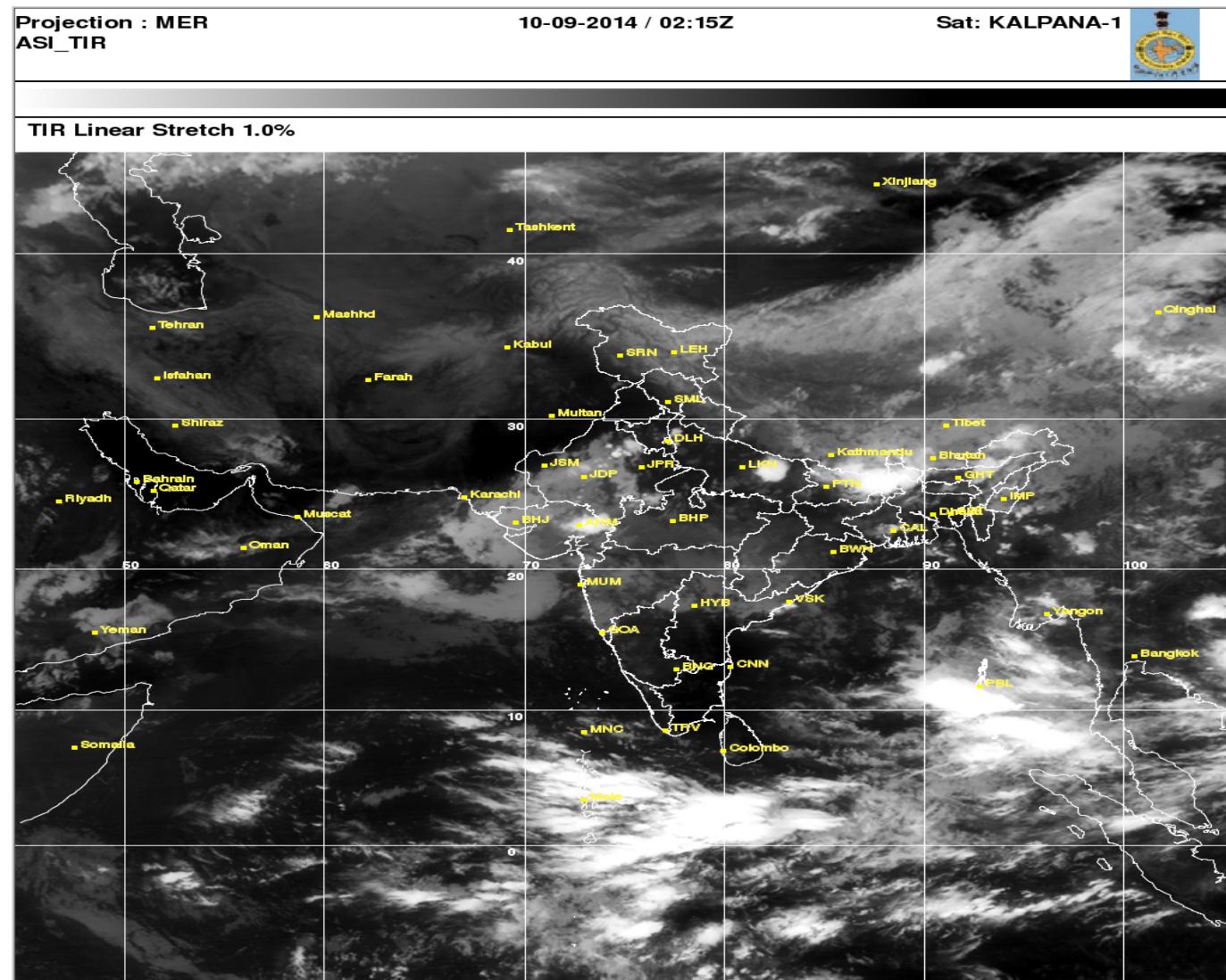
- Raster Data

They contain bitmap or pixel map in two or more dimensions

- example

satellite image of cloud cover, where each pixel represents the cloud visibility of a particular area

