Unit-I

Databases and Databases users - Database system concepts and architecture - Data modeling using Entity Relationship (ER) Model. Relational Model - The Relational Data Model and Relational Database Constraints - The Relational Algebra and Relational Calculus.

Database

A database is a collection of related data. For example, name, telephone number and address of the people who are coming from foreign countries.

Implicit Properties of Database:

1. A database represents some aspect of the real world, sometimes called the miniworld or the Universe of Discourse (UoD). Changes to the miniworld are reflected in the database.
2. A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
3. A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

Database Management System

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database.

The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications.

Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database.

Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.

Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.

Data Element

Each fields of the record. For example, each STUDENT record includes data to represent the student's Name, Student Number(, Major (MATH, computer science or CS, . . .).

Characteristics of the Database Approach

1. Self-Describing Nature of a Database System
2. Insulation between Programs and Data, and Data Abstraction
3. Support of Multiple Views of the Data
4. Sharing of Data and Multiuser Transaction Processing

1. Self-Describing Nature of a Database System:

   database system contains a complete definition or description of the database structure and constraints. This definition is stored in the system catalog

   CataLog : Information about the structure of each file, the type and storage format of each data item, and various constraints on the data. The catalog is used by the DBMS software and also by database users who need information about the database structure.

   Meta Data: The Information stored in the catalog and it describes the structure of the primary database.

   In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

2. Insulation between Programs and Data, and Data Abstraction

   In traditional file processing, the structure of data files is embedded in the access programs, so any changes to the structure of a file may require changing all programs that access this file. But in DBMS access program do not need to changes I the most cases, the structure of data files are stored in the catalog. This property program –data independence.

   In object-oriented and object-relational databases, users can define operations on data as part of the database definitions. An operation is specified in two parts:

   a. The **interface** (or signature) of an operation includes the operation name and the data types of its arguments.
   b. The **implementation** (or method) of the operation is specified separately and can be changed without affecting the interface.
   c. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence**.

   The characteristic that allows program-data independence and program-operation independence is called **data abstraction**.

   A DBMS provides users with a **conceptual representation** of data that does not include the details of how the data is stored or how the operations are implemented.

   A **data model** is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts.

   For Example, A conceptual representation of the STUDENT records is shown data of the name of the student , number of the student, Course of the student. Many other details like how the data is stored and what is the access paths specified on a file can be hidden from database users by the DBMS.

3. Support of Multiple Views of the Data

   A database typically has many users, each of whom may require a different perspective or view of the database. A **view** may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.

4. Sharing of Data and Multiuser Transaction Processing

   A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database.

   The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

   For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger. These types of applications are generally called **on-line transaction processing (OLTP) applications**.

Actors on the Scene

1. Database Administrators
2. Database Designers
3. End Users
4. System Analysts and Application Programmers (Software Engineers)

1. Database Administrators

In a database environment, the primary resource is the database itself and the secondary resource is the DBMS and related software.

Database administrator (DBA): Responsible and Administering the database resources. The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed.

The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

2. Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users, in order to understand their requirements, and to come up with a design that meets these requirements.

The designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop **a view** of the database that meets the data and processing requirements of the group. These views are then analyzed and integrated with the views of other user groups. The

final database design must be capable of supporting the requirements of all user groups.

3. End Users

End users are the people who access to the database for querying, updating, and generating reports.

Catagories :

1. Casual end users
2. Naive or parametric end users
3. Sophisticated end users
4. Stand-alone users

1. **Casual end users - occasionally access the database**, but they may need different information each time. They use a sophisticated database query language to specify their requests.

They are typically middle or high-level managers or other occasional browsers.

2. **Naive or parametric end users -** make up a sizable portion of database end users. Their main job is constantly querying and updating the database, using standard types of queries and updates that have been carefully programmed and tested is **called canned transactions**.
   The tasks that such users perform are varied: Bank tellers check account balances and post withdrawals and deposits.
   Reservation clerks for airlines, hotels, and car rental companies check availability for a given request and make reservations
3. **Sophisticated end users** - include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS and to implement their applications to meet their complex requirements.
4. **Stand-alone users -** maintain personal databases by using ready-made program packages that provide easy-to-use menu- or graphics-based interfaces.
   An example is the user of a tax package that stores a variety of personal financial data for tax purposes

4. System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers.

Software Engineers should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

Workers behind the Scene

The people who are associated with the design, development, and operation of the DBMS software and system environment. These persons are typically not interested in the database itself. We call them the "**workers behind the scene,".**

Catagories:

1. DBMS system designers and implementers
2. Tool developers
3. Operators and maintenance personnel

1. **DBMS system designers and implementers**
   The persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a complex software system that consists of many components or modules, including modules for implementing the catalog, query language, interface processors, data access, concurrency control, recovery, and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.
2. **Tool developers**
   The persons who design and implement tools—the software packages that facilitate database system design and use, and help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation.
3. **Operators and maintenance personnel**
   The people who are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Advantages of Using a DBMS

1. Controlling Redundancy
2. Restricting Unauthorized Access
3. Providing Persistent Storage for Program Objects and Data Structures
4. Permitting Inferencing and Actions Using Rules
5. Providing Multiple User Interfaces
6. Representing Complex Relationships Among Data
7. Enforcing Integrity Constraints
8. Providing Backup and Recovery

1. **Controlling Redundancy**

   In File Processing System, duplicate data is created in many places because all the programs have their own files. This creates data redundancy which in turns wastes labor and space. In Database Management System, all the files are integrated in a single database. The whole data is stored only once at a single place so there is no chance of duplicate data.
   For example: A student record in library or examination can contain duplicate values, but when they are converted into a single database, all the duplicate values are removed.

2. **Restricting Unauthorized Access**
   Data security means protecting your precious data from unauthorized access. Data in database should be kept secure and safe to unauthorized modifications. Only authorized users should have the grant to access the database. There is a username set for all the users who access the database with password so that no other guy can access these information. DBMS always keep database tamperproof, secure and theft free.

3. **Providing Persistent Storage for Program Objects and Data Structures**
   Databases can be used to provide persistent storage for program objects and data structures. This is one of the main reasons for the emergence of the object-oriented database systems. The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so-called impedance mismatch problem, since the data structures provided by the DBMS were incompatible with the programming language's data structures.
   Data loss is a very big problem for all the organizations. In traditional file processing system, a user needs to back up the database after a regular interval of time that wastes lots of time and resources. If the volume of data is large then this process may take a very long time.
   DBMS solves this problem of taking back up again and again because it allows automatic backup and recovery of database. For examples, if a system fails in the middle of any process then DBMS stores the values of that state in which database were before query execution.

4. **Permitting Inferencing and Actions Using Rules**

   Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called deductive database systems.

   For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified declaratively as rules, which when compiled and maintained by the DBMS can determine all students on probation.

5. **Providing Multiple User Interfaces**
   Many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users; programming language interfaces for application programmers; forms and

command codes for parametric users; and menu-driven interfaces and natural language interfaces for stand-alone users. Both forms-style interfaces and menu-driven interfaces are commonly known as graphical user interfaces (GUIs).

6. Representing Complex Relationships Among Data

A database may include numerous varieties of data that are interrelated in many ways. A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data easily and efficiently.

For example, The record for Brown in the student file is related to four records in the GRADE_REPORT file. Similarly, each section record is related to one course record as well as to a number of GRADE_REPORT records—one for each student who completed that section.

7. Enforcing Integrity Constraints

Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. All of these databases contain data that is visible to multiple users. So it is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

8. Providing Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure to its previous condition.

Database System Concepts and Architecture

Data Models, Schemas, and Instances

Data Model

It defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

Categories of Data Models

1. High-level or conceptual data models
2. Low-level or physical data models
3. Representational (or implementation) data model

1. High-level or Conceptual Data Models

The conceptual data model is a structured business view of the data required to support business processes, record business events, and track related performance measures.

This model focuses on identifying the data used in the business but not it's processing flow or physical characteristics.

2. Low-level or physical data models
   To provide concepts that describe the details of how data is stored in the computer.
3. Representational (or implementation) data model
   To provide concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer.

Entity: it represents a real-world object or concept, such as an employee or a project, that is described in the database.

Attribute: It represents some property of interest that further describes an entity, such as the employee's name or salary.

Relationship: Two or more entities represents an interaction among the entities.

for example, a works-on relationship between an employee and a project.

Schemas, Instances, and Database State

Database schema - The description of a database.

Schema diagram - Diagrammatical representation of database.

Database state or Snapshot - The data in the database at a particular moment in time. It is also called the current **set of occurrences or instances** in the database.

DBMS Architecture and Data Independence

The Three-Schema Architecture

3 levels,

1. The **internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.
3. The **external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

Data Independence

The three-schema architecture can be used to explain the concept of data independence,

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs.

2. Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas.

Database Languages and Interfaces

DBMS Language: Data Definition Language (DDL)

Storage Definition Language(SDL)

View Definition Language (VDL)

Data Manipulation Language (DML)

DBMS Interfaces

1. Menu-Based Interfaces for Browsing
2. Forms-Based Interfaces
3. Graphical User Interfaces
4. Natural Language Interfaces
5. Interfaces for Parametric Users
6. Interfaces for the DBA

1. Menu-Based Interfaces for Browsing
   These interfaces present the user with lists of options, called menus that lead the user through the formulation of a request. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step by step by picking options from a menu that is displayed by the system. Pull-down menus are becoming a very popular technique in window-based user interfaces.
2. Forms-Based Interfaces
   A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.
3. Graphical User Interfaces
   A graphical interface (GUI) typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to pick certain parts of the displayed schema diagram.
4. Natural Language Interfaces
   These interfaces accept requests written in English or some other language and attempt to "understand" them. A natural language interface usually has its own "schema," which is similar to the database conceptual schema. The natural language interface refers to the words in its schema, as well as to a set of standard words, to interpret the request.
5. Interfaces for Parametric Users
   Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. Systems analysts and programmers design and implement a

special interface for a known class of naive users. Usually, a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

6. Interfaces for the DBA
   Most database systems contain privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

The Database System Environment

DBMS Component Modules

The **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names of files, data items, storage details of each file, mapping information among schemas, and constraints, in addition to many other types of information that are needed by the DBMS modules.

The **run-time database processor** handles database accesses at run time; it receives retrieval or update operations and carries them out on the database. Access to disk goes through the stored data manager. The **query compiler** handles high-level queries that are entered interactively. It parses, analyzes, and compiles or interprets a query by creating database access code, and then generates calls to the run-time processor for executing the code.

Database System Utilities

Loading: A loading utility is used to load existing data files such as text files or sequential files into the database.

Backup: A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape.

File reorganization: This utility can be used to reorganize a database file into a different file organization to improve performance.

Performance monitoring: Such a utility monitors database usage and provides statistics to the DBA.

Data Modeling Using the Entity Relationship Model

Entity Types, Entity Sets, Attributes, and Keys

Entities and Attributes

An entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence—a particular person, car, house, or employee.

Attributes—the particular properties that describe it. For example, an employee entity may be described by the employee's name, age, address, salary, and job.

Composite versus Simple (Atomic) Attributes

Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the employee entity.

Single-valued Versus Multivalued Attributes

Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a single-valued attribute of person. In some cases an attribute can have a set of values.

Stored Versus Derived Attributes

The Age attribute is hence called a derived attribute and is said to be derivable from the BirthDate attribute, which is called a stored attribute.

Null Values

The attribute not having values.

For Example, a person with no college degree would have null for College Degrees.

Complex Attributes

Arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces {}. Such attributes are called complex attributes.

Entity Types, Entity Sets, Keys, and Value Sets

Entity Types and Entity Sets

An **Entity type** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

Entity Set - The collection of all entities of a particular entity type in the database at any point in time is called an entity set.

Key Attributes - An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute.

Value Set - Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

In the ER model, these references should not be represented as attributes but as relationships, which are discussed in this section. The **COMPANY** database **schema** will be refined to represent **relationships** explicitly. In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

Relationship Types, Sets, and Instances

A **relationship** type **R** among **n** entity types **E1, E2, ..., En** defines a set of **associations**—or a **relationship** set—among entities from these entity types. As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the same name, R.

Mathematically, the relationship set **R** is a set of **relationship** instances r_i, where each r_i associates n individual entities (e1, e2, ..., en), and each entity e_j in r_i is a member of entity set E_j, 1 ? j ? n. Hence, a relationship set is a mathematical relation on E1, E2, ..., En; alternatively, it can be defined as a subset of the Cartesian product of the entity sets E1 × E2 × ... × En.

Relationships

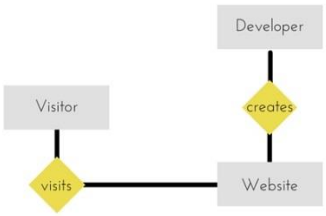
When an Entity is related to another Entity, they are said to have a relationship. For example, A **Class** Entity is related to **Student** entity, because students study in classes, hence this is a relationship.

Depending upon the number of entities involved, a **degree** is assigned to relationships.

For example, if 2 entities are involved, it is said to be **Binary relationship**, if 3 entities are involved, it is said to be **Ternary** relationship, and so on.

ER Diagram is a visual representation of data that describes how data is related to each other. In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.

For example, in the below diagram, anyone can see and understand what the diagram wants to convey: *Developer develops a website, whereas a Visitor visits a website*.



Components of ER Diagram

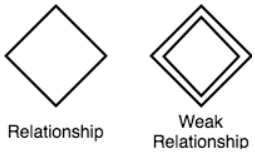
Entitiy, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.

Entity

Simple rectangular box represents an Entity.

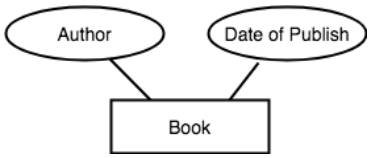
Relationships between Entities - Weak and Strong

Rhombus is used to setup relationships between two or more entities.



Attributes for any Entity

Ellipse is used to represent attributes of any entity. It is connected to the entity.



Weak Entity

A weak Entity is represented using double rectangular boxes. It is generally connected to another entity.

Key Attribute for any Entity

To represent a Key attribute, the attribute name inside the Ellipse is underlined.

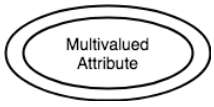
Derived Attribute for any Entity

Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

To represent a derived attribute, another dotted ellipse is created inside the main ellipse.

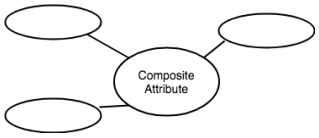
Multivalued Attribute for any Entity

Double Ellipse, one inside another, represents the attribute which can have multiple values.



Composite Attribute for any Entity

A composite attribute is the attribute, which also has attributes.



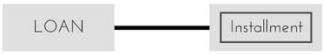
ER Diagram: Entity

An **Entity** can be any object, place, person or class. In ER Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.

The yellow rhombus in between represents a relationship.

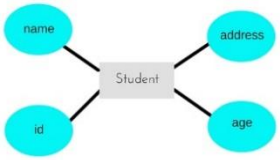
ER Diagram: Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doesn't have anay key attribute of its own. Double rectangle is used to represent a weak entity.



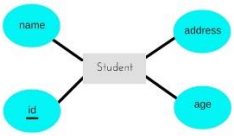
ER Diagram: Attribute

An **Attribute** describes a property or characterstic of an entity. For example, **Name**, **Age**, **Address** etc can be attributes of a **Student**. An attribute is represented using eclipse.



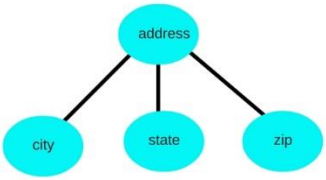
ER Diagram: Key Attribute

Key attribute represents the main characterstic of an Entity. It is used to represent a Primary key. Ellipse with the text underlined, represents Key Attribute.



ER Diagram: Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attributes.



ER Diagram: Relationship

A Relationship describes relation between **entities**. Relationship is represented using diamonds or rhombus.

There are three types of relationship that exist between Entities.

1. Binary Relationship
2. Recursive Relationship
3. Ternary Relationship

ER Diagram: Binary Relationship

Binary Relationship means relation between two Entities. This is further divided into three types.

One to One Relationship

This type of relationship is rarely seen in real world.

The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.

One to Many Relationship

The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.

Many to One Relationship

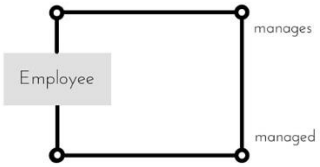
It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.

Many to Many Relationship

The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

ER Diagram: Recursive Relationship

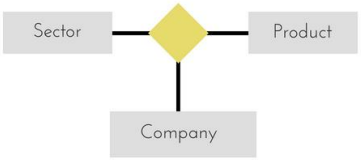
When an Entity is related with itself it is known as **Recursive** Relationship.



ER Diagram: Ternary Relationship

Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon the third.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

For example, in the diagram above, we have three related entities, **Company**, **Product** and **Sector**. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.

A **Company** produces many **Products**/ each product is produced by exactly one company.

A **Company** operates in only one **Sector** / each sector has many companies operating in it.

The Enhanced ER Model

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

Hence, as part of the **Enhanced ER Model**, along with other improvements, three new concepts were added to the existing ER Model, they were:

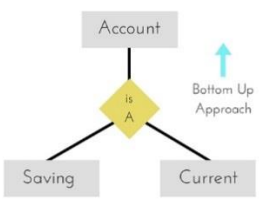
1. Generalization
2. Specialization
3. Aggregration

Let's understand what they are, and why were they added to the existing ER Model.

Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

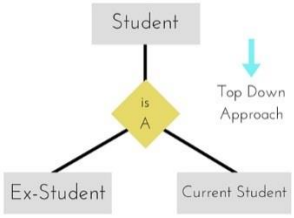
It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a super-class.