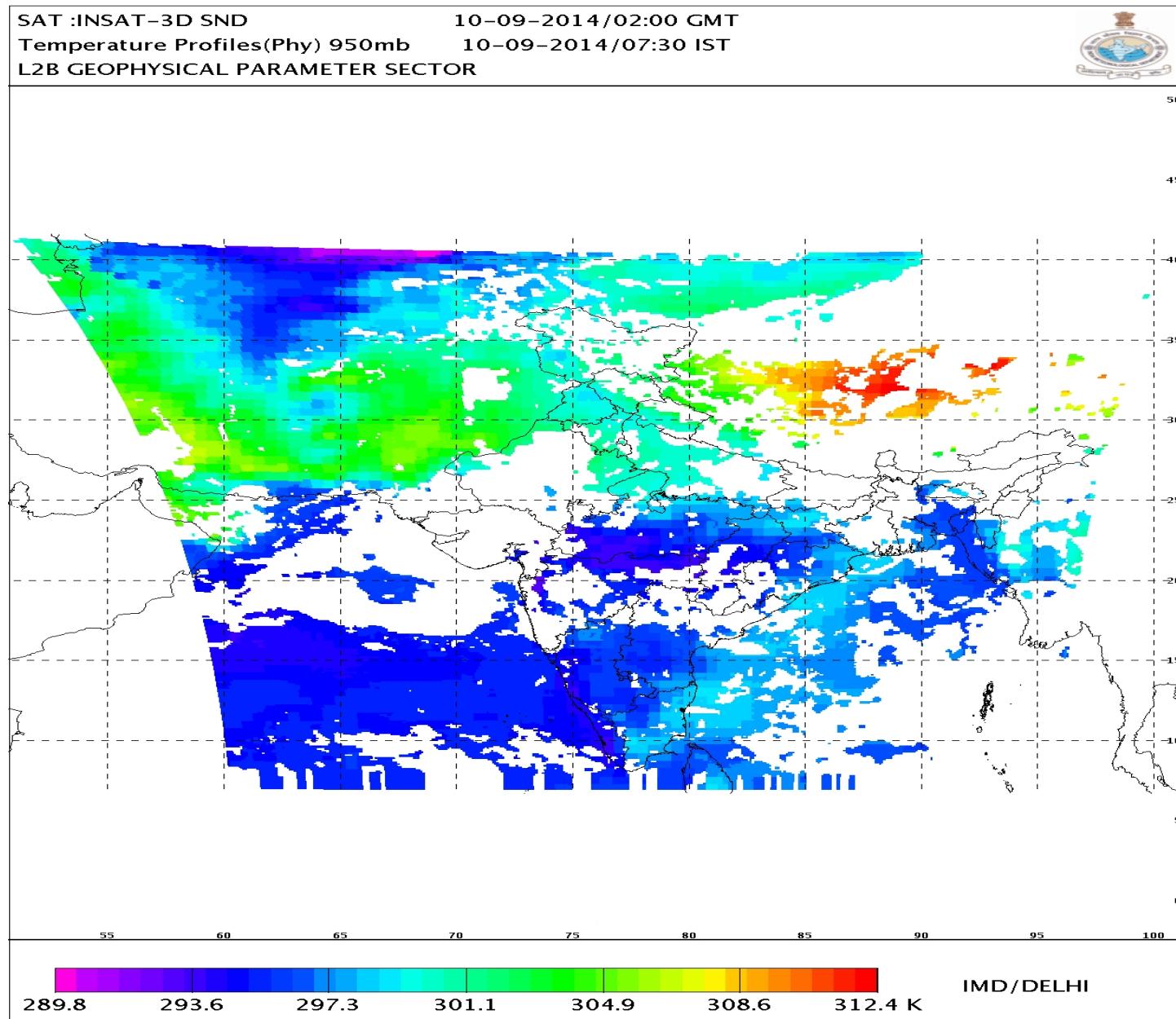
# Geographic Information Systems (GIS)



# Geographic Information Systems (GIS)

- Raster Data
  - example ( three dimensional)  
satellite image showing temperature at different altitudes in different locations
  - example ( Time Dimension )  
satellite image showing surface temperature measurements at different points of time

# Geographic Information Systems(GIS)



# Geographic Information Systems (GIS)

- Vector Data

They are constructed using basic geometric objects such as

- points, line segments, triangles
- other polygons
- cylinders, spheres, cuboids
- other polyhedrons

# Geographic Information Systems (GIS)

- Vector Data
  - Map data is usually represented in this format
  - Rivers, Roads - union of multiple line segments
  - Rivers - complex curves, complex polygons
  - States, Countries, Lakes - complex polygons

# **Geographic Information Systems (GIS)**

- Applications
  - vehicle navigation systems
  - network distribution information of telephone, electric power and water supply systems
  - land usage information for ecologists and planners