For example, **Saving** and **Current** account types entities can be generalised and an entity with name **Account** can be created, which covers both.

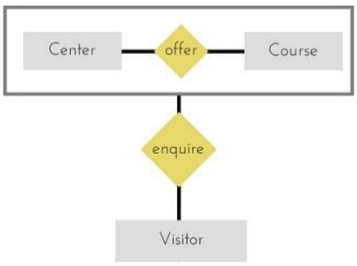
Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.



Aggregration

Aggregration is a process when relation between two entities is treated as a **single entity**.

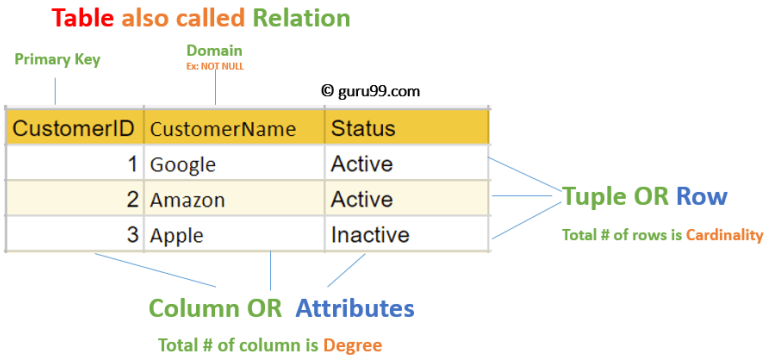


In the diagram above, the relationship between **Center** and **Course** together, is acting as an Entity, which is in relationship with another entity **Visitor**. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.

8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain



Relational Integrity constraints

Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain constraints
2. Key constraints
3. Referential integrity constraints

Domain Constraints
Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName
CHECK (value not NULL)
```

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

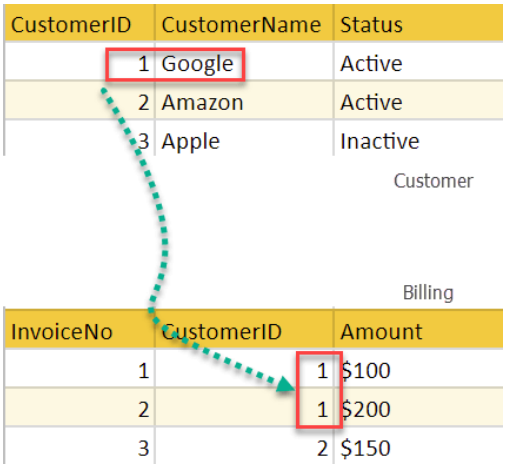
Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

Referential integrity constraints

Referential integrity constraints is base on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300

Operations in Relational Model

Four basic update operations performed on relational database model are

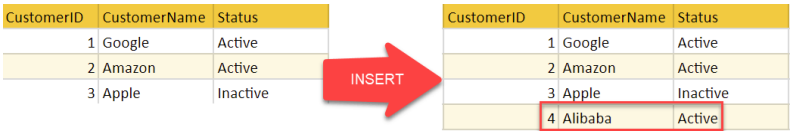
Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

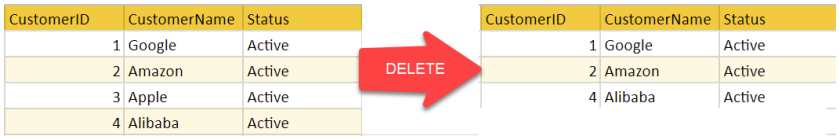
You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

Relational Algebra Operations

Relational Algebra devided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol:)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Let's study them in detail:

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

Example 1

$\sigma_{\text{topic = "Database"}}$ (Tutorials)

Output - Selects tuples from Tutorials where topic = 'Database'.

Example 2

$\sigma_{\text{topic = "Database" and author = "guru99"}}$(Tutorials)

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.

Example 3

$\sigma_{\text{sales > 50000}}$ (Customers)

Output - Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Union operation (υ)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A		Table B	
column 1	**column 2**	**column 1**	**column 2**
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
column 1	**column 2**
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A-B

Table A - B

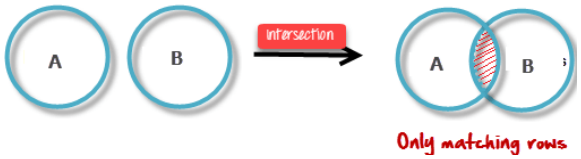
column 1	column 2
1	2

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Example:

A ∩ B

Table A ∩ B

column 1	column 2
1	1

Cartesian product(X)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

Example – Cartesian product

$\sigma_{\text{column 2} = \text{'1'}}(A \times B)$

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)	
column 1	**column 2**
1	1
1	1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

$A \bowtie_\theta B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

A ⋈ A.column 2 > B.column 2 (B)

column 1	column 2
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

A ⋈ A.column 2 = B.column 2 (B)

column 1	column 2
1	1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	27

C ⋈ D

C ⋈ D		
Num	**Square**	**Cube**
2	4	4
3	9	27

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Consider the following 2 Tables

A	
Num	**Square**
2	4
3	9
4	16

B	
Num	**Cube**
2	8
3	18
5	75

A ⋈ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	4

3	9	9
4	16	-

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

A ⟖ B

A ⋈ B		
Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	-

Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**

2	4	8
3	9	18
4	16	-
5	-	75

Operation	**Purpose**
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.

Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Relational Calculus

Relational Calculus in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

In tuple relational calculus, we work on filtering tuples based on the given condition.

Syntax: { T | Condition }

In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.

We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.

A lot of information, right! Give it some time to sink in.

A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

To specify the name of the relation(table) in which we want to look for data, we do the following:

Relation(T), where T is our tuple variable.

For example if our table is **Student**, we would put it as Student(T)

Then comes the condition part, to specify a condition applicable for a particular attribute(column), we can use the . dot variable with the tuple variable to specify it, like in table **Student**, if we want to get data for students with age greater than 17, then, we can write it as,

T.age > 17, where T is our tuple variable.

Putting it all together, if we want to use Tuple Relational Calculus to fetch names of students, from table **Student**, with age greater than **17**, then, for T being our tuple variable,

T.name | Student(T) AND T.age > 17

Domain Relational Calculus (DRC)

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

For example,

{< name, age > | ∈ Student ∧ age > 17}

Again, the above query will return the names and ages of the students in the table **Student** who are older than 17.

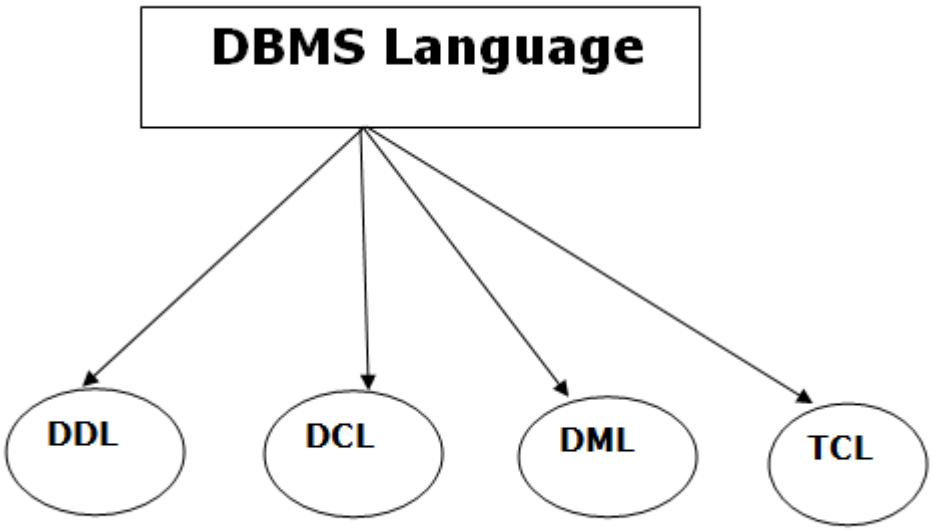
Unit II

Overview of the QBE Language - Overview of the Hierarchical Data Model - Overview of the Network Data Model - SQL-99: Schema Definition, Constraints, Queries, and Views- Functional Dependencies and Normalization for Relational Databases.

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

Overview of Hierarchical Data Model:

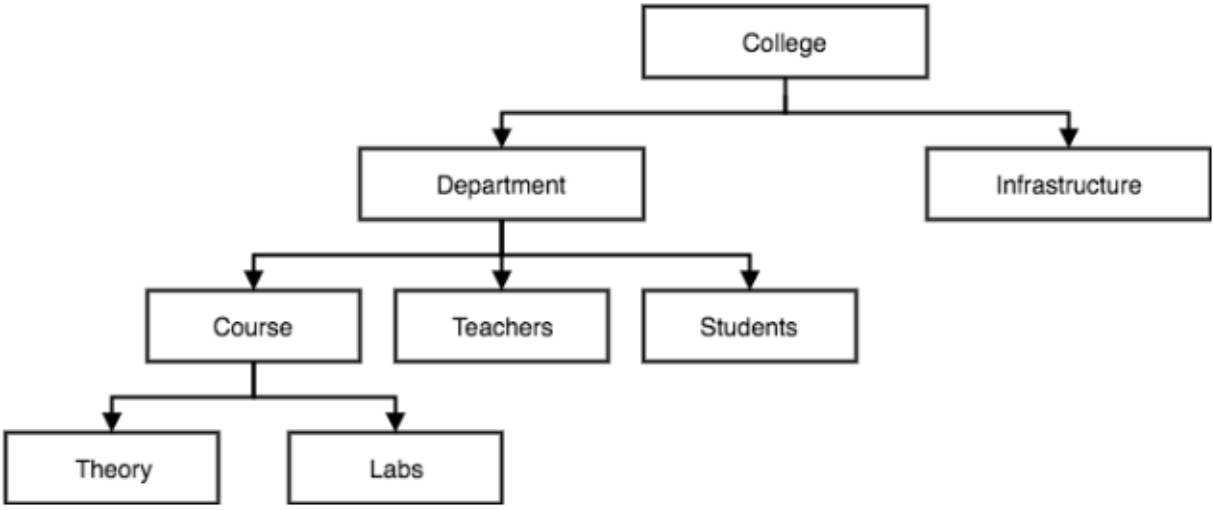
Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

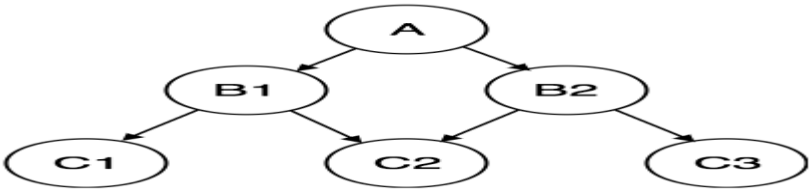
This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.



Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.



Network Model of database

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.

SQL:

Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database. Today almost all RDBMS(MySql, Oracle, Infomix, Sybase, MS Access) use **SQL** as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

SQL Command

SQL defines following ways to manipulate data stored in an RDBMS.

DDL: Data Definition Language

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc.

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

DML: Data Manipulation Language

DML commands are used for manipulating the data stored in the table and not the table itself.

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	to insert a new row

update	to update existing row
delete	to delete a row
merge	merging two rows or two tables

TCL: Transaction Control Language

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling the data back to its original state. It can also make any temporary change permanent.

Command	**Description**
commit	to permanently save
rollback	to undo change
savepoint	to save temporarily

DCL: Data Control Language

Data control language are the commands to grant and take back authority from any database user.

Command	**Description**
grant	grant permission of right
revoke	take back permission.

DQL: Data Query Language

Data query language is used to fetch data from tables based on conditions that we can easily apply.

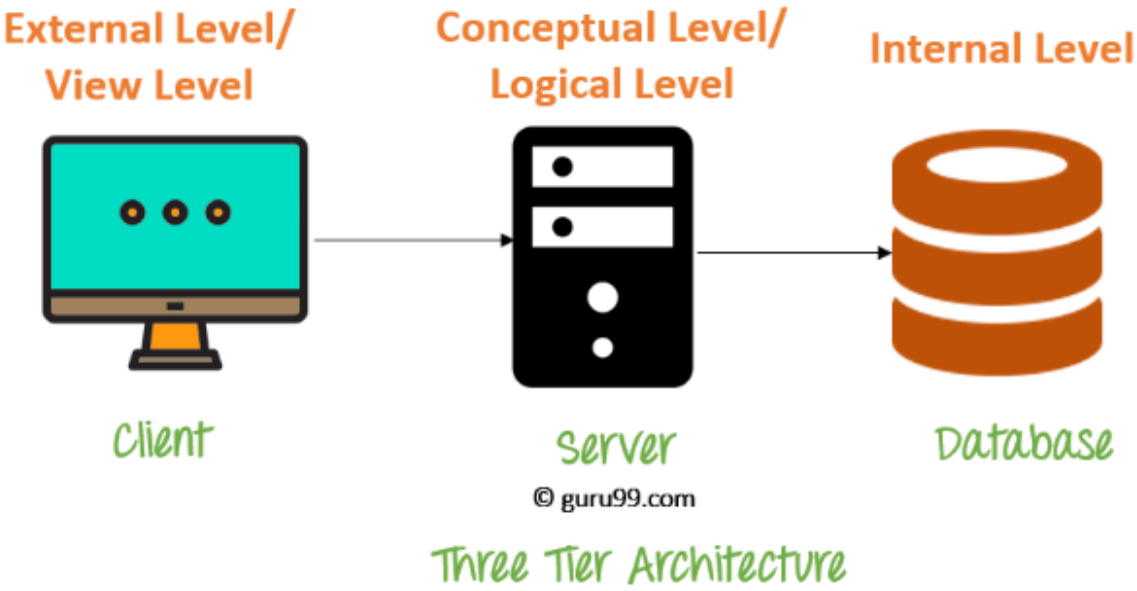
Command	Description
select	retrieve records from one or more table

Schema Definition:

Database systems comprise of complex data structures. Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

There are mainly three levels of data abstraction:

1. Internal Level: Actual PHYSICAL storage structure and access paths.
2. Conceptual or Logical Level: Structure and constraints for the entire database
3. External or View level: Describes various user views



Internal Level/Schema

The internal schema defines the physical storage structure of the database. The internal schema is a very low-level representation of the entire database. It contains multiple occurrences of multiple types of internal record. In the ANSI term, it is also called "stored record'.

Facts about Internal schema:

- The internal schema is the lowest level of data abstraction
- It helps you to keeps information about the actual representation of the entire database. Like the actual storage of the data on the disk in the form of records
- The internal view tells us what data is stored in the database and how

- It never deals with the physical devices. Instead, internal schema views a physical device as a collection of physical pages

Conceptual Schema/Level

The conceptual schema describes the Database structure of the whole database for the community of users. This schema hides information about the physical storage structures and focuses on describing data types, entities, relationships, etc.

This logical level comes between the user level and physical storage view. However, there is only single conceptual view of a single database.

Facts about Conceptual schema:

- Defines all database entities, their attributes, and their relationships
- Security and integrity information
- In the conceptual level, the data available to a user must be contained in or derivable from the physical level

External Schema/Level

An external schema describes the part of the database which specific user is interested in. It hides the unrelated details of the database from the user. There may be "n" number of external views for each database.

Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

An external view is just the content of the database as it is seen by some specific particular user. For example, a user from the sales department will see only sales related data.

Facts about external schema:

- An external level is only related to the data which is viewed by specific end users.
- This level includes some external schemas.
- External schema level is nearest to the user
- The external schema describes the segment of the database which is needed for a certain user group and hides the remaining details from the database from the specific user group

Goal of 3 level/schema of Database

Here, are some Objectives of using Three schema Architecture:

- Every user should be able to access the same data but able to see a customized view of the data.
- The user need not to deal directly with physical database storage detail.
- The DBA should be able to change the database storage structure without disturbing the user's views
- The internal structure of the database should remain unaffected when changes made to the physical aspects of storage.

Advantages Database Schema

- You can manage data independent of the physical storage
- Faster Migration to new graphical environments
- DBMS Architecture allows you to make changes on the presentation level without affecting the other two layers
- As each tier is separate, it is possible to use different sets of developers
- It is more secure as the client doesn't have direct access to the database business logic
- In case of the failure of the one-tier no data loss as you are always secure by accessing the other tier

Disadvantages Database Schema

- Complete DB Schema is a complex structure which is difficult to understand for every one
- Difficult to set up and maintain
- The physical separation of the tiers can affect the performance of the Database

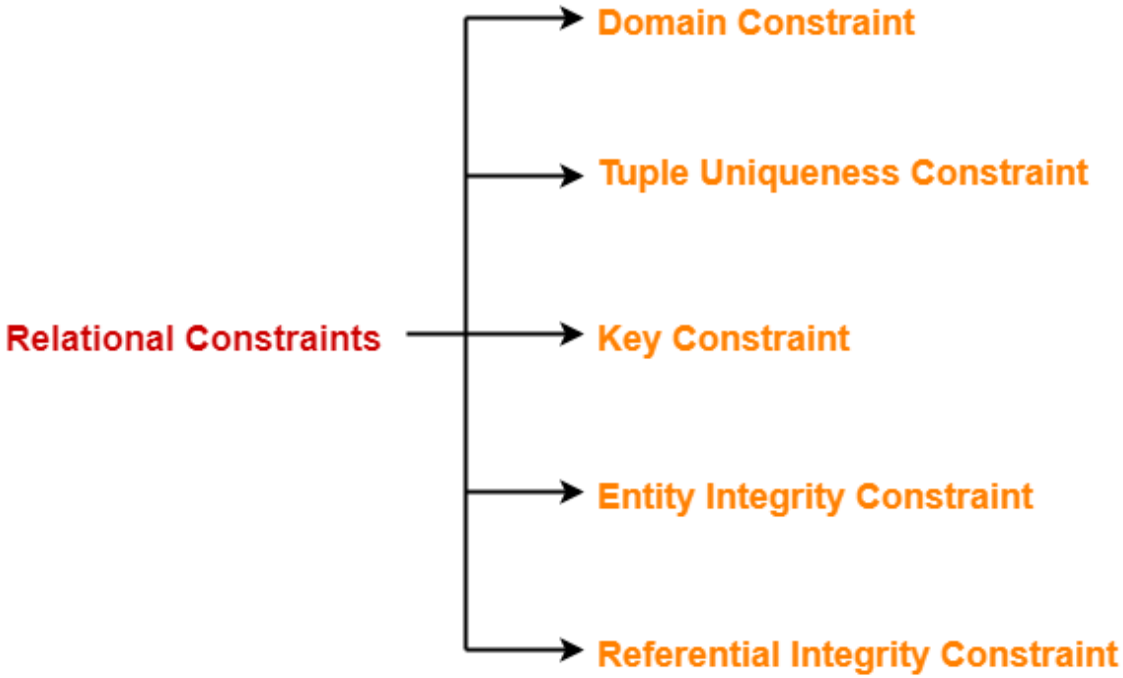
Constraints in DBMS-

Relational constraints are the restrictions imposed on the database contents and operations.

They ensure the correctness of data in the database.

Types of Constraints in DBMS-

In DBMS, there are following 5 different types of relational constraints-



1. Domain constraint
2. Tuple Uniqueness constraint
3. Key constraint
4. Entity Integrity constraint
5. Referential Integrity constraint

1. Domain Constraint-

- Domain constraint defines the domain or set of values for an attribute.
- It specifies that the value taken by the attribute must be the atomic value from its domain.

Example-

Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20

S002	Abhishek	21
S003	Shashank	20
S004	Rahul	**A**

Here, value **'A'** is not allowed since only integer values can be taken by the age attribute.

3. Key Constraint-

Key constraint specifies that in any relation-

- All the values of primary key must be unique.
- The value of primary key must not be null.

Example-

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

4. Entity Integrity Constraint-

- Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
- This is because the presence of null value in the primary key violates the uniqueness property.

Example-

Consider the following Student table-

STU_ID	**Name**	**Age**
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

5. Referential Integrity Constraint-

- This constraint is enforced when a foreign key references the primary key of a relation.
- It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

mportant Results-

The following two important results emerges out due to referential integrity constraint-

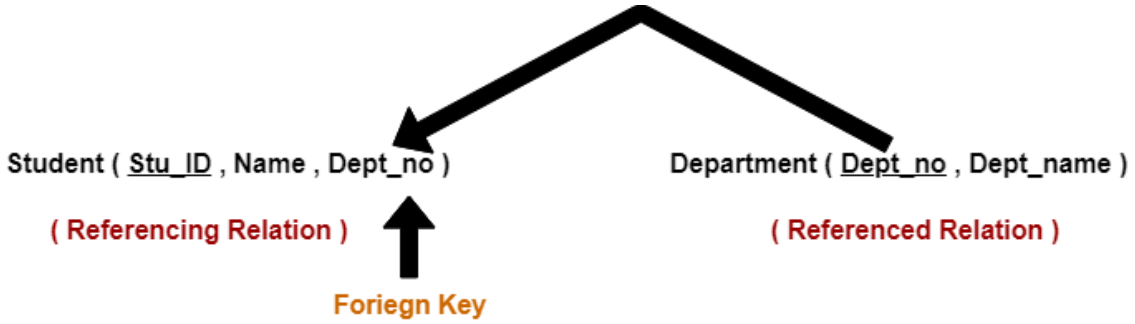
- We can not insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.

- We can not delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

Example-

Consider the following two relations- 'Student' and 'Department'.

Here, relation 'Student' references the relation 'Department'.



Student

STU_ID	**Name**	**Dept_no**
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	**D14**

Department

Dept_no	**Dept_name**
D10	ASET

D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation 'Student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department', no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

Queries and views:

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
1. CREATE VIEW view_name AS
2. SELECT column1, column2.....
3. FROM table_name
4. WHERE condition;
```