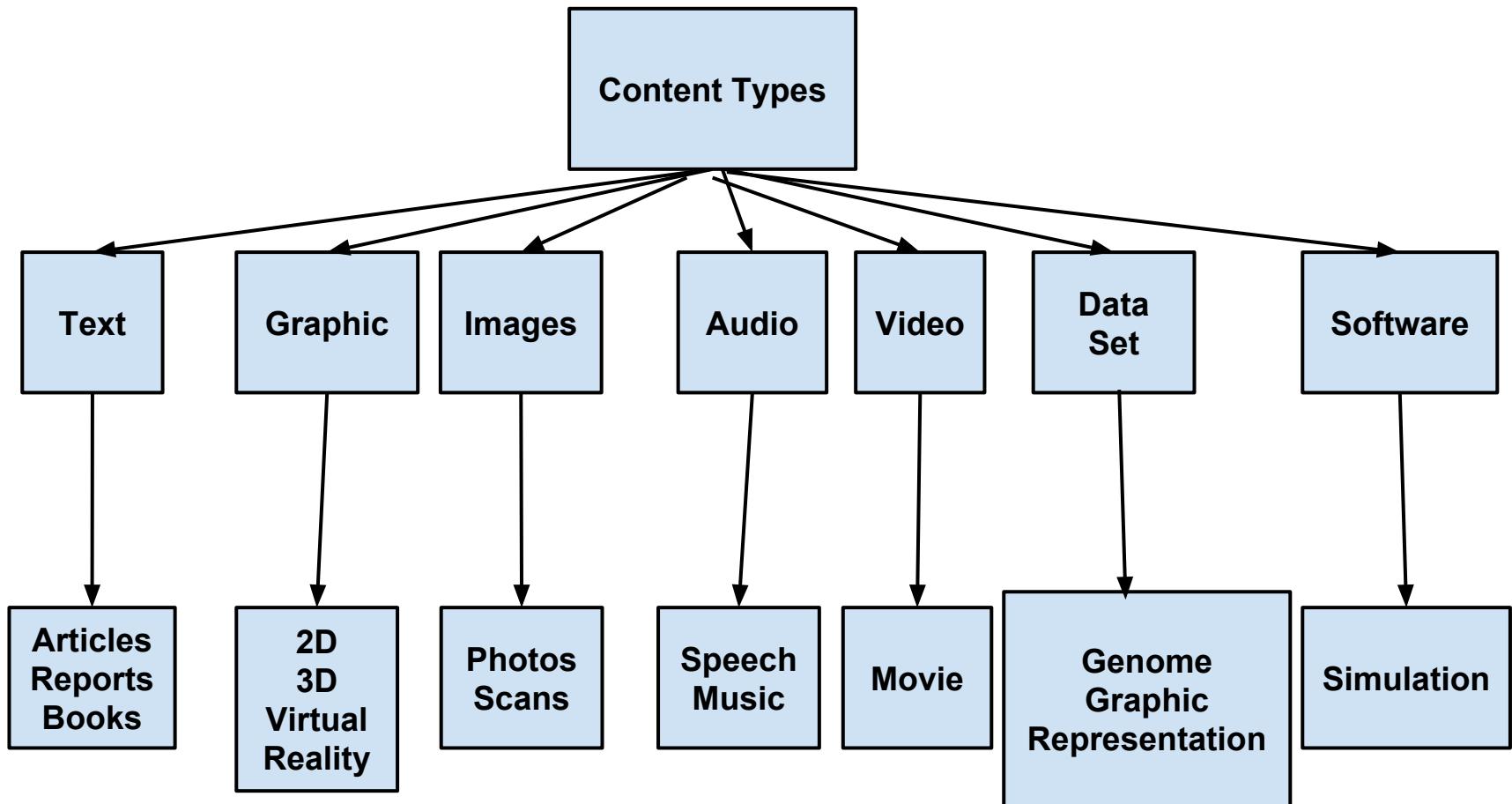
# **Geographic Information Systems (GIS)**

- Vehicle Navigation Systems
  - They store information about roads and services for the use of drivers
  - Information is stored in the form of online maps
  - They can be used for automatic trip planning
  - Information includes
    - layout of roads
    - speed limit on roads
    - road conditions
    - connections between roads
    - one way restrictions
    - hotels, petrol pumps

# Digital Libraries

- A digital library is a library in which collections are stored in digital formats and accessible via computers
- It can contain text, image, audio and video

# Digital Libraries



# Digital Libraries

- Characteristics
  - Remote access is quick and easy
  - Round the clock availability
  - Materials are copied from a master version
  - Keeping extra copies on hand is easy

# Digital Libraries

- Characteristics
  - Multiple Access
  - Information Retrieval is Easy ( Search )
  - Preservation and Conservation
  - Storage Space is not limited
  - Added Value ( Quality of objects such as images can be improved )

# Digital Libraries

- Sample List
  - [www.gutenberg.org](http://www.gutenberg.org) (Project Gutenberg)
  - [books.google.co.in](http://books.google.co.in) (Google Books)
  - [www.wdl.org](http://www.wdl.org) (World Digital Library)
  - [www.ipl.org](http://www.ipl.org) (Internet Public Library)
  - [www.dli.ernet.in](http://www.dli.ernet.in) ( Digital Library of India)
  - [www.nationallibrary.gov.in](http://www.nationallibrary.gov.in) (National Library)
  - [www.tkdl.res.in](http://www.tkdl.res.in) (Traditional Knowledge Digital Library)

# **Module V**

- Oracle
- Microsoft Access

# Oracle

- Developed by Oracle Corporation
- An early player in the RDBMS field
- One of the widely used RDBMSs

# **Oracle**

- It runs on a wide range of platforms, varying from mainframes to workstations
- It was released in 1978
- It is written in assembly language, C and C++
- Latest version - Oracle 12c

# **Access**

- It is a relational DBMS developed for PCs by Microsoft Corporation
- It was released in November 1992
- Latest version is Access 2013

# RDBMS

- To qualify as a genuine RDBMS a system must have at least the following properties
  - It must store data as relations such that each column is independently identified by its column name, and the ordering of rows is immaterial

# RDBMS

- To qualify as a genuine RDBMS a system must have at least the following properties
  - The operations in the RDBMS should be able to generate new relations from old relations

# RDBMS

- To qualify as a genuine RDBMS a system must have at least the following properties
  - The system must support at least one variant of join operation

# RDBMS

- In fact there are 12 rules for determining whether a DBMS is relational
- These rules were specified by Codd, who developed the relational model

# The Basic Structure - Oracle System

- An Oracle Server consists of
  - Oracle Database
  - Oracle Instance

# The Basic Structure - Oracle System

- The Oracle Database has two primary structures
  - Physical Structure  
referring to the actual stored data
  - Logical Structure  
abstract representation of the stored data

# The Basic Structure - Oracle System

- The Database contains the following types of files
  - Data Files  
contain the actual data
  - Log Files ( Redo Log Files)
    - They record all changes made to data
    - They are used in the process of recovery