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
1. CREATE VIEW DetailsView AS
2. SELECT NAME, ADDRESS
3. FROM Student_Details
4. WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
1. SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida

David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
1. CREATE VIEW MarksView AS
2. SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
3. FROM Student_Detail, Student_Mark
4. WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
1. SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
1. DROP VIEW view_name;
```

Example:

If we want to delete the View **MarksView**, we can do this as:

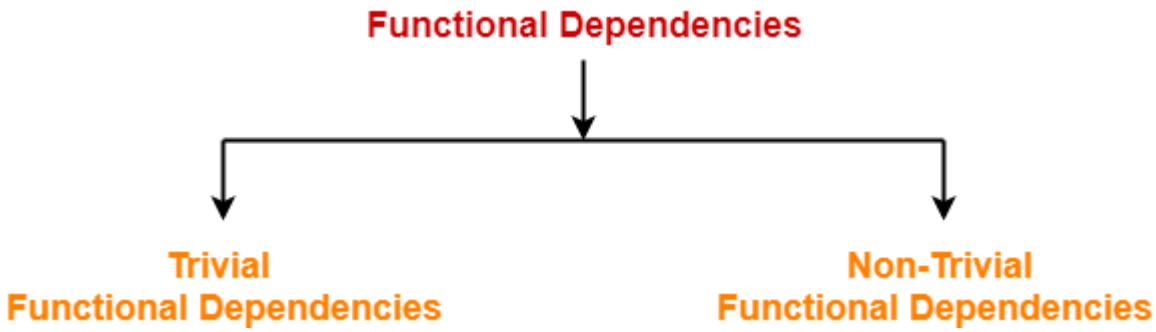
```
1. DROP VIEW MarksView;
```

Functional Dependency

Functional Dependency (FD) determines the relation of one attribute to another attribute in a database management system (DBMS) system. Functional dependency helps you to maintain the quality of data in the database. A functional dependency is denoted by an arrow →. The functional dependency of X on Y is represented by X → Y. Functional Dependency plays a vital role to find the difference between good and bad database design.

Types Of Functional Dependencies-

There are two types of functional dependencies-



1. Trivial Functional Dependencies
2. Non-trivial Functional Dependencies

Example:

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Rules of Functional Dependencies

Below given are the Three most important rules for Functional Dependency:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Types of Functional Dependencies

- **Multivalued dependency**
- **Trivial functional dependency**
- **Non-trivial functional dependency**
- **Transitive dependency**

Multivalued dependency in DBMS

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

Example:

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue

H010	2015	Metallic
H033	2012	Gray

In this example, maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multivalue dependent on car_model.

This dependence can be represented like this:

car_model -> maf_year

car_model-> colour

Trivial Functional dependency:

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X.

For example:

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Consider this table with two columns Emp_id and Emp_name.

{Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non trivial functional dependency in DBMS

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Example:

(Company} -> {CEO} (if we know the Company, we knows the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive dependency:

A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} -> {CEO} (if we know the compay, we know its CEO's name)

{CEO } -> {Age} If we know the CEO, we know the Age

Therefore according to the rule of rule of transitive dependency:

{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Note: You need to remember that transitive dependency can only occur in a relation of three or more attributes.

Advantages of Functional Dependency

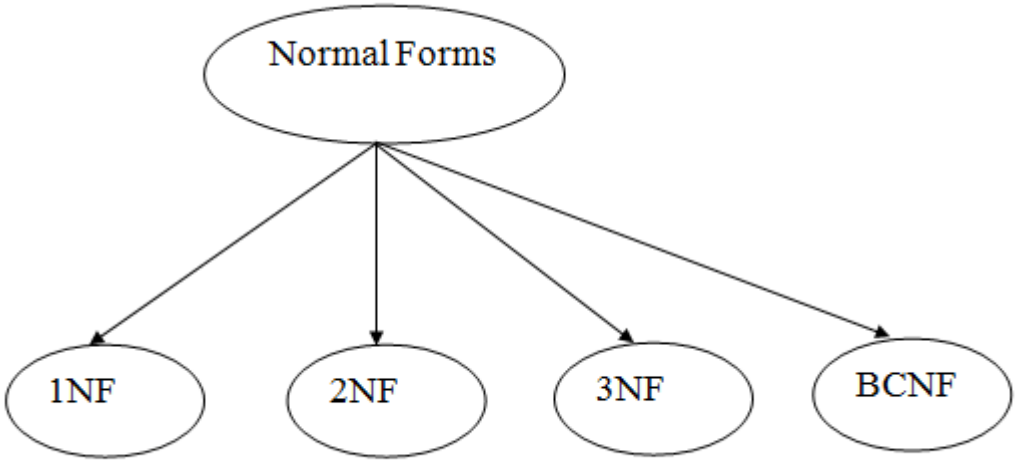
- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

```
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

```
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

UNIT III

Algorithms for Query Processing and Optimization - Introduction to Transaction Processing Concepts and Theory - Concurrency control techniques.

QUERY PROCESSING

3.1 Algorithms for Query Processing and Optimization

Query Processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

The steps involved in processing a query appear in Figure 3.1.1. The basic steps are:

1. Parsing and translation.
2. Optimization.
3. Evaluation.

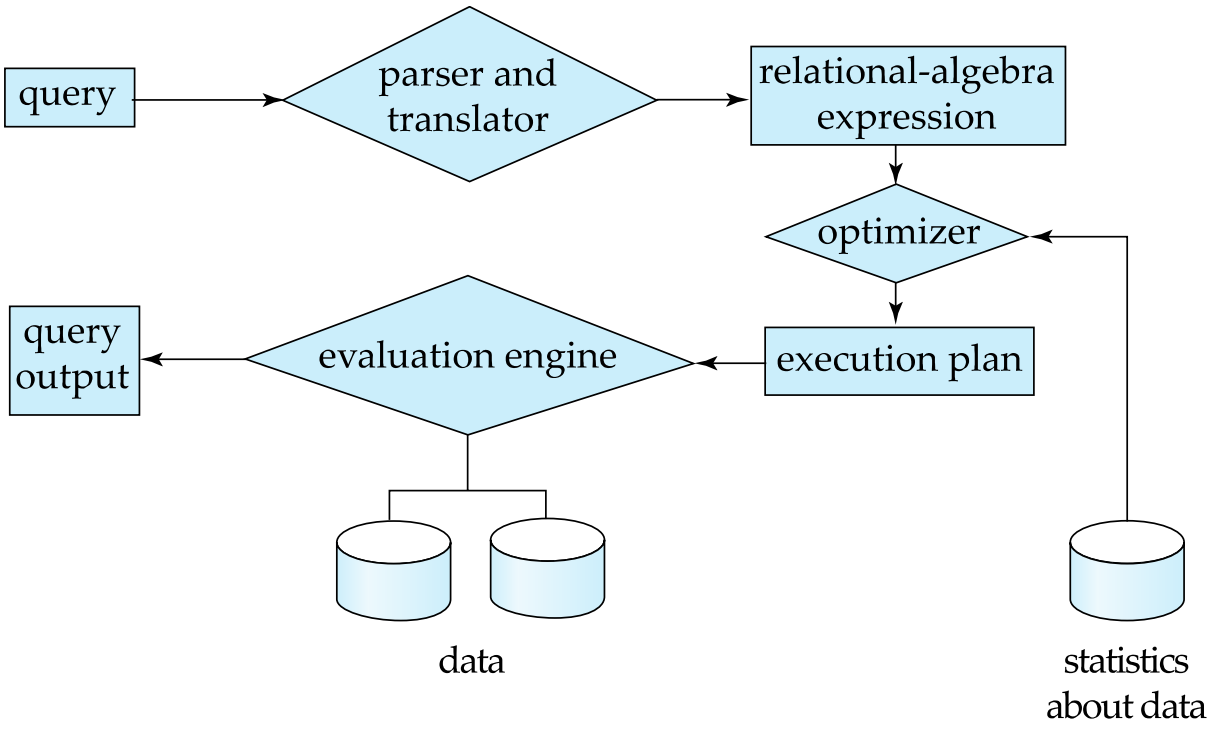


Fig. 3.1.1 Typical steps when processing a high level query.

1. Parsing and translation

- Translate the query into its internal form. This is then translated into relational algebra.
- Parser checks syntax, verifies relation.

2. Optimization

- SQL is a very high level language:
  - The users specify what to search for - not how the search is actually done

  - The algorithms are chosen automatically by the DBMS.
- For a given SQL query there may be many possible execution plans.
- Amongst all equivalent plans choose the one with lowest cost.
- Cost is estimated using statistical information from the database catalog.

3. Evaluation

- The query evaluation engine takes a query evaluation plan, executes that plan and returns the answer to that query.

As an illustration, consider the query:

select salary from instructor where salary < 75000;

This query can be translated into either of the following relational-algebra expressions:

$$\sigma_{salary\ <75000}\ (\Pi_{salary}\ (instructor))$$

(or)

$$\Pi_{salary}\ (\sigma_{salary<75000}\ (instructor))$$

Further, each relational-algebra operation can be executed by one of several different algorithms. For example, to implement the preceding selection, we can search every tuple in instructor to find tuples with salary less than 75000. If a B^+ tree index is available on the attribute salary, we can use the index instead to locate the tuples. To specify fully how to evaluate a query, providing the relational-algebra expression will not be anought, but also to annotate it with instructions specifying how to evaluate each operation.

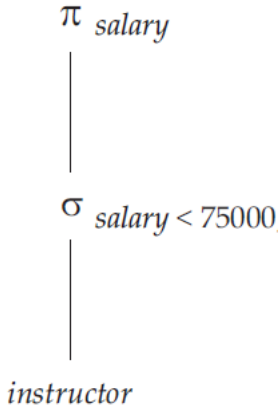


Figure 3.1.2 A query-evaluation plan

A sequence of primitive operations that can be used to evaluate a query is a query-execution plan or query-evaluation plan. Figure 3.1.2 illustrates an evaluation plan for our example query. The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query. The different evaluation plans for a given query can have different costs. It is the responsibility of the system to construct a query evaluation plan that minimizes the cost of query evaluation; this task is called query optimization. Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output. In order to optimize a query, a query optimizer must know the cost of each operation.

Measures of Query Cost

The cost of query evaluation can be measured in terms of a number of different resources, including disk accesses, CPU time to execute a query and in a distributed or parallel database system, the cost of communication.

The response time for a query-evaluation plan could be used as a good measure of the cost of the plan. In large database systems, however, disk accesses are usually the most important cost, since disk accesses are slow compared to in-memory operations. Most people consider the disk accesses cost a reasonable measure of the cost of a query-evaluation plan.

The number of block transfers from disk is also used as a measure of the actual cost. It takes more time to write a block to disk than to read a block from disk. For more accurate measure, find:

i. The number of seek operations performed
ii. The number of blocks read
iii. The number of blocks written

and then add up these numbers after multiplying them by average seek time, average transfer time for reading a block and average transfer time for writing a block respectively.

3.1.1 Using Heuristics in Query Optimization

Query Optimization - Heuristics

Query Trees and Graphs

```
select  P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
from    Project as P, Department as D, Employee as E
where   P.Dnum=D.Dnumber and D.Mgr_ssn=E.Ssn and P.Plocation='Stafford';
```

$\pi_{Pnumber, Dnum, Lname, Address, Bdate}[[(\sigma_{Plocation='Stafford'}(Project)) \bowtie_{Dnum=Dnumber} (Department)] \bowtie_{Mgr\_ssn=Ssn} (Employee)]$

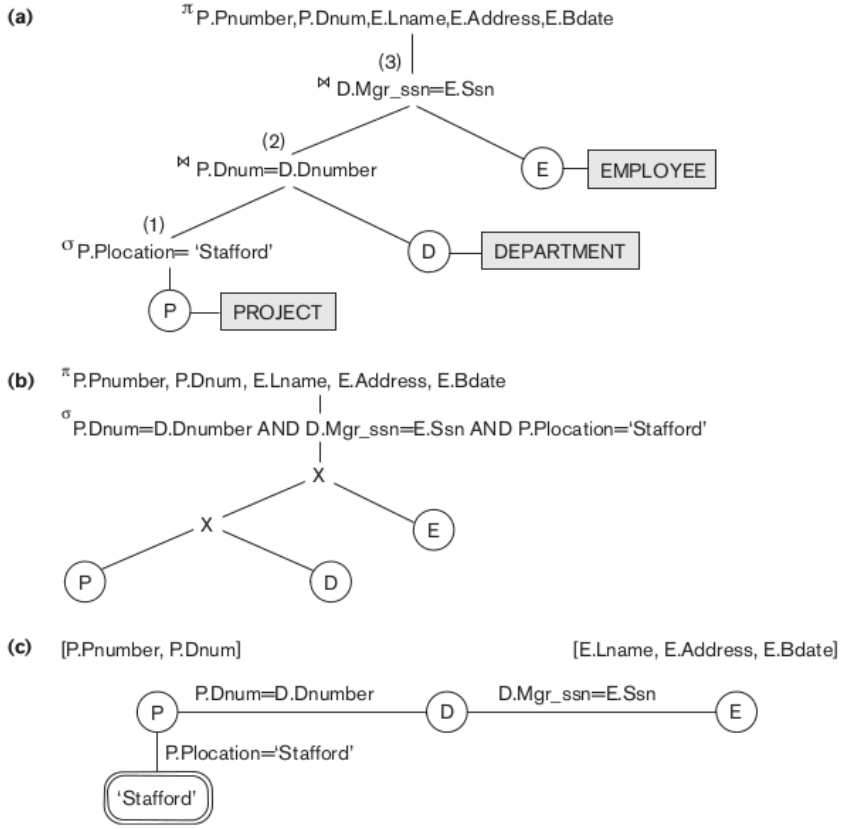


Figure 19.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Optimizing Them

In actuality the initial translation is simplistic:

From	×
Where	σ
Select	π

Then optimizations transform the query tree into something more efficient based on reordering and transforming the extended relational operations. After that the tree is converted to a *query execution plan* that chooses the best algorithms to implement the portions of the tree. $\pi_{Pnumber, Dnum, Lname, Address, Bdate}(\sigma_{P.Dnum=D.Dnumber \wedge D.Mgr\_ssn=E.Ssn \wedge P.Plocation='Stafford'}((Project \times Department) \times Employee))$

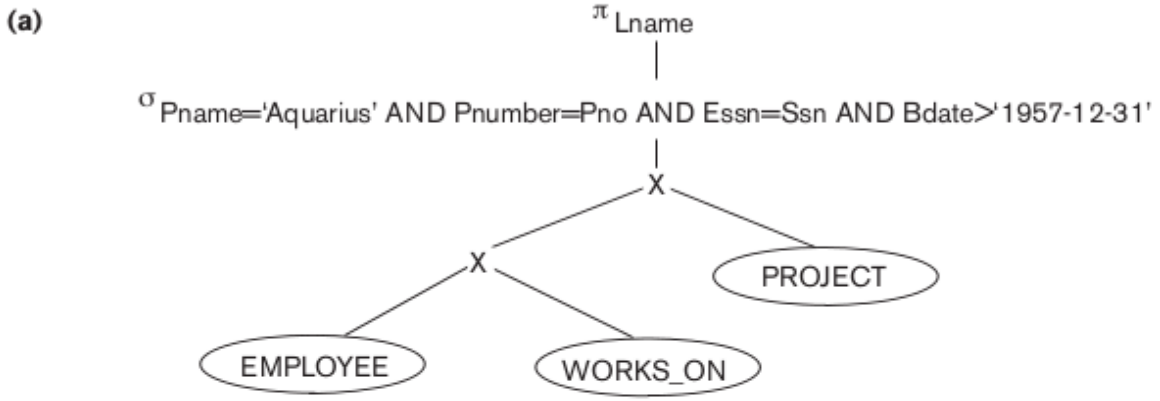
---

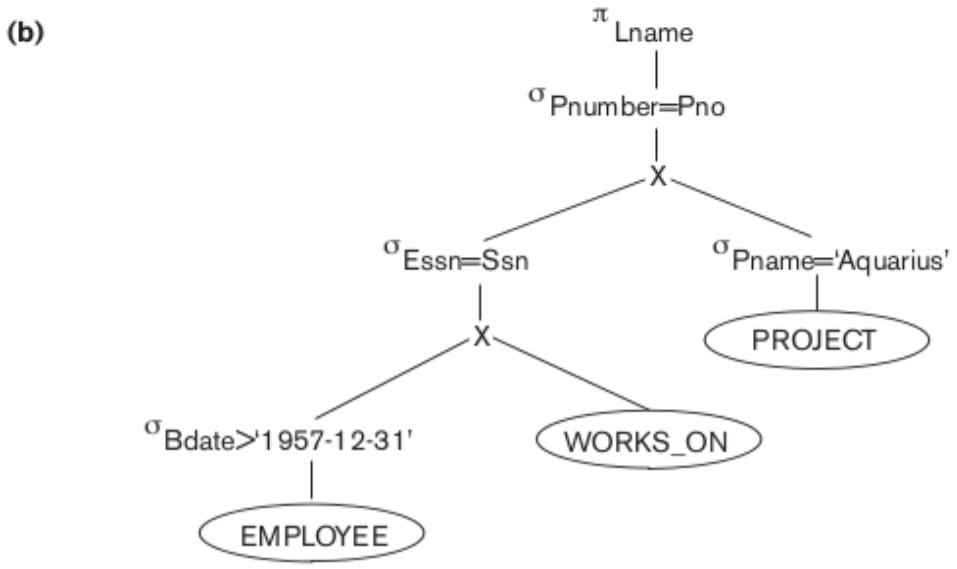
1. Find the last names of employees born after 1957 who work on a project named 'Aquarius'.

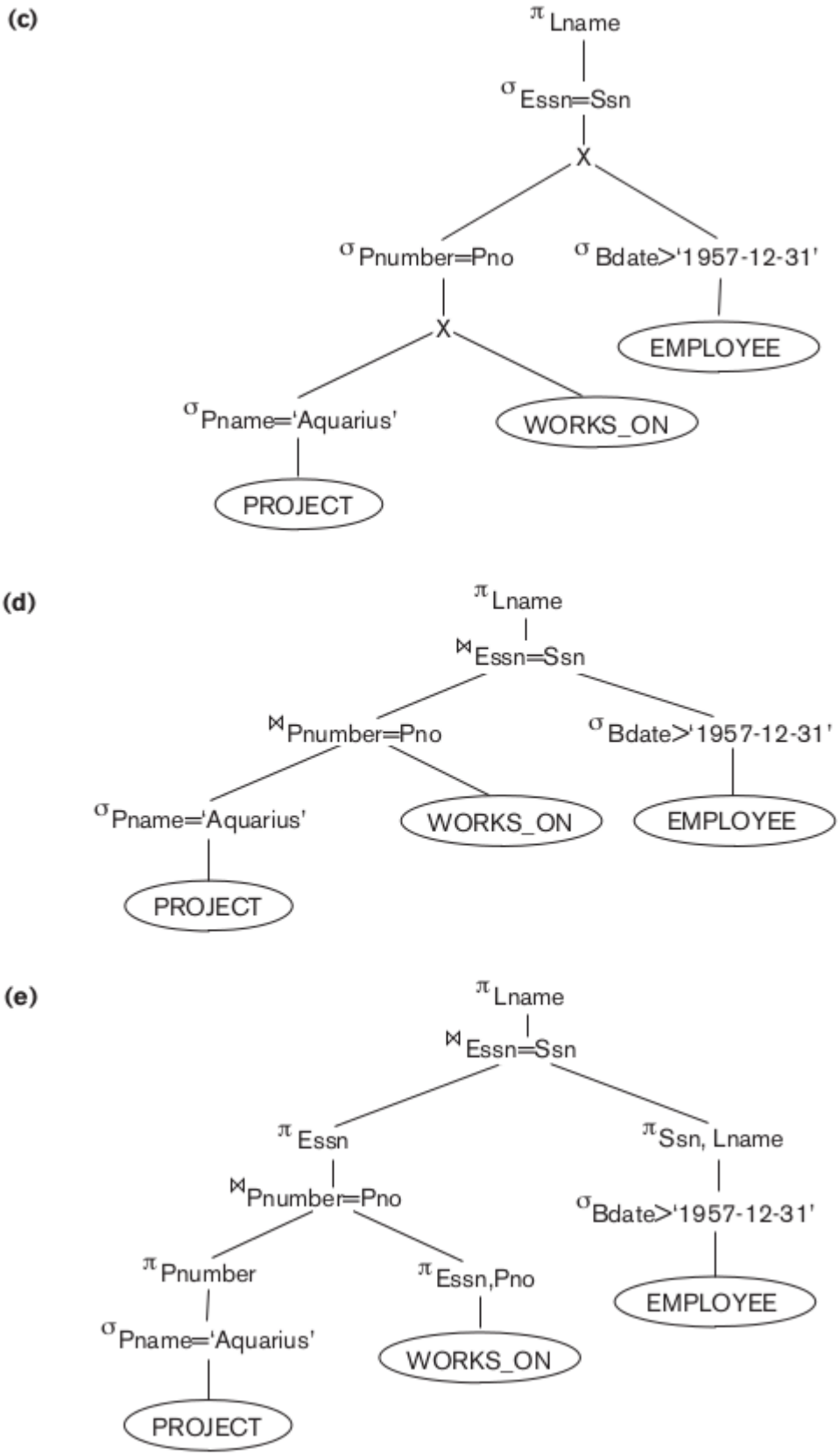
```
2. select  Lname
3. from    Employee, Works_On, Project
```

```
4.  where   Pname='Aquarius' and Pnumber=Pno and Essn=Ssn
5.          and Bdate > '1957-12-31';
```

Figure 19.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.







General Transformation Rules

Generally, perform σ and π (which tend to reduce the size of the intermediate tables) *before* any &bowie; operations (which tend to multiply their sizes). .

1	Cascade σ	$\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge cn} \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots\sigma_{cn}(R)))$	
2	Commutativity of σ	$\sigma_a(\sigma_b(R)) \equiv \sigma_b(\sigma_a(R))$	
3	Cascade π	$\pi_a(\pi_b(\ldots(R))) \equiv \pi_a(R)$	
4	Commute σ with π	$\pi_{a,b,\ldots}(\sigma_c(R)) \equiv \sigma_c(\pi_{a,b,\ldots}(R))$	Only if c only depends on the attributes of π
5	Commutativity of ⋈	$R \bowtie S \equiv S \bowtie R$	Also true of ×
6	Commuting σ with ⋈	$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$	Only if c involves only the attributes of R. Also applies to ×.
6a	Commuting σ with ⋈, conjunction version	$\sigma_{c_1 \wedge c_2}(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$	Where c_1 and c_2 involve the attributes of R and S, respectively.
7	Commuting π with ⋈	$\pi_L (R \bowtie_c S) \equiv (\pi_{A_1,\ldots,An}(R)) \bowtie_c (\pi_{B_1,\ldots,Bm}(S))$	So long as the projection attributes L = $\{A_1,\ldots,A_n,B_1,\ldots,B_m\}$ where $A \in Attr(R)$, $B \in Attr(S)$, and the join condition c only dependes on attributes in L.
7a	Commuting π with ⋈, extra join attributes version	$\pi_L (R \bowtie_c S) \equiv \pi_L [(\pi_{A_1,\ldots,An,\ldots,An+k}(R)) \bowtie_c (\pi_{B_1,\ldots,Bm,\ldots,Bm+p}(S))]$	Here the projection attributes in L are the same, but the join condition c depends on additional attributes from R and S. These attributes are shown as $A_{n+1}\ldots A_{n+k}$ and $B_{m+1}\ldots B_{m+p}$. The original π_L must be wrapped around the right-side expression to remove those extra join atttributes one they're unneeded.
8	Commutativity of set operations		∩ and ∪ are commutative but – is not
9	Associativity of ⋈, ×, ∪, and ∩	$(R \times S) \times T \equiv R \times (S \times T)$	Also true for each of the other three

10	Commute σ with set operations	$\sigma_c (R \theta S) \equiv (\sigma_c(R)) \theta (\sigma_c(S))$	Where θ is any of the set operations ∪, ∩, –
11	Commute π with ∪	$\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$	
12	Convert a (σ,×) sequence into ⋈	$\sigma_c(R \times S) \equiv R \bowtie_c S$	

DeMorgan's laws:

- $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$
- $\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$

General approach:

- Start with the canonical tree.
- Move σs as low as possible.
- Merge operations when possible (×,σ → ⋈ is classic).
- Move πs down, but not probably not below σs.

Example query optimization