# The Basic Structure - Oracle System

- The Database contains the following types of files
    - Control Files
      - contain the control information such as
        - database name
        - file names and locations
        - database creation timestamp
- They are also needed for recovery

# The Basic Structure - Oracle System

- The Database contains the following types of files
  - Trace Files and an Alert Log
    - background processes have a trace file associated with them
    - The alert log maintains major database events

# The Basic Structure - Oracle System

- The Oracle Instance
  - It contains all the processes created for a specific instance of the database operation
  - It includes
    - User Processes
    - System ( Oracle ) Processes

# The Basic Structure - Oracle System

- The Oracle Instance
  - User Processes

They correspond to the execution of some application or some tool

# The Basic Structure - Oracle System

- The Oracle Instance
  - System Processes
    - Server Processes
    - Background Processes

# The Basic Structure - Oracle System

- The Oracle Instance

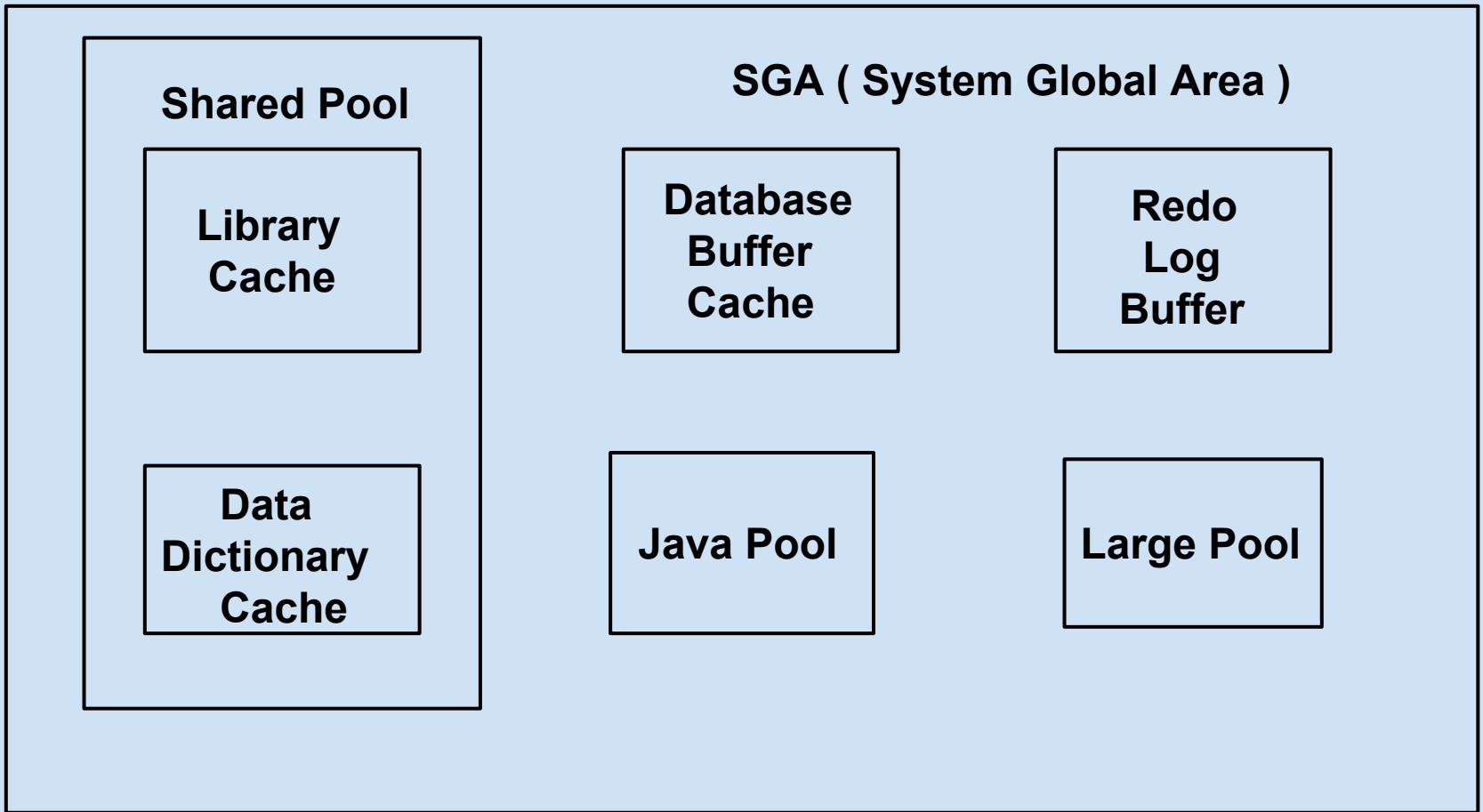
- Server Processes

- They handle requests from connected user processes

- Background Processes

- They are created for each instance of Oracle to perform I/O asynchronously and provide parallelism for better performance and reliability

# Oracle Instance



PMON

SMON

DBWR

LGWR

CKPT

SP

UP

# The Basic Structure - Oracle System

- The Oracle Instance
  - System Global Area ( SGA)

Used for storing database information shared by users

It contains the following memory structures

    - Database Buffer Cache
      - keeps the most recently accessed data blocks from the database
    - Redo Log Buffer
      - used for recovery purposes