```
select Fname, Lname, Address
from   Employee as E, Department as D
where  D.Super_ssn=E.Ssn and D.Dname like 'Plan%';
```

Converting Query Trees to Execution Plans

(Up to algorithms sections)

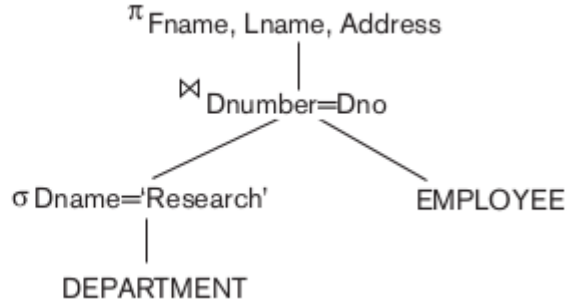


Figure 19.6
A query tree for query Q1.

- If there is an index on Department.Dname, use it for an index search to implement the $\sigma_{Dname='Research'}$.
- If there is an index on Employee.Dno, implement the $\bowtie_{Dnumber=Dno}$ by a J2 (Single-loop join).
- Lastly a simple π over those results.

- Ideally all these algorithms would be pipelined as a single step without writing intermediate files.

Using Selectivity and Cost Estimates in Query Optimization

The DBMS attempts to form a good *cost model* of various query operations as applied to the current database state, including the attribute value statistics (histogram), nature of indices, number of block buffers that can be allocated to various pipelines, selectivity of selection clauses, storage speed, network speed (for distributed databases in particular), and so on.

Access cost to secondary storage
: Reading and writing blocks between storage and ram. (time)

Disk storage cost
: Cost of temporary intermediate files in storage. (time/space)

Computation cost
: Usually slower than storage, but sometimes not, the cpu cost of evaluating the query. (time)

Memory usage cost
: The amount of ram needed by the query. (space)

Communication cost
: The cost to transport query data over a network between database nodes. (time)

"Typical" databases emphasize access cost, the usual limiting factor. In-memory databases minimize computation cost, while distributed databases put increasing emphasis on communication cost.

Catalog Information Used in Cost Functions

- Basic file data: number of records (r), record size (R), number of blocks (b).
- Primary file organization (heap file, ordered, ordered with index, hashed, overflow?)
- Other indices, if present.
- Number of levels (x) for multilevel indices, top-level block count (b_{I1}).
- Number of distinct values (d) of an attribute. If the values are uniformly distributed, the *selectivity* (sl) is ($^1/_d$), and the selection cardinality ($s = sl \cdot r$). If the attribute values are *skewed* this isn't a good estimate, and a more detailed histogram of sl is appropriate.

Examples of Cost Functions for Select

S1. linear search
: On average half the blocks must be accessed for an equality condition on the key, all the blocks otherwise.

S2. binary search
: lg b for the search, plus more if it's nonkey.

S3a. primary index for a single record
: One more than the number of index levels.

S3b. hash index for a single record
: Average of 1 or 2 depending on the type of hash.

S4,.ordered index for multiple records
: $C_{S4}=x+{}^b/_2$ as a rough estimate.

S5. clustering index for multiple records

x for the index search to get the cluster block, then ⌈s/bfr⌉ file blocks pointed to by the cluster.

S6. B^+-Tree

Tree height + 1 if its key, that plus the selectivity (s) if not.

S7. conjunctive selection

The sum of the costs of the subconditions, if the set intersection fits in memory.

S8,.conjunctive selection with composite index

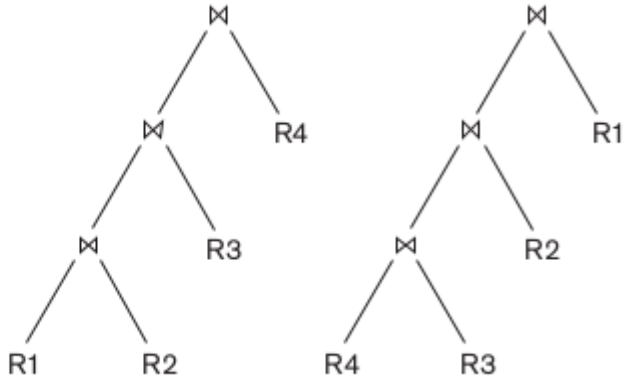
The same as above for whichever type of index.

Examples of Cost Functions for Join

A *join selectivity* (js) is the ratio of the number of tuples produced by a join to the number of tuples produced by a cross product of the underlying relations, or js = $|(R \bowtie_c S)| / |(R \times S)| = |(R \bowtie_c S)| / (|R| \cdot |S|)$.

Multiple Relation Queries and Join Ordering

Figure 19.7
Two left-deep (JOIN) query trees.



Unit-IV

Recovery Techniques

Database Recovery Techniques - Database Security – Debate on the distributed databases and Client- Server Architecture with reference to Indian Railway Reservation System.

a. Database Recovery

1 Purpose of Database Recovery

To bring the database into the last consistent state, which existed prior

to the failure.

To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example:

If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts

2. Types of Failure

The database may become unavailable for use due to

Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.

System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.

Media failure: Disk head crash, power disruption, etc.

Transaction Log

For recovery from any type of failure data values prior to modification (BFIM

BeFore Image) and the new value after modification (AFIM – AFter Image) are required.

These values and other information is stored in a sequential file called Transaction log.

Data Update

Immediate Update: As soon as a data item is modified in cache, the disk copy is updated.

Deferred Update: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

Shadow update: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

In-place update: The disk version of the data item is overwritten by the cache version.

Data Caching

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.

The flushing is controlled by **Modified** and **Pin-Unpin** bits.

Pin-Unpin: Instructs the operating system not to flush the data item.

Modified: Indicates the AFIM of the data item.'

Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**

To maintain atomicity, a transaction's operations are redone or undone.

Undo: Restore all BFIMs on to disk (Remove all AFIMs).

Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

(b)

	A	B	C	D
	30	15	40	20
[start_transaction,T_3]				
[read_item,T_3,C]				

Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules).

Roll-back: One execution of T1, T2 and T3 as recorded in the log

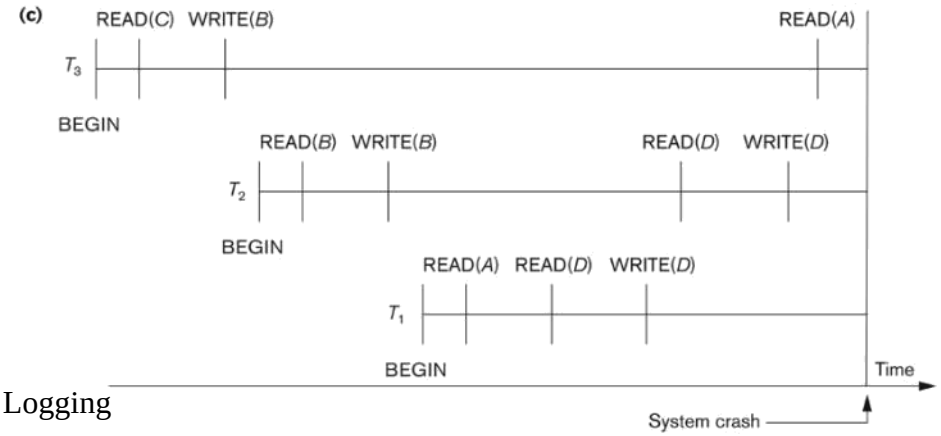


Figure 19.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

Write-Ahead Logging

When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that

- **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

- **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Checkpointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

Suspend execution of transactions temporarily.

Force writes modified buffer data to disk.

Write a [checkpoint] record to the log, save the log to disk.

Resume normal transaction execution.

During recovery redo or undo is required to transactions appearing

after [checkpoint] record.

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:

Steal: Cache can be flushed before transaction commits.

No-Steal: Cache cannot be flushed before transaction commit.

Force: Cache is immediately flushed (forced) to disk.

No-Force: Cache is deferred until transaction commits

These give rise to four different ways for handling recovery:

Steal/No-Force (Undo/Redo)

Steal/Force (Undo/No-redo)

No-Steal/No-Force (Redo/No-undo)

No-Steal/Force (No-undo/No-redo)

8 **. Recovery Scheme**

a. Deferred Update (No Undo/Redo)

The data update goes as follows:

A set of transactions records their updates in the log.

At commit point under WAL scheme these updates are saved on database disk.

After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

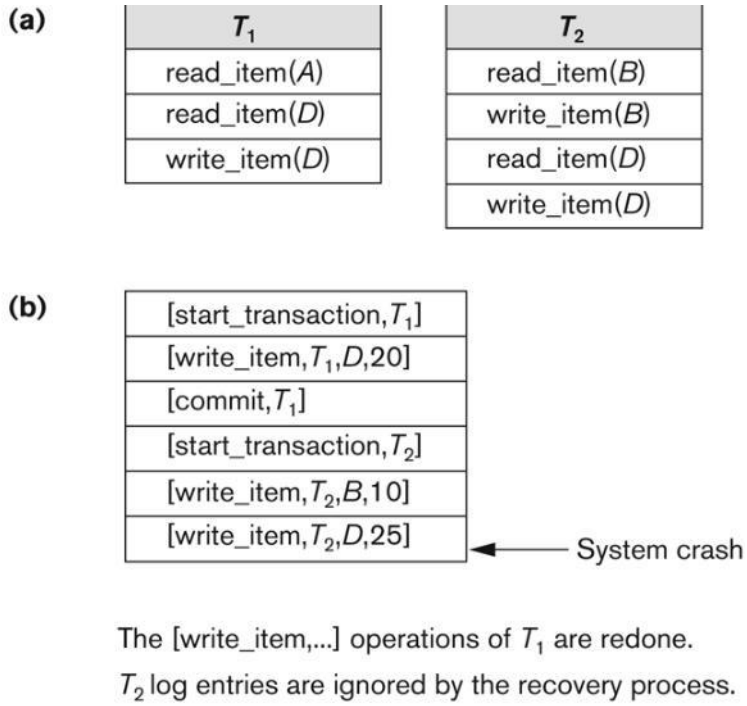


Figure 19.2
An example of recovery using deferred update in a single-user environment. (a) The READ and WRITE operations of two transactions. (b) The system log at the point of crash.

b. Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs

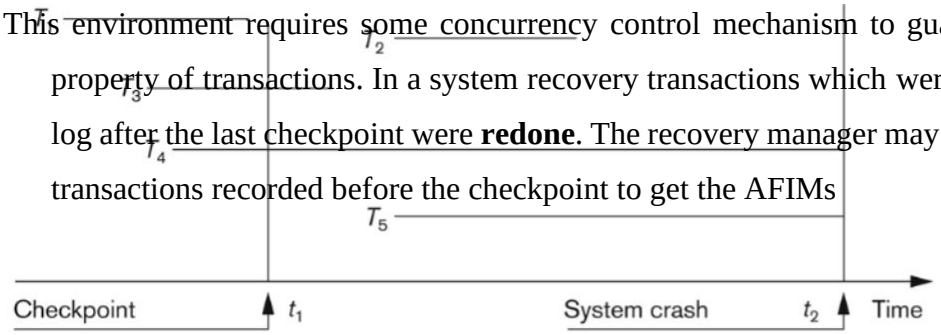


Figure 19.3
An example of recovery in a multi-user environment

(a)

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(b)

[start_transaction,T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item,T_2, D, 25] ← System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Figure 19.4
An example of recovery using deferred update with concurrent transactions.
(a) The READ and WRITE operations of four transactions. (b) System log at the point of crash.

Deferred Update with concurrent users

Two tables are required for implementing this protocol:

Active table: All active transactions are entered in this table.

Commit table: Transactions to be committed are entered in this table.

During recovery, all transactions of the **commit** table is redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is

"**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

c. Recovery Techniques Based on Immediate Update

Undo/No-redo Algorithm

In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits. For this reason the recovery manager **undoes** all transactions during recovery.No transaction is **redone**. It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Undo/Redo Algorithm (**Single-user** environment)

Recovery schemes of this category apply **undo** and also **redo** for recovery. In a single-user environment no concurrency control is required but a log is maintained under WAL.Note

that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.

The recovery manager performs:

Undo of a transaction if it is in the **active** table.

Redo of a transaction if it is in the **commit** table.

- **Undo/Redo Algorithm** (**Concurrent** execution)
- Recovery schemes of this category applies **undo** and also **redo** to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager check pointing is used.
- The recovery performs:
  - **Undo** of a transaction if it is in the **active** table.
  - **Redo** of a transaction if it is in the **commit** table.

d.Shadow Paging

The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

X and Y: Shadow copies of data items

X' and Y': Current copies of data items

To manage access of data items by concurrent transactions two directories (current and shadow) are used.

The directory arrangement is illustrated below. Here a page is a data item.

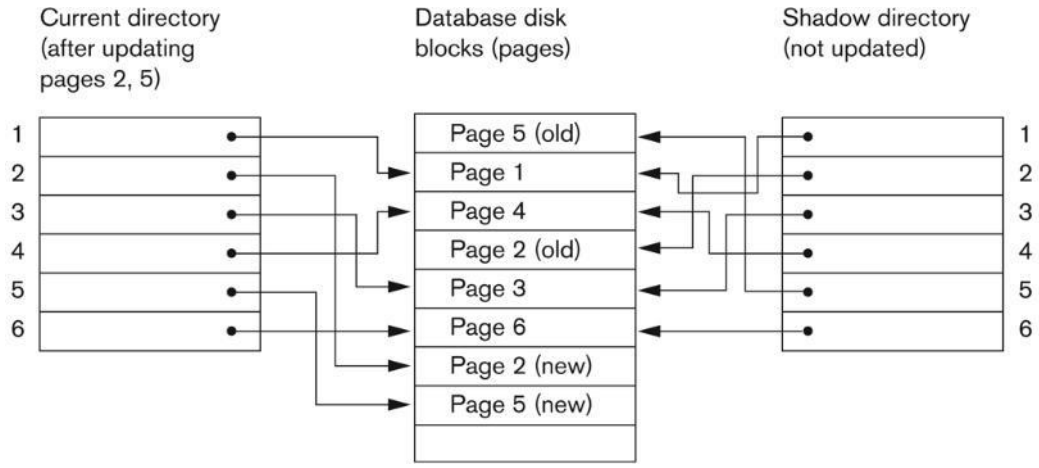


Figure 19.5
An example of shadow paging.

e. The ARIES Recovery Algorithm

The ARIES Recovery Algorithm is based on:

- **WAL** (Write Ahead Logging)

- **Repeating history during redo**:

  - ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.

- **Logging changes during undo**:

  - It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

The ARIES recovery algorithm consists of three steps:

Analysis: step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash. The appropriate point in the log where redo is to start is also determined.

Redo: necessary redo operations are applied.

Undo: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

The Log and Log Sequence Number (LSN)

A log record is written for:

data update

transaction commit

transaction abort

undo

transaction end

In the case of undo a compensating log record is written.

The Log and Log Sequence Number (LSN) (contd.)

A unique LSN is associated with every log record.

LSN increases monotonically and indicates the disk address of the log record it is associated with.

In addition, each data page stores the LSN of the latest log

record corresponding to a change for that page.

A log record stores

(a) the previous LSN of that transaction

(b) the transaction ID

A log record stores:

Previous LSN of that transaction: It links the log record of each transaction. It is like a back pointer points to the previous record of the same transaction

Transaction ID

Type of log record

For a write operation the following additional information is logged:

- Page ID for the page that includes the item
- Length of the updated item
- Its offset from the beginning of the page
- BFIM of the item
- AFIM of the item

The **Transaction table** and the **Dirty Page table**

For efficient recovery following tables are also stored in the log during check pointing:

- **Transaction table**: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.

Dirty Page table: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

e. Checkpointing

A checkpointing does the following:

Writes a begin checkpoint record in the log Writes an end checkpoint record in the log. With this record the contents of transaction table and dirty page table are appended to the end of the log.Writes the LSN of the begin_checkpoint record to a special file. This special file is accessed during recovery to locate the last checkpoint information.

To reduce the cost of checkpointing and allow the system to continue to execute transactions, ARIES uses "fuzzy checkpointing".

The following steps are performed for recovery

Analysis phase: Start at the begin_checkpoint record and proceed to the end_checkpoint record. Access transaction table and dirty page table are appended to the end of the log. Note that during this phase some other log records may be written to the log and transaction table may be modified. The analysis phase compiles the set of redo and undoes to be performed and ends.

Redo phase: Starts from the point in the log up to where all dirty pages have been flushed, and move forward to the end of the log. Any change that appears in the dirty page table is redone.

Undo phase: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a compensating record in the log. The recovery completes at the end of undo phase.

(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	T_1	update	C	...
2	0	T_2	update	B	...
3	1	T_1	commit		...
4	begin checkpoint				
5	end checkpoint				
6	0	T_3	update	A	...
7	2	T_2	update	C	...
8	7	T_2	commit		...

(b)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	2	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2

(c)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2
A	6

Figure 19.6
An example of recovery in ARIES. (a) The log at point of crash. (b) The Transaction and Dirty Page Tables at time of checkpoint. (c) The Transaction and Dirty Page Tables after the analysis phase.

f. Recovery in multidatabase system

A multi database system is a special distributed database system where one node may be running relational database system under UNIX, another may be running object-oriented system under Windows and so on.

A transaction may run in a distributed fashion at multiple nodes. In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.

This commit scheme is referred to as "**two-phase commit**" (**2PC**). If any one of these nodes fails or cannot commit the part of the transaction,

then the transaction is aborted.Each node recovers the transaction under its own recovery protocol.

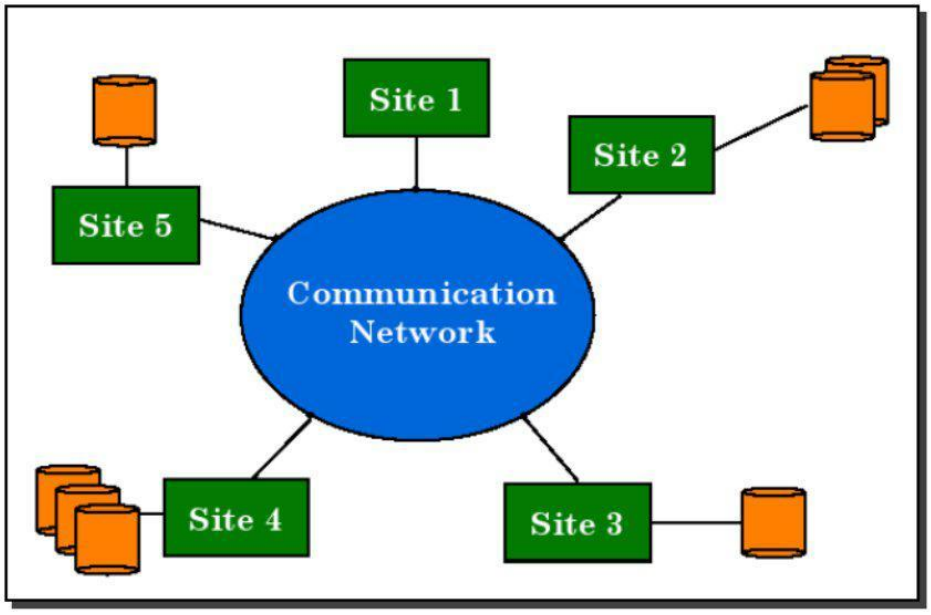
Distributed Database:

Distributed databases bring the advantages of distributed computing to the database management domain. A distributed computing system consists of a number of processing elements, not necessarily homogeneous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks.

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network • A distributed database management system (DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

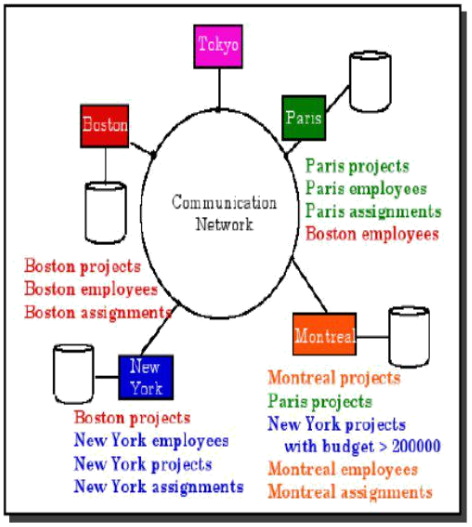
Data stored at a number of sites each site logically consists of a single processor at different sites are interconnected by a computer network (we do not consider multiprocessors in DDBMS, cf. parallel systems).