# The Basic Structure - Oracle System

- The Oracle Instance
  - System Global Area ( SGA)

Used for storing database information shared by users

It contains the following memory structures

    - Shared Pool

contains two key performance related memory structures

      - Library cache

stores most recently executed SQL statements

# The Basic Structure - Oracle System

- The Oracle Instance
  - System Global Area ( SGA)

Used for storing database information shared by users

It contains the following memory structures

    - Shared Pool

contains two key performance related memory structures

      - Data Dictionary Cache

stores most recently used data definitions

# The Basic Structure - Oracle System

- The Oracle Instance
  - System Global Area ( SGA)

Used for storing database information shared by users

It contains the following memory structures

    - Java Pool
      - required if using Java (optional)
    - Large Pool
      - relieves the burden placed on the shared pool (optional)

# The Basic Structure - Oracle System

- The Oracle Instance
  - Program Global Area ( PGA) (not shown in figure )  
Used for storing information about a server process

# The Basic Structure - Oracle System

- The Oracle Instance
  - Background Processes
    - PMON ( Process Monitor )
      - performs process recovery when a user process fails
      - It is responsible for releasing locks and rolling back the transaction

# The Basic Structure - Oracle System

- The Oracle Instance
  - Background Processes
    - SMON ( System Monitor )
      - performs instance recovery
      - makes changes in redo logs
      - rolls back uncommitted transactions
      - responsible for managing the storage area by making the space contiguous

# The Basic Structure - Oracle System

- The Oracle Instance
  - Background Processes
    - DBWR ( DataBase WRiter )
      - responsible for writing the modified blocks from the database buffer cache to the data files on disk

# The Basic Structure - Oracle System

- The Oracle Instance
  - Background Processes
    - LGWR ( LoG WRiter )
      - writes from the log buffer area to the online disk log file

# The Basic Structure - Oracle System

- The Oracle Instance
  - Background Processes
    - CKPT ( CheckPoint )
      - works along with DBWR to execute a checkpointing operation

# Oracle Startup and Shutdown

- Steps in starting an Oracle
  - **Starting an instance of the database**  
The SGA is allocated and background processes are created
  - **Mounting a database**  
associates a previously started oracle instance with the database
  - **Opening a database**  
making the mounted database available for normal database operations

# Oracle Startup and Shutdown

- Steps in shutting down an Oracle Instance
  - Close the database
  - Dismount the database
  - Shut down the oracle instance

# **Database Structure and its manipulation in Oracle**

- Oracle was designed originally as a RDBMS
- From version 8 onwards it is an object relational DBMS

# Database Structure and its manipulation in Oracle

- Schema Objects
  - They refer to the collection of data definition objects such as tables, views etc.

# Database Structure and its manipulation in Oracle

- Schema Objects
  - Tables
  - Views
  - Synonyms
  - Program Units
  - Sequence
  - Indexes
  - Cluster
  - Database Links

# Database Structure and its manipulation in Oracle

- Schema Objects
  - **Synonyms**

Another name that can be used for referring an object

# Database Structure and its manipulation in Oracle

- Schema Objects
  - Program Units
    - function
    - stored procedure
      - a procedure that is part of data defn which can implement some integrity rule
    - package
      - a collection of related procedures

# Database Structure and its manipulation in Oracle

- Schema Objects
  - sequence
    - automatically generated internal number which can provide value to attributes

## example

values for empid of the employee table may be internally generated as a sequence

# Database Structure and its manipulation in Oracle

- Schema Objects

- **indexes**

They are used for improving the speed of data retrieval

- **cluster**

A group of records from multiple tables stored together to improve the performance of the database

# **Database Structure and its manipulation in Oracle**

- Schema Objects
  - database links
    - They establish paths from one database to another
    - They are used in distributed databases

# **Database Structure and its manipulation in Oracle**

- Oracle Data Dictionary

It is a read only set of tables that keeps the metadata for a database

# **Database Structure and its manipulation in Oracle**

- Oracle Data Dictionary

It contains

- Names of users
- Security information ( about which user has access to what data )
- Integrity Constraints
- space allocation and utilisation of database objects
- audit trail information
- statistics on attributes, tables and predicates

# Database Structure and its manipulation in Oracle

- Oracle Data Dictionary

It is possible to query the data dictionary using SQL

## Query

```
SELECT owner, object_name, object_type
FROM all_objects;
```

This query returns information about all objects to which the user has access

# **Database Structure and its manipulation in Oracle**

- SQL in Oracle

Oracle handles the following SQL statements

- DDL statements
- DML statements
- Transaction Control Statements
- Session Control Statements

For example, statements for enabling or disabling roles of users ( Create Role )

# Database Structure and its manipulation in Oracle

- SQL in Oracle

Oracle handles the following SQL statements

- System Control Statements

It allows the administrator to change settings such as the minimum number of shared servers, or to kill a session

example

ALTER SYSTEM

# Database Structure and its manipulation in Oracle

- SQL in Oracle

Oracle handles the following SQL statements

- Embedded SQL statements
  - SQL statements can be embedded in PL/SQL of Oracle or languages like C
  - The PL/SQL is Oracle's procedural language that adds procedural functionality to SQL