DDBS is a database, not a collection of files (cf. relational data model).Placement and query of data is impacted by the access patterns of the user DDBMS is a collections of DBMSs (not a remote file system).



Distributed System

- **Example:** Database consists of 3 relations `employees`, `projects`, and `assignment` which are partitioned and stored at different sites (fragmentation).



Distributed Databases

Applications of Distributed Databases

- Manufacturing, especially multi-plant manufacturing
- Military command and control
- Airlines
- Hotel chains
- Any organization which has a decentralized organization structure

Advantages of Distributed Databases

a)**Management of distributed data with different levels of transparency:**

Distribution or network transparency: This refers to freedom for the user from the operational details of the network. It may be divided into location transparency and naming transparency. **Location transparency** refers to the fact that the command used to perform a task is independent of the location of data and the location of the system where the command was issued. **Naming transparency** implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.

b)**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.

c) Fragmentation transparency:

Two types of fragmentation are possible.

1)**Horizontal fragmentation** distributes a relation into sets of tuples (rows).

2)**Vertical fragmentation** distributes a relation into subrelations where each subrelation is defined by a subset of the columns of the original relation. A global query by the user must be transformed into several fragment queries. Fragmentation transparency makes the user unaware of the existence of fragments.

d) Increased reliability and availability:

These are two of the most common potential advantages cited for distributed databases. Reliability is broadly defined as the probability that a system is running (not down) at a certain time point, whereas availability is the probability that the 1 Page 658 of 893 system is continuously available during a time interval. When the data and DBMS software are distributed over several sites, one site may fail while other sites continue to operate. Only the data and software that exist at the failed site cannot be accessed. This improves both reliability and availability.

e)Improved performance:

f.)**Easier expansion:** In a distributed environment, expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier.

Data Fragmentation

Split a relation into logically related and correct parts. A relation can be fragmented in two ways:

Horizontal Fragmentation

Vertical Fragmentation

Horizontal fragmentation:

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

Vertical fragmentation

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

Fragmentation Representation

Horizontal fragmentation

1. Each horizontal fragment on a relation can be specified by a sCi

(R) operation in the relational algebra.

2. Complete horizontal fragmentation

3. A set of horizontal fragments whose conditions C1, C2, …, Cn include all the tuples in R- that is, every tuple in R satisfies (C1 OR C2 OR … OR Cn).

4. Disjoint complete horizontal fragmentation: No tuple in R satisfies (Ci AND Cj) where i ≠ j.

5. To reconstruct R from horizontal fragments a UNION is applied.

Vertical fragmentation

1. A vertical fragment on a relation can be specified by a PLi(R) operation in the relational algebra.

2. Complete vertical fragmentation

3. A set of vertical fragments whose projection lists L1, L2, …, Ln include all the attributes in R but share only the primary key of R. In this case the projection lists satisfy the following two conditions:

4. L1 » L2 » ... » Ln = ATTRS (R)

5. Li « Lj = PK(R) for any i j, where ATTRS (R) is the set of attributes of R and PK(R) is the primary key of R.

6. To reconstruct R from complete vertical fragments a OUTER UNION is applied.

Mixed (Hybrid) fragmentation

1. A combination of Vertical fragmentation and Horizontal fragmentation.
2. This is achieved by SELECT-PROJECT operations which is represented by PLi(sCi (R)).
3. If C = True (Select all tuples) and L ≠ ATTRS(R), we get a vertical fragment, and if C ≠ True and L ≠ ATTRS(R), we get a mixed fragment.
4. If C = True and L = ATTRS(R), then R can be considered a fragment

Fragmentation schema

A definition of a set of fragments (horizontal or vertical or horizontal and vertical) that includes all attributes and tuples in the database that satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of UNION (or OUTER JOIN) and UNION operations.

Allocation schema

It describes the distribution of fragments to sites of distributed databases. It can be fully or partially replicated or can be partitioned

Data Replication

Database is replicated to all sites. In full replication the entire database is replicated and in partial replication some selected part is replicated to some of the sites.

Data replication is achieved through a replication schema.

Data Distribution (Data Allocation)

This is relevant only in the case of partial replication or partition.

The selected portion of the database is distributed to the database sites.

Types of Distributed Database Systems

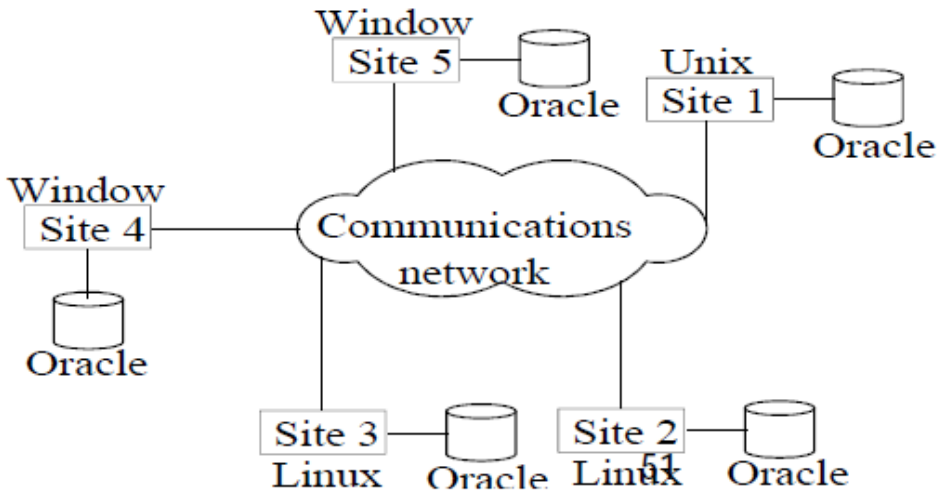
Homogeneous

1)All sites of the database system have identical setup, i.e., same database system software.

2)The underlying operating system may be different.

For example, all sites run Oracle or DB2, or Sybase or some other database system.

The underlying operating systems can be a mixture of Linux, Window, Unix, etc.



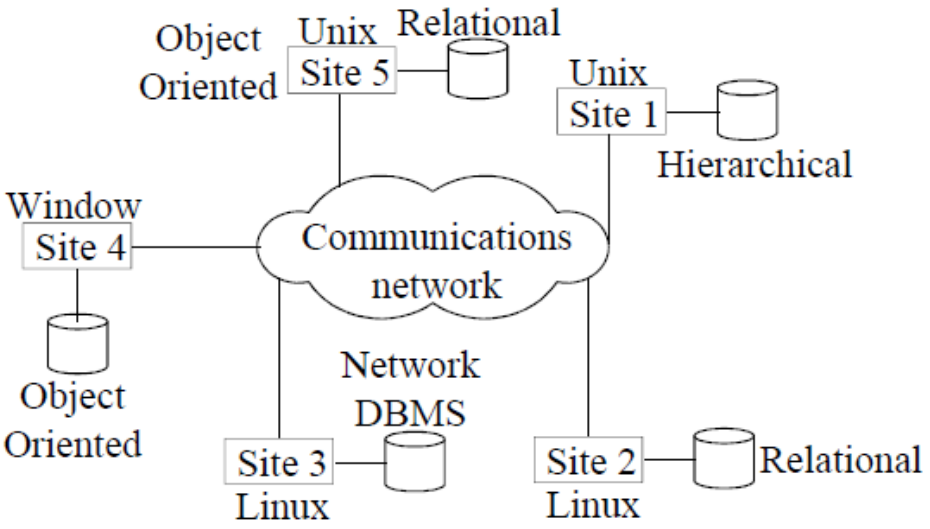
Homogeneous System

Heterogeneous

1) Federated: Each site may run different database system but the data access is managed through a single conceptual schema.

This implies that the degree of local autonomy is minimum. Each site must adhere to a centralized access policy. There may be a global schema.

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



Heterogeneous System

Query processing in Distributed Databases:

Issues:

1) Cost of transferring data (files and results) over the network. This cost is usually high so some optimization is necessary. Example relations: Employee at site 1 and Department at Site 2

Example: Employee at site 1. 10,000 rows. Row size = 100 bytes. Table size = 106 bytes.

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Superssn	Dno

Department at Site 2. 100 rows. Row size = 35 bytes. Table size = 3,500 bytes.

For each employee, retrieve employee name and department name Where the employee works.

A: PFname, Lname, Dname (Employee Dno = Dnumber Department)

The result of this query will have 10,000 tuples, assuming that every employee is related to a department. Suppose each result tuple is 40 bytes long. The query is submitted at site 3 and the result is sent to this site.

Problem: Employee and Department relations are not present at site 3.

Strategies:

1. Transfer Employee and Department to site 3. Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3 Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
3. Transfer Department relation to site 1, executes the join at site 1, and sends the result to site 3.

Total bytes transferred = 400,000 + 3500 = 403,500 bytes. Optimization criteria: minimizing data transfer.

Preferred approach: strategy 3.

Concurrency Control and Recovery

Distributed Databases encounter a number of concurrency control and recovery problems which are not present in centralized databases. Some of them are listed below.

1) Dealing with multiple copies of data items

The concurrency control must maintain global consistency. Likewise the recovery mechanism must recover all copies and maintain consistency after recovery.

2) Failure of individual sites

Database availability must not be affected due to the failure of one or two sites and the recovery scheme must recover them before they are available for use

3) Communication link failure

This failure may create network partition which would affect database availability even though all database sites may be running

4) Distributed commit

A transaction may be fragmented and they may be executed by a number of sites. This require a two or three-phase commit approach for transaction commit

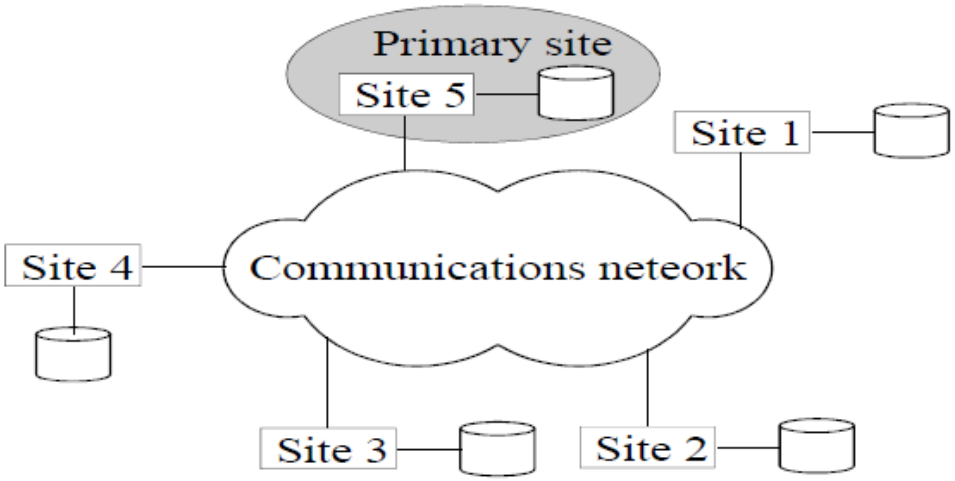
1) Distributed deadlock

Since transactions are processed at multiple sites, two or more sites may get involved in deadlock. This must be resolved in a distributed manner.

Concurrency Control and Recovery

Distributed Concurrency control based on a distributed copy of a data item

Primary site technique: A single site is designated as a primary site which serves as a coordinator for transaction management



Primary site technique

Transaction management:

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

Primary Copy Technique:

In this approach, instead of a site, a data item partition is designated as primary copy. To lock a data item just the primary copy of the data item is locked.

Advantages:

Since primary copies are distributed at various sites, a single site is not overloaded with locking and unlocking requests.

Disadvantages:

Identification of a primary copy is complex. A distributed directory must be maintained, possibly at all sites.

Recovery from a coordinator failure

In both approaches a coordinator site or copy may become unavailable. This will require the selection of a new coordinator.

Primary site approach with no backup site

Aborts and restarts all active transactions at all sites. Elects a new coordinator and initiates transaction processing.

Primary site approach with backup site:

Suspends all active transactions, designates the backup site as the primary site and identifies a new back up site. Primary site receives all transaction management information to resume processing.

Primary and backup sites fail or no backup site:

Use election process to select a new coordinator site.

Concurrency control based on voting:

There is no primary copy of coordinator.
Send lock request to sites that have data item.

If majority of sites grant lock then the requesting transaction gets the data item.
Locking information (grant or denied) is sent to all these sites.

To avoid unacceptably long wait, a time-out period is defined. If the requesting transaction does not get any vote information then the transaction is aborted.

Architectural Models

Some of the common architectural models are −

- Client - Server Architecture for DDBMS
- Peer - to - Peer Architecture for DDBMS
- Multi - DBMS Architecture

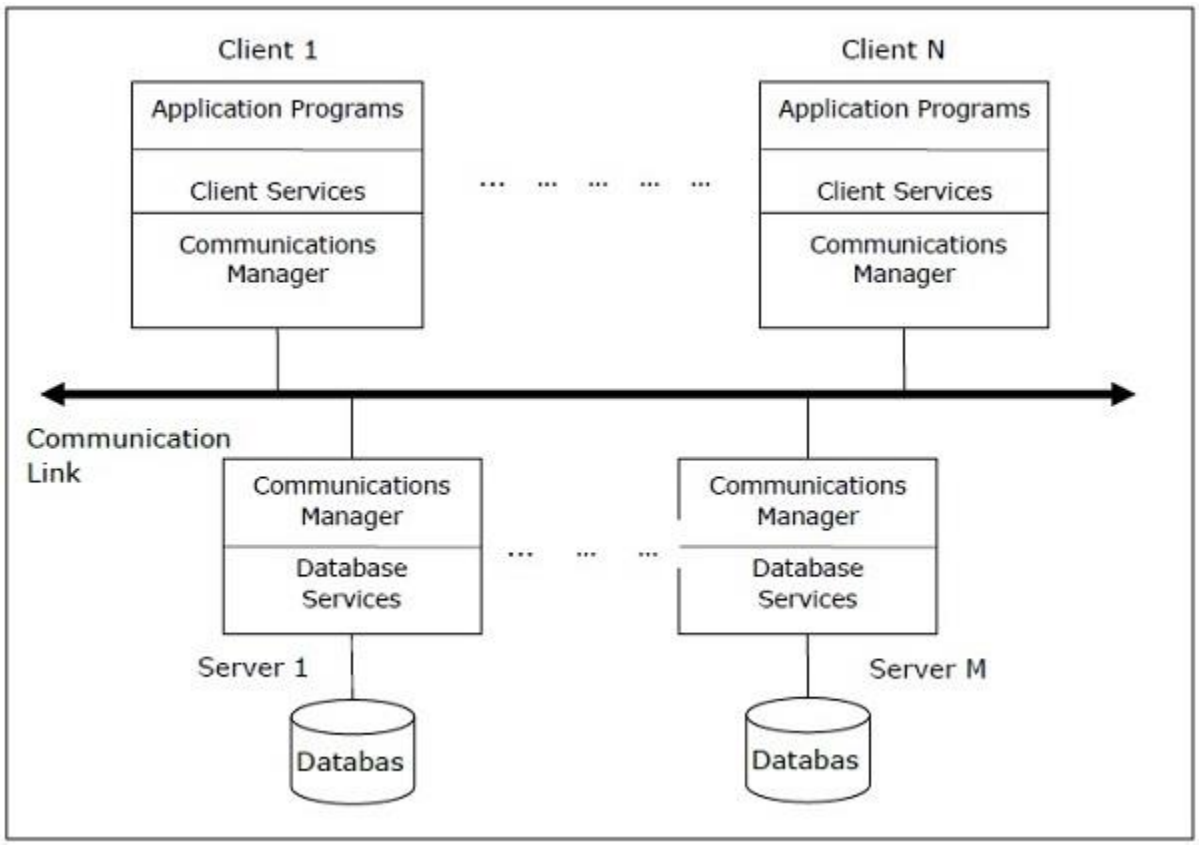
Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client

functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The two different clients - server architecture are −

- Single Server Multiple Client
- Multiple Server Multiple Client (shown in the following diagram)

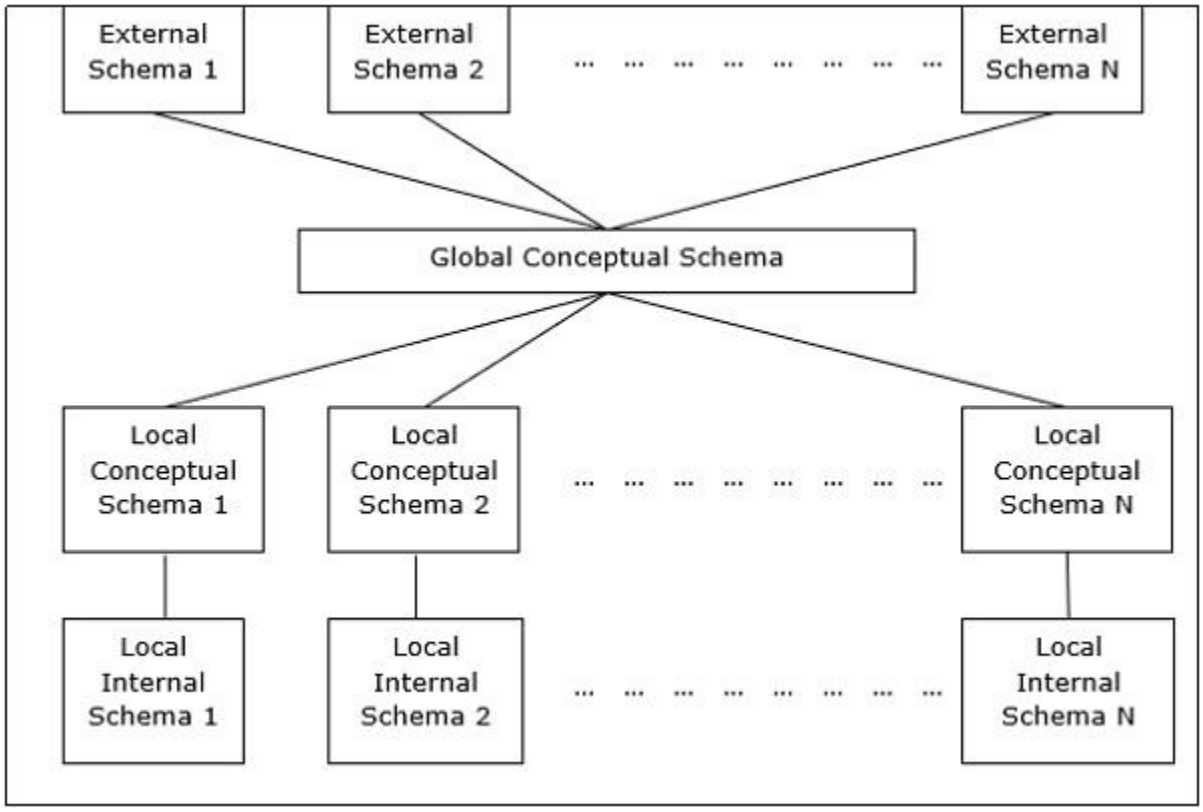


Peer- to-Peer Architecture for DDBMS

In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas −

- **Global Conceptual Schema** − Depicts the global logical view of data.
- **Local Conceptual Schema** − Depicts logical data organization at each site.
- **Local Internal Schema** − Depicts physical data organization at each site.
- **External Schema** − Depicts user view of data.



Multi - DBMS Architectures

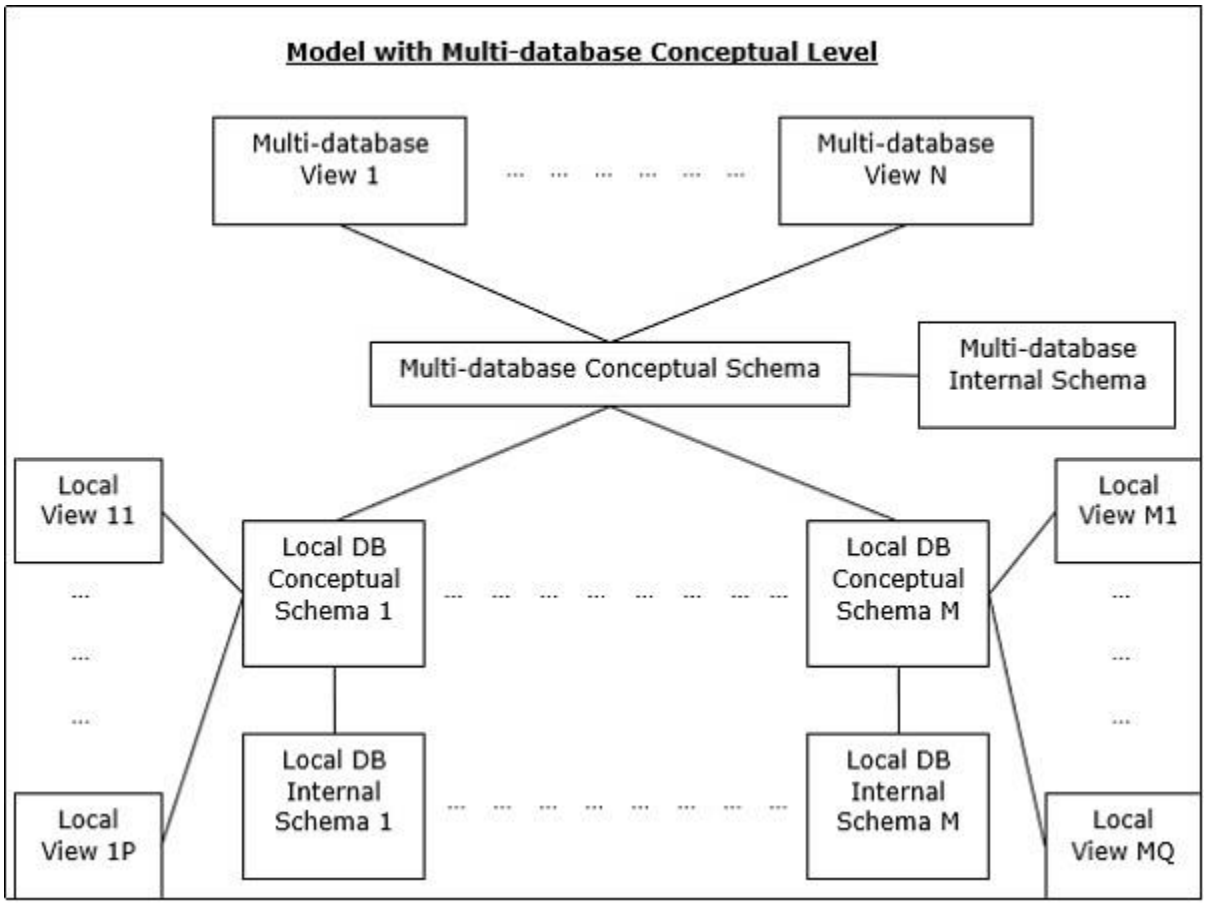
This is an integrated database system formed by a collection of two or more autonomous database systems.

Multi-DBMS can be expressed through six levels of schemas −

- **Multi-database View Level** − Depicts multiple user views comprising of subsets of the integrated distributed database.
- **Multi-database Conceptual Level** − Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
- **Multi-database Internal Level** − Depicts the data distribution across different sites and multi-database to local data mapping.
- **Local database View Level** − Depicts public view of local data.
- **Local database Conceptual Level** − Depicts local data organization at each site.
- **Local database Internal Level** − Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS −

- Model with multi-database conceptual level.
- Model without multi-database conceptual level.



Database Security

Types of Security

- Legal and ethical issues
- Policy issues
- System-related issues
- The need to identify multiple security levels

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.

Two types of database security mechanisms:

Discretionary security mechanisms

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking **privileges**.

Mandatory security mechanisms

In many applications, and *additional security policy* is needed that classifies data and users based on security classes. This approach as **mandatory access control**, would typically be *combined* with the discretionary access control mechanisms.

Security Issues in Databases

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole; this function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.
- The security problem associated with databases is that of controlling the access to a **statistical database**, which is used to provide statistical information or summaries of values based on various criteria.
- The counter measures to **statistical database security** problem is called **inference control** measures.
- Another security is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
- Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**.
- A final security issue is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type communication network.
- The data is **encoded** using some coding algorithm. An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

Database Security and the DBA

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

accordance with the policy of the organization. The DBA has a **DBA account** in the DBMS, sometimes called a **system** or **superuser account**, which provides powerful capabilities :

1. *Account creation*

2. *Privilege granting*

3. *Privilege revocation*

4. *Security level assignment*

The DBA is responsible for the overall security of the database system.

Action 1 is access control, whereas 2 and 3 are discretionary and 4 is used to control mandatory authorization.

Whenever a person or group of person s need to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database.

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

A database log that is used mainly for security purposes is sometimes called an **audit trail**.

Types of Discretionary Privileges

The *account level*: At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.

The privileges at the **account level** apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege,

To apply schema changes such adding or removing attributes from relations; the DROP privilege, to delete relations or views; the MODIFY privilege, to insert, delete, or update tuples; and the SELECT privilege, to retrieve information from the database by using a SELECT query.