# **Database Structure and its manipulation in Oracle**

- Methods in Oracle

A method ( operation ) is a procedure or function that is part of a user-defined abstract data type ( like class )

# Database Structure and its manipulation in Oracle

- Triggers
  - A trigger is a stored procedure that is implicitly executed when the table with which it is associated has an insert, delete, or update performed on it
- Triggers are used to enforce additional constraints

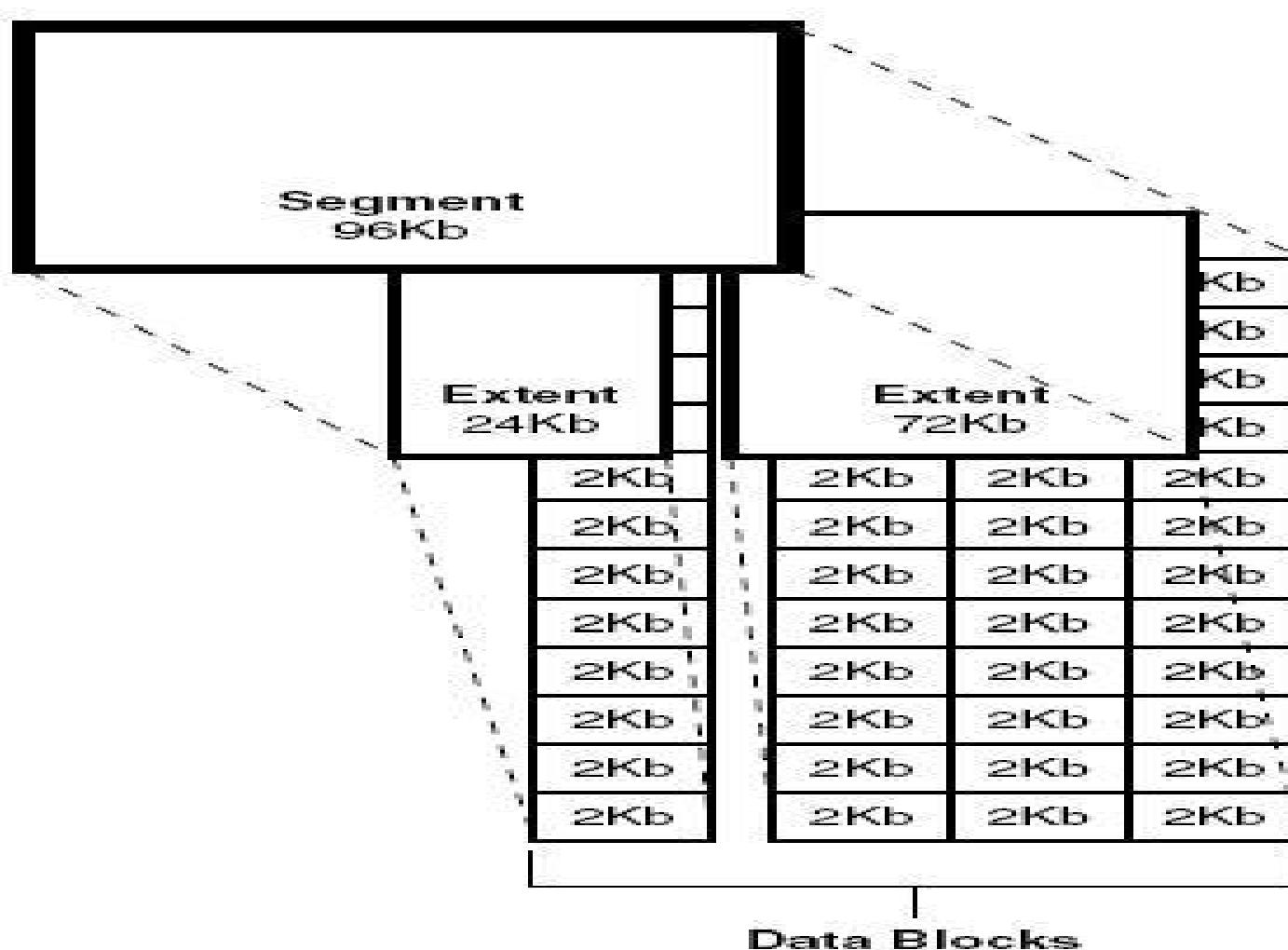
# Storage Organisation in Oracle

- A database is divided into logical storage units called **tablespaces**
- There is **system tablespace** and **users tablespace**
- A **system tablespace** is used for storing system related data like data dictionary objects
- **Users tablespace** is used for storing tables created by users

# Storage Organisation in Oracle

- Physical storage is organised in terms of **data blocks, extents and segments**
- A **data block** is a fixed number of bytes
- An **extent** is a specific number of contiguous blocks
- A **segment** is a set of extents allocated to a specific data structure

# Storage Organisation in Oracle



# Storage Organisation in Oracle

- A **data block** has the following components
  - **Header** - contains general block information such as block address and type of segment
  - **Table Directory** - contains information about tables that have data in the data block
  - **Row Directory** - contains information about the actual rows

# **Storage Organisation in Oracle**

- A **data block** has the following components
  - **Row Data** - uses the bulk of the space in the data block
  - **Free Space** - space allocated for row updates and new rows

# Storage Organisation in Oracle

- Two space management parameters **PCTFREE** and **PCTUSED** can be used for controlling the use of free space
- **PCTFREE** sets the minimum percentage of a data block to be preserved as free space for possible updates to rows
- For example, **PCTFREE 30** states that 30% of every data block will be preserved as free space. Hence, after a data block is filled to 70%, new rows cannot be inserted

# Storage Organisation in Oracle

- **PCTUSED** sets the minimum percentage of a data block's space to be reached - due to **DELETE** and **UPDATE** statements that reduce the size of data - before new rows can be added to the block
- For example

**PCTUSED 50**

sets 50% as the minimum percentage and hence new rows can be added after the used space decreases from 70% to 50%

# Storage Organisation in Oracle

- If a row does not fit in a data block, we can use a chain of data blocks for storing it. This is called **row chaining**
- During updation, if a row becomes unfit to be stored in a data block, we can move to a new data block and try to fit it there. This is called **migration**

# Storage Organisation in Oracle

- The following are the different types of segments used in Oracle
  - Data Segments
  - Index Segments
  - Temporary Segments
    - used as a temporary work area for SQL statements
  - Rollback Segments
    - They hold old values of data which can be used during rollback

# Programming Oracle Applications

- Programming in Oracle is done in the following ways
  - Writing interactive SQL Queries
  - Writing programs in a host language such as C or COBOL. A precompiler such as PRO\*COBOL or PRO\*C is used to link the application to Oracle
  - Writing a PL/SQL which is Oracle's own procedural language
  - Using Oracle Call Interface(OCI) and the Oracle runtime library

# Programming Oracle Applications

- Programming in PL/SQL
  - PL/SQL is a block structured language
  - A block groups logically related declaration and statements
  - The declarations are local to the block
  - It offers data encapsulation, information hiding, overloading and exception handling

# Programming Oracle Applications

- Programming in PL/SQL
- Here data can be processed using conditional, iterative and sequential control flow statements such as IF..THEN..ELSE, FOR..LOOP, WHILE..LOOP, EXIT..WHEN and GO..TO

# Programming Oracle Applications

- Programming in PL/SQL
- A PL/SQL block has three parts
  1. Declaration part - where variables and objects are declared
  2. Executable part - where these variables are manipulated
  3. Exception part - where exceptions or errors can be handled

# Programming Oracle Applications

- Programming in PL/SQL
- A PL/SQL block has three parts

[ DECLARE

..... declarations]

BEGIN

..... statements

[ EXCEPTION

..... handlers]

END;

# Programming Oracle Applications

- Programming in PL/SQL
- Sample PL/SQL program

DECLARE

```
v_fname employee.fname%TYPE;
v_lname employee.lname%TYPE;
v_bdate employee.bdate%TYPE;
v_salary employee.salary%TYPE;
```

# Programming Oracle Applications

- Programming in PL/SQL
- Sample PL/SQL program - contd

BEGIN

```
 SELECT fname, lname, bdate, salary
 INTO v_fname, v_lname, v_bdate,v_salary
 FROM EMPLOYEE
 WHERE salary = (select max (salary)
 from employee);
 dbms_output.put_line(v_fname, v_lname,
 v_bdate, v_salary);
```

# Programming Oracle Applications

- Programming in PL/SQL
- Sample PL/SQL program - contd

EXCEPTION

WHEN OTHERS

```
 dbms_output.put_line('Error Detected');
```

END;

# Programming Oracle Applications

- Programming in PL/SQL
- Sample PL/SQL program - explanation
- The program variables may or may not have names identical to their corresponding attributes
- The %TYPE in each variable declaration means that the variable is of the same type as the corresponding attribute in the table

# Programming Oracle Applications

- Programming in PL/SQL
- Sample PL/SQL program - explanation
- The INTO clause specifies the program variables into which attribute values from the database are retrieved
- The exception handling part prints out an error message if an error is detected - in this case, if more than one employee is selected

# Programming Oracle Applications

- Cursors in PL/SQL
- A query can return zero or more rows depending on the search criteria
- When a query returns multiple rows, it is necessary to explicitly declare a cursor to process the rows
- A cursor is similar to a file variable or file pointer that points to a single row ( tuple ) from the result of a query

# Programming Oracle Applications

- Cursors in PL/SQL
- Sample PL/SQL Program
- It displays the SSN of employees whose salary is greater than their supervisor's salary

DECLARE

```
emp_salary NUMBER;
emp_super_salary NUMBER;
emp_ssn CHAR(9);
emp_superssn CHAR(9);
```

# Programming Oracle Applications

- Cursors in PL/SQL
- Sample PL/SQL Program - contd

```
CURSOR salary_cursor IS
 SELECT ssn, salary, superssn FROM
 employee;
BEGIN
 OPEN salary_cursor;
 LOOP
 FETCH salary_cursor INTO emp_ssn,
 emp_salary, emp_superssn;
```

# Programming Oracle Applications

- Cursors in PL/SQL
- Sample PL/SQL Program - contd

```
EXIT WHEN salary_cursor % NOT FOUND;
IF emp_superssn is NOT NULL THEN
 SELECT salary INTO emp_super_salary
 FROM employee
 WHERE ssn = emp_superssn;
 IF emp_salary > emp_super_salary
 THEN dbms_output.put_line (emp_ssn);
ENDIF;
```

# Programming Oracle Applications

- Cursors in PL/SQL
- Sample PL/SQL Program - contd

ENDIF;

END LOOP;

IF salary\_cursor%ISOPEN THEN CLOSE  
salary\_cursor;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line ('Errors with ssn ' ||  
emp\_ssn);

# Programming Oracle Applications

- Cursors in PL/SQL

- Sample PL/SQL Program - contd

```
IF salary_cursor % IS OPEN THEN CLOSE
 salary_cursor;
END;
```

- Here the cursor is initialised with the OPEN statement, which executes the query, retrieves the resulting set of rows and sets the cursor to the beginning

# Programming Oracle Applications

- Cursors in PL/SQL
- The `FETCH` statement when executed for the first time sets the cursor point to the first row and retrieves the corresponding values to the program variables
- Subsequent executions of the `FETCH` statement sets the cursor point to the next row in the result set and retrieves the values of that row to the program variables

# Programming Oracle Applications

- Cursors in PL/SQL
- When the last row has been processed, the cursor is released with the CLOSE statement
- Here %NOTFOUND is a cursor attribute which returns TRUE if a FETCH operation does not return a row
- %ISOPEN is another cursor attribute which returns TRUE if the cursor is already open
- Here the EXCEPTION part handles the situation where an incorrect supervisor ssn is assigned to an employee

# Programming Oracle Applications

- An example in PRO\*C
- PRO\*C is the precompiler for C language
- An Oracle precompiler is a programming tool using which SQL statements can be embedded in a high level programming language
- This precompiler translates all the embedded SQL statements into the native programming language which can then be compiled, linked and executed

# Programming Oracle Applications

- An example in PRO\*C
- PRO\*C provides automatic conversion between Oracle and C language data types
- Here both SQL statements and PL/SQL blocks can be embedded in a C host program

# Oracle Languages

- SQL
  - A special purpose programming language for managing data stored in a RDBMS
- PL/SQL
  - A procedural language to include control flow structures, to use variables and to provide error handling procedures

# Database Structure

- Physical Structure
  - Refers to physical organisation of data
- Logical Structure
  - Corresponds to abstract representation of stored data - a conceptual schema of the database

# Files in The Database

- Data Files
  - Files that contain the actual data
- Redo Log Files
  - Log files for redo based recovery
- Control Files
  - Files that contain control information like database name, file names, locations etc. Also used in the process of recovery
- Trace Files and Alert Log
  - Trace files track background processes
  - Alert log maintains major database events

# Files in The Database

- Log files and control files can be configured to be multiplexed
- That is, multiple copies of them can be written to multiple devices for resilience in the case of failures

# Database Organisation

- Schema Objects
  - Contain definitions of tables, views, sequences, stored procedures, indexes, clusters and database links
- Oracle Data Dictionary
  - A system catalog containing user names, security information, schema objects information, ICs, statistics, audit trails
- Table Spaces
  - Describe physical storage structures and govern how physical space of the database is used

# Oracle Instance

- Set of processes comprising an instance of the server's operation

An instance comprises of

- System Global Area (SGA)
  - Database Buffer Cache
  - Redo Log Buffer
  - Shared Pool
- User Processes

# Oracle Instance

- Program Global Area (PGA)
  - A memory buffer that contains data and control information for a server process
  
- Oracle Processes
  - Server Processes
  - Background processes

# Oracle Processes

## ■ Server Processes

- Handle requests from user processes
- Dedicated server configuration
- Multithreaded server configuration

## ■ Background Processes

- Created for each instance of Oracle
- Perform asynchronous I / O
- Provide parallelism for better performance and reliability

# SQL in Oracle

- Types of SQL statements
  - DDL Statements
  - DML Statements
  - Transaction Control Statements
  - Session Control Statements
  - System Control Statements
  - Embedded SQL Statements

# **Methods and Triggers in Oracle**

- Added as part of object - relational extension
- A method is a function that is part of the definition of a user - defined abstract data type
- Different from stored procedures
- Triggers - Active rule capability

# Storage Organisation

- Tablespace characteristics
  - Database divided into tablespaces
    - System Tablespace
    - User Tablespace
  - Data files in Tablespace
  - System Tablespace holds data dictionary objects

# Storage Organisation

- Physical Storage

- Data block ( page ) ( logical block )
  - Smallest level of granularity of storage
  - Fixed number of bytes
- Extents
  - An extent is a specific number of contiguous blocks
- Segments
  - A segment is a set of extents allocated to a specific data structure

# Data Blocks

- Contents of a data block

- Header
- Table Directory
- Row Directory
- Row Data
- Free Space

# Extents

- When a table is created Oracle allocates an initial extent to it
- When the initial extent gets full, extents are allocated incrementally
- Extents allocated in index segments remain allocated as long as index exists
- When index is dropped, Oracle will reclaim the space

# Segments

- A collection of segments belonging to a specific tablespace
- Types of Segments
  - Data Segments
    - Belongs to each non-clustered table and each cluster
  - Index Segments
    - Each index has a single index segment, created with the CREATE INDEX command

# Segments

- Temporary Segments
  - Provide a temporary work area
  - Some statements requiring temporary work area
    - CREATE INDEX
    - SELECT ..... { ORDER BY } { GROUP BY }
    - SELECT DISTINCT
- Rollback Segments
  - Used for undoing transactions