The *relation (or table level)*: At this level, the DBA can control the privilege to access each individual relation or view in the database.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an **owner account**, which is typically the account that was used when the relation was created in the first place.

The owner of a relation is given *all* privileges on that relation. In SQL2, the DBA can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the CREATE SCHEMA command.

The owner account holder can pass privileges on any of the owned relation to other users by **granting** privileges to their accounts.

In SQL the following types of privileges can be granted on each individual relation R:

- SELECT (retrieval or read) privilege on R: Gives the account retrieval privilege. In SQL this gives the account the privilege to use the SELECT statement to retrieve tuples from R.
- MODIFY privileges on R: This gives the account the capability to modify tuples of R. In SQL this privilege is further divided into UPDATE, DELETE, and INSERT privileges to apply the corresponding SQL command to R. In addition, both the INSERT and UPDATE privileges can specify that only certain attributes can be updated by the account.

** Note that to create a view, the account must have SELECT privilege on *all relations* involved in the view definition.

The mechanism of **views** is an important discretionary authorization mechanism in its own right.

For example, if the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B. The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access.

Revoking Privileges

For example, the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed. Hence, a mechanism for **revoking** privileges is needed.

In SQL, a REVOKE command is included for the purpose of canceling privileges.

Example of Grant and Revoke Commands in SQL

Suppose that the DBA creates four accounts --A1, A2, A3, and A4-- and wants only A1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

GRANT CREATETAB TO A1;

In SQL2 the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command as follows:

CREATE SCHEMA EXAMPLE AUTHORIZATION A1;

GRANT privilege_name

ON object_name

TO {user_name |PUBLIC |role_name}

[WITH GRANT OPTION];

For eg:

GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION;

GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

REVOKE privilege_name

ON object_name

FROM {user_name |PUBLIC |role_name}

REVOKE SELECT ON EMPLOYEE FROM A3

Privilege for Views

Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.

A1 then create the view:

CREATE VIEW A3EMPLOYEE AS

SELECT NAME, BDATE, ADDRESS

FROM EMPLOYEE

WHERE DNO = 5;

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;

Mandatory Access Control and Role-Based Access Control for Multilevel Security

** The discretionary access control techniques of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems.

This is an all-or-nothing method: A user either has or does not have a certain privilege.

In many applications, and *additional security policy* is needed that classifies data and users based on security classes. This approach as **mandatory access control**, would typically be *combined* with the discretionary access control mechanisms.

Typical **security classes** are

Top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest: $TS \geq S \geq C \geq U$

The commonly used model for multilevel security, known as the Bell-LaPadula model.

Classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications, T, S, C, or U.

Clearance (classification) of a subject S as **class(S)** and to the **classification** of an object O as **class(O)**.

Two restrictions are enforced on data access based on the subject/object classifications:

1. A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the **simple security property**.

2. A subject S is not allowed to write an object O unless class(S) ≤ class(O). This known as the **star property** (or * property).

3. To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects. Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification.

4. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. Hence, a **multilevel relation** schema R with n attributes would be represented as

   $R(A_1,C_1,A_2,C_2, \ldots, A_n,C_n,TC)$

   where each C_i represents the classification attribute associated with attribute A_i.

   The value of the TC attribute in each tuple t – which is the *highest* of all attribute classification values within t – provides a general classification for the tuple itself, whereas each C_i provides a finer security classification for each attribute value within the tuple.

   A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**.

   In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple.

Comparing Discretionary Access Control and Mandatory Access Control

- Discretionary Access Control (DAC) policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains.

- The main drawback of DAC models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs.

- By contrast, mandatory policies ensure a high degree of protection in a way, they prevent any illegal flow of information.

- Mandatory policies have the drawback of being too rigid and they are only applicable in limited environments.

- In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

Role-Based Access Control

Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprisewide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands.

The GRANT and REVOKE commands discussed under DAC can then be used to assign and revoke privileges from roles.

CREATE ROLE role_name [WITH ADMIN {CURRENT_USER | CURRENT_ROLE}]

With the above syntax, a role with role_name is created and immediately assigned to the current user or the currently active role is passed on to other users. The default usage is WITH ADMIN CURRENT_USER.

- RBAC appears to be a viable alternative to traditional discretionary and mandatory access controls; it ensures that only authorized users are given access to certain data or resources.
- Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.

- Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.
- Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as time and duration of role activations, and timed triggering of a role by an activation of another role.
- Using an RBAC model is highly desirable goal for addressing the key security requirements of Web-based applications.

In contrast, discretionary access control (DAC) and mandatory access control (MAC) models lack capabilities needed to support the security requirements emerging enterprises and Web-based applications.

UNIT 5 OBJECT DATABASE

Concepts for Object Database - Enhanced Data Models for Advanced Application
Emerging Database Technologies and Application

CONCEPTS FOR OBJECT DATABASE

OBJECT-ORIENTED DATABASE (OODB)

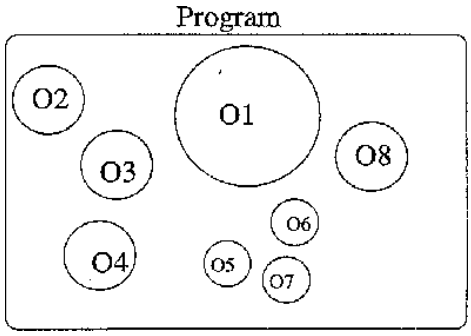
- Object-Oriented Language Features:
  - abstract data types
  - inheritance
  - object identity
- Object-Oriented Database Features:
  - Persistence
  - support of transactions
  - simple querying of bulk data
  - concurrent access
  - resilience
  - security

WHY OBJECT-ORIENTED DATABASE?

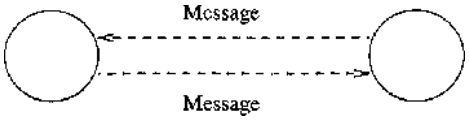
- Industry Trends:
- Integration and Sharing
- Seamless integration of operating systems, databases, languages, spreadsheets, word processors, AI expert system shells.
- Sharing of data, information, software components, products, computing environments.
- Referential sharing:
  - Multiple applications, products, or objects share common sub-

objects. (Hypermedia links are then used to navigate from one object to another)
Object-oriented databases allows referential sharing through the support of object identity and inheritance.

Fundamentals of Object-Oriented Approach

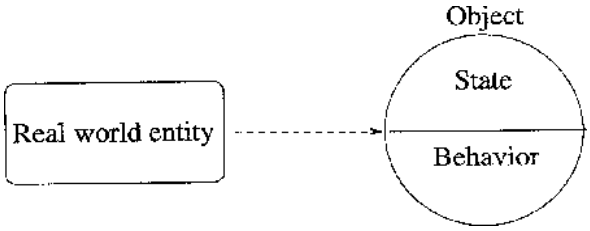


The object-oriented paradigm is illustrated below:



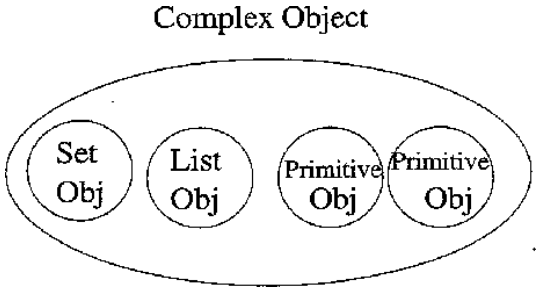
Objects and Identity

The following figure shows object with state and behavior. The state is represented by the values of the object's attributes, and the behavior is defined by the methods acting on the state of the object. There is a unique object identifier OID to identify the object.



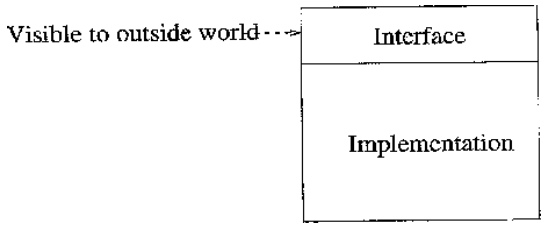
Complex Objects

Complex objects are built by applying constructors to simpler objects including: sets, lists and tuples. An example is illustrated below:



Encapsulation

Encapsulation is derived from the notion of Abstract Data Type (ADT). It is motivated by the need to make a clear distinction between the specification and the implementation of an operation. It reinforces modularity and provides a form of logical data independence.



Class

A class object is an object which acts as a

template. It specifies:

A structure that is the set of attributes of the

instances A set of operations

A set of methods which implement the operations

Instantiation means generating objects, Ex. 'new' operation in

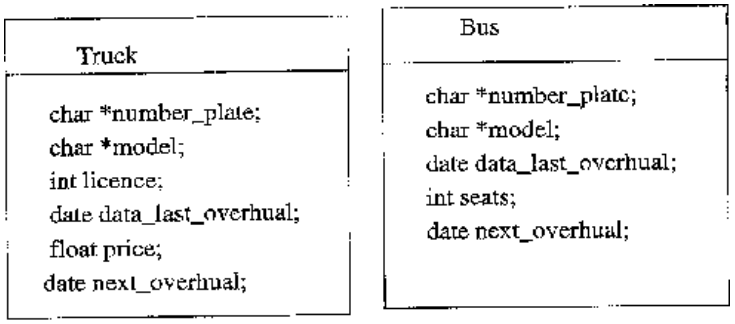
C++ Persistence of objects: Two approaches

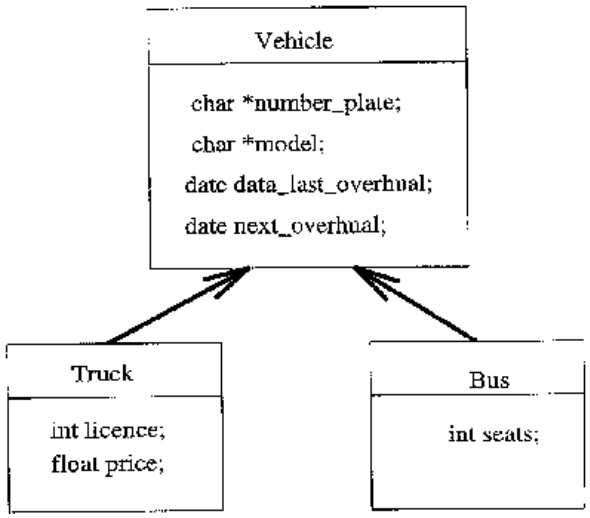
An implicit characteristic of all objects

An orthogonal characteristic - insert the object into a persistent collection of objects

Inheritance

A mechanism of reusability, the most powerful concept of OO programming





Association

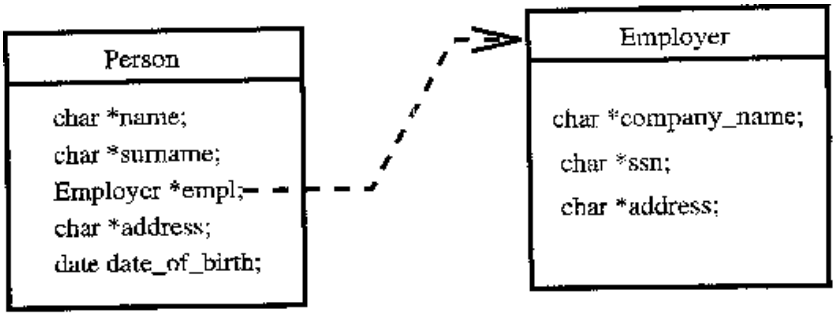
Association is a link between entities in an application

In OODB, associations are represented by means of references between objects

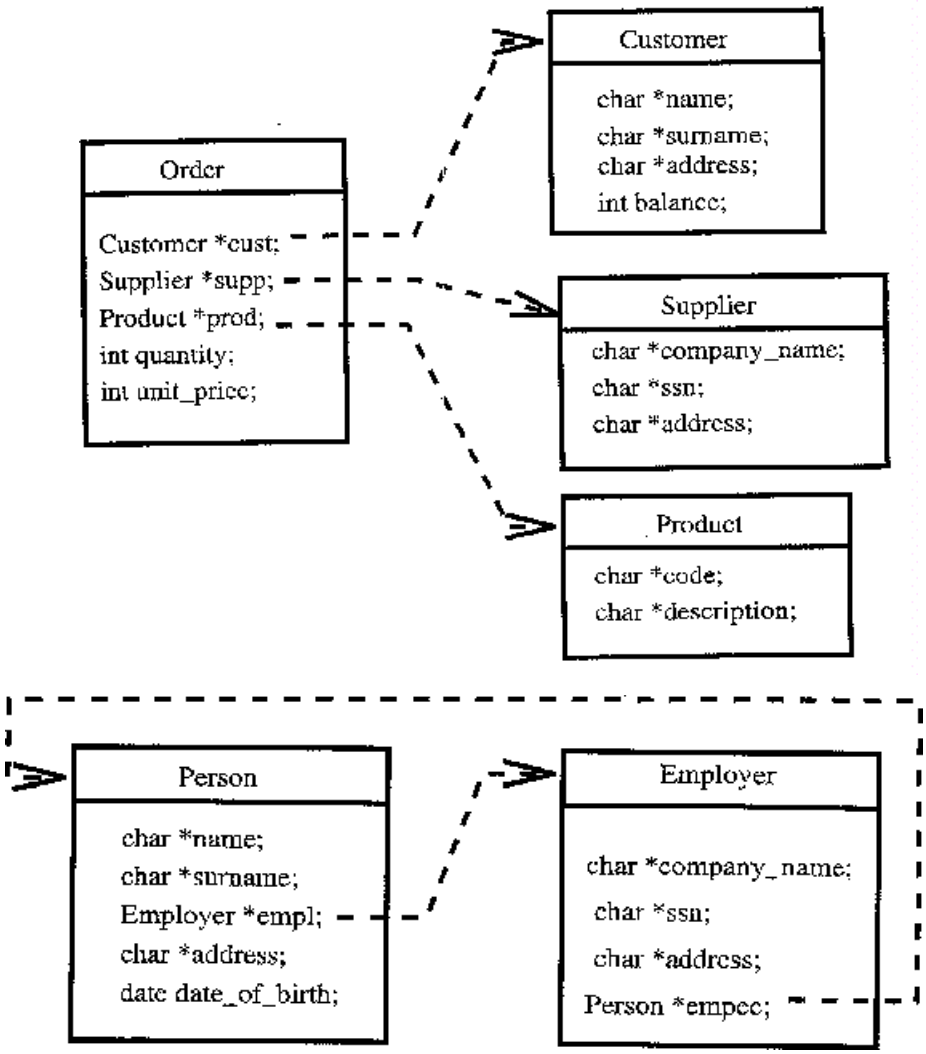
a representation of a binary association

a representation of a ternary association

reverse reference



Representation of a binary Association



ADVANTAGES OF OODB

- An integrated repository of information that is shared by multiple users, multiple products, multiple applications on multiple platforms.
- It also solves the following problems:

1. The semantic gap: The real world and the Conceptual model is very similar.
2. Impedance mismatch: Programming languages and database systems must be interfaced to solve application problems. But the language style, data structures, of a programming language (such as C) and the DBMS (such as Oracle) are different. The OODB supports general purpose programming in the OODB framework.
3. New application requirements: Especially in OA, CAD, CAM, CASE, object-orientation is the most natural and most convenient.

COMPLEX OBJECT DATA MODELS

Complex object data model is non-1NF data model. It allows the following extensions:

1. Sets of atomic values
2. Tuple-valued attributes
3. Sets of tuples (nested relations)
4. General set and tuple constructors
5. Object identity

Formal definition:

1. Every atomic value in A is an object.
2. If a1, ..., an are attribute names in N, and O1, ..., On are objects, then T = [a1:O1, ..., an:On] is also an object, and T.ai retrieves the value Oi.
3. If O1, ..., On are objects, then S = {O1, ..., On} is an abject.

Example: {[Name:John, Age: 30], [Name:Mary, Friends:{Mark, Vicki}]}

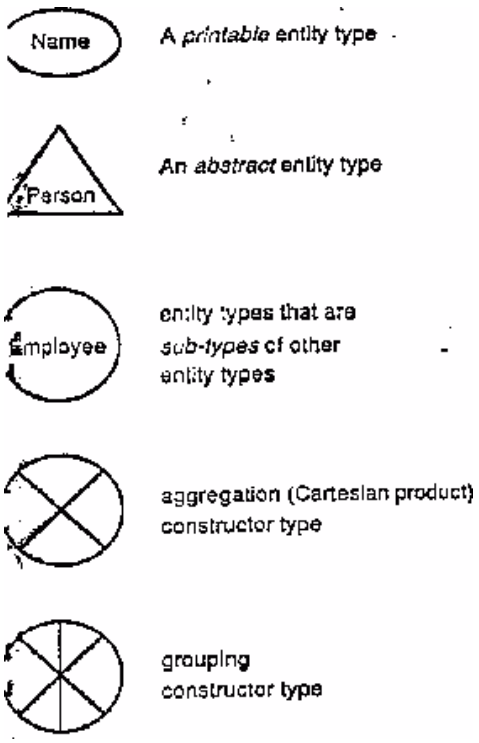
An object is defined by a triple (OID, type constructor, state) where OID is the unique object identifier, type constructor is its type (such as atom, tuple, set, list, array, bag, etc.) and state is its actual value.

Example:

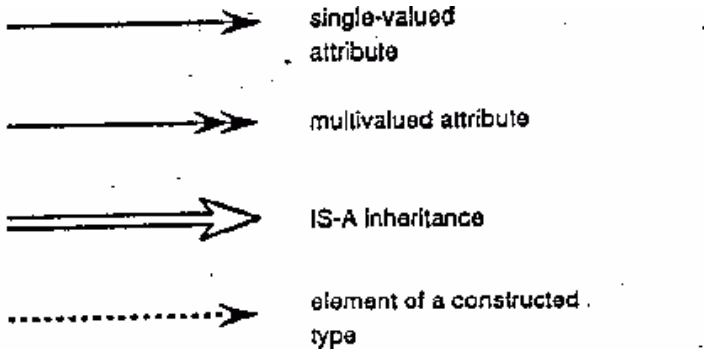
(i1, atom, 'John')
(i2, atom, 30)
(i3, atom,
'Mary')
(i4, atom, 'Mark')
(i5, atom 'Vicki')
(i6, tuple, [Name:i1, Age:i3])
(i7, set, {i4, i5})
(i8, tuple, [Name:i3,
Friends:i7]) (i9, set, {i6, i8})

Semantic Data Models

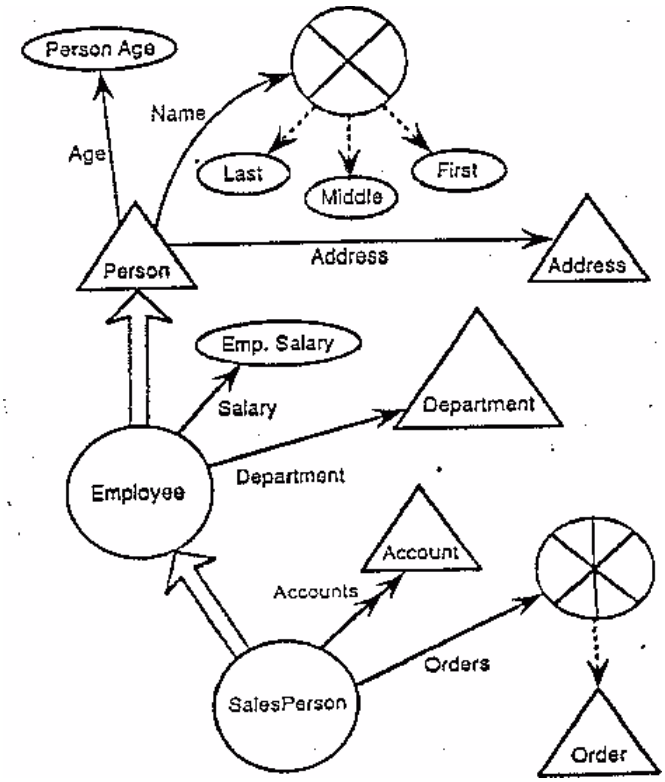
The following figure shows different types of nodes in GSM.



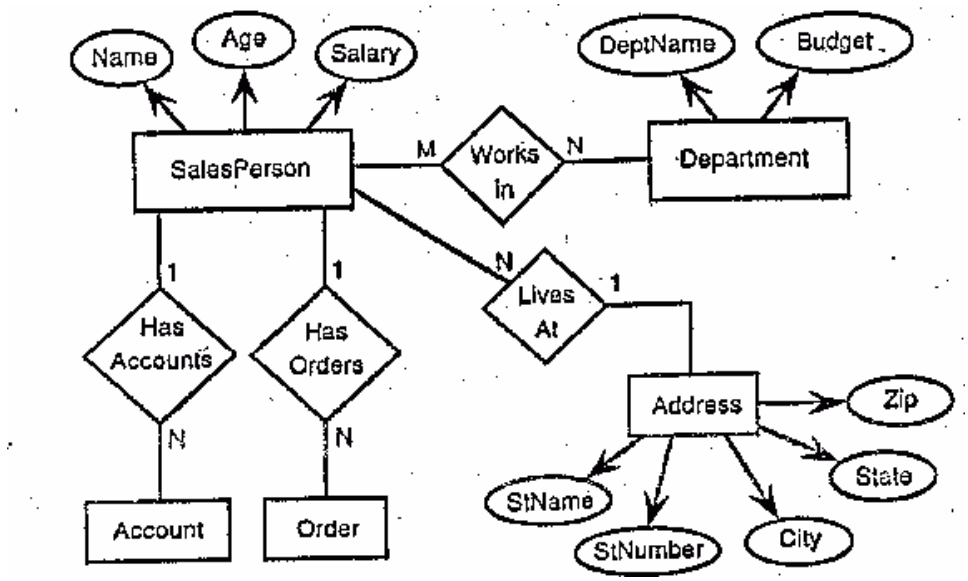
The different types of links (relationships) in GSM are shown below.



A semantic model for the sales office automation example is shown below.



An entity-relationship model for the sales office automation example is shown below.



OBJECT-ORIENTED DATABASES

OODB = Object Orientation + Database
Capabilities FEATURES TO BE CONSIDERED:

Persistence
support of transactions
simple querying of bulk
data concurrency control
resilience and recovery
security
versioning integrity
performance
issues

DATA MODELS TO BE CONSIDERED:
Complex object model
Semantic data model such as Extended ER (EER) model, OPM model
NOT ALL OODB SUPPORTS SAME OBJECT-ORIENTATION, SAME DATA MODEL AND SAME SET OF FEATURES

RESEARCH PROTOTYPES

- ORION: Lisp-based system, built at MCC 1987. Handles schema evolution and complex object locking.
- IRIS: Built at HP 1987. Functional data model, version control, object-SQL.
- Galileo: Built at Univ Pisa 1985. Strong typed language, complex objects.
- PROBE: CCA 1986.
- POSTGRES: Univ. California, Berkeley 1990. Extended relational database supporting objects.

COMMERCIAL OODB

- O2: O2 Technology. Language O2C to define classes, methods and types. Supports multiple inheritance. C++ compatible. Supports an extended SQL language O2SQL which can refer to complex objects.
- G-Base: Graphael 1987. Lisp-based system, supports ADT, multiple inheritance of classes.
- CORBA: Standards for distributed objects.
- GemStone: Servio Logic 1987, Beaverton, Oregon. Earliest OODB supporting object identity, inheritance, encapsulation. Language OPAL is based upon Smalltalk.
- Ontos: Ontologic, 1988, Berlington, Mass. C++ based system, offers C++ clients library. Ontos has a predecessor called Vbase. Ontos model supports encapsulation, inheritance, ability to construct complex objects.

- Object Store: Object Design Inc. C++ based sustem. A good feature is that it supports the creation of indexes.
- Statics: Symbolics 1988. Supports entity types, set valued attributes, and inheritance of entity types and methods.
- SIM: Semantic Information Manager, UNISYS 1987. Supports semantic data model. Core system of the InfoExec Environment of UNISYS. Uses the Semantic Data Model of Hammer and McLeod 1981. User can define entity types that can inherit from one another. Attributes of entities are like functions from one entity to another.
- Relational DB Extensions: Many relational systems support OODB extensions.

1. User-defined functions (dBase).
2. User-defined ADTs (POSTGRES)
3. Very-long multimedia fields (BLOB or Binary Large Object). (DB2 from IBM, SQL from SYBASE, Informix, Interbase)

ALTERNATIVE OODB STRATEGIES

1. Develop novel database data model or data language (SIM)
2. Extend an existing database language with object-oriented capabilities. (IRIS, O2 and VBASE/ONTOS extended SQL)
3. Extend existing object-oriented programming language with database capabilities (GemStone OPAL extended SmallTalk)
4. Extendable object-oriented DBMS library (ONTOS)

OODB QUERY LANGUAGE

ONTOS (from Ontologic), O2 (from O2 Technology) and IRIS (from HP) all offer object-oriented extension of SQL.
IRIS has Object SQL. Each entity type specifies one or more properties. Properties are functions that apply to the instances of the type. Entity types have extensions and can inherit from one another.

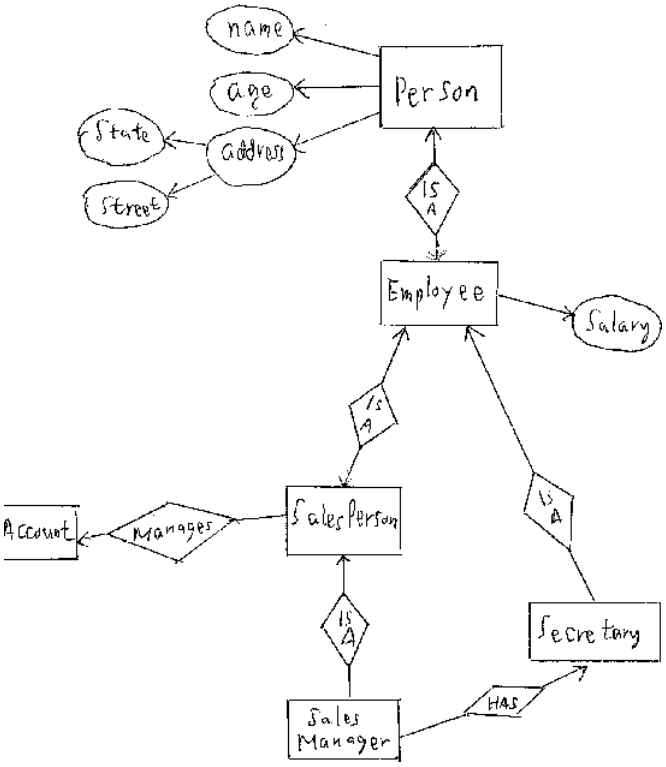
Example: The type PERSON can be created by:

```
Create Type Person (
  Name          char(20),
  Age           integer
  Address       address)
```

where address is another type. In IRIS, Name, Age and Address are called properties (functions) and apply to instances of type Person.
Object SQL Query: retrieve the name and state of all people who are older than 21:

Select Name(p), State(Address(p)
for each Person p
where Age(p) > 21
(Here we assume address has another property called "state")
The user can compose the function application: P5(P4(...P1(P))...)in the select statement.

EXAMPLE:
Employee is a subtype of Person
SalesPerson is a subtype of
Employee Addresses have street
names Salespeople have sales
managers sales managers have
secretaries secretaries are



employees

Retrieve the street name of the address of the secretary of the manager of each salesperson whose salary is more than 50,000:

Select
StreetName(Address(Secretary(Manager(0))))
for each SalesPerson p
where Salary(p) > $50,000

WHY EXTENDING SQL?

- SQL is the most popular relational query language
- SQL is also the only relational language that has a standard.
- SQL is being promoted by many companies as the interface language for database engines and database servers. New applications developed in SQL extensions can easily call these servers for remote access.

ENHANCED DATA MODELS FOR ADVANCED APPLICATION EMERGING DATABASE TECHNOLOGIES AND APPLICATION

Active Database Concepts and Triggers

Generalized Model for Active Databases and Oracle **Triggers**

- **Triggers** are executed when a specified condition occurs during insert/delete/update
  - Triggers are action that fire automatically based on these conditions

Event-Condition-Action (ECA) Model

Generalized Model (contd.)

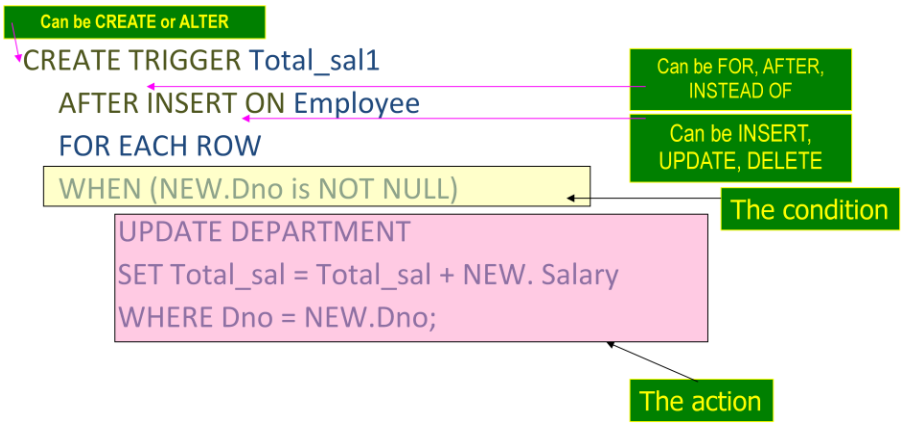
- Triggers follow an Event-condition-action (ECA) model
  - **Event**:
    - Database modification
      - E.g., insert, delete, update),
  - **Condition**:
    - Any true/false expression
      - Optional: If no condition is specified then condition is always true
  - **Action**:
    - Sequence of SQL statements that will be automatically executed

Trigger Example

- When a new employees is added to a department, modify the Total_sal of the Department to include the new employees salary
  - Logically this means that we will CREATE a TRIGGER, let us call the trigger Total_sal1
    - This trigger will execute AFTER INSERT ON Employee table
    - It will do the following FOR EACH ROW
      - WHEN NEW.Dno is NOT NULL
      - The trigger will UPDATE DEPARTMENT
      - By SETting the new Total_sal to be the sum of

- old Total_sal and NEW. Salary
- WHERE the Dno matches the NEW.Dno;

Example :



CREATE or ALTER TRIGGER

- CREATE TRIGGER <name>
  - Creates a trigger
- ALTER TRIGGER <name>
  - Alters a trigger (assuming one exists)
- CREATE OR ALTER TRIGGER <name>
  - Creates a trigger if one does not exist
  - Alters a trigger if one does exist
    - Works in both cases, whether a trigger exists or not

Conditions

- AFTER
  - Executes after the event
- BEFORE
  - Executes before the event
- INSTEAD OF
  - Executes **instead of** the event
    - Note that event does not execute in this case
      - E.g., used for modifying views

Row-Level versus Statement-level

- Triggers can be
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**

        - Default (when FOR EACH ROW is not specified)
- Row level triggers
    - Executed separately for each affected row
- Statement-level triggers
    - Execute once for the SQL statement,

Condition

- Any true/false condition to control whether a trigger is activated on not
    - Absence of condition means that the trigger will always execute for the even
    - Otherwise, condition is evaluated
        - before the event for BEFORE trigger
        - after the event for AFTER trigger

Action

- Action can be
    - One SQL statement
    - A sequence of SQL statements enclosed between a BEGIN and an END
- Action specifies the relevant modifications

Triggers on Views

- INSTEAD OF triggers are used to process view modifications

Active Database Concepts and Triggers

Design and Implementation Issues for Active Databases

- An active database allows users to make the following changes to triggers (rules)
    - Activate
    - Deactivate
    - Drop

Design and Implementation Issues for Active Databases

- An event can be considered in 3 ways
    - Immediate consideration
    - Deferred consideration
    - Detached consideration

Design and Implementation Issues (contd.)

- Immediate consideration
  - Part of the same transaction and can be one of the following depending on the situation
    - Before
    - After
    - Instead of
- Deferred consideration
  - Condition is evaluated at the end of the transaction
- Detached consideration
  - Condition is evaluated in a separate

transaction Potential Applications for Active Databases

- Notification
  - Automatic notification when certain condition occurs
  - Enforcing integrity constraints
    - Triggers are smarter and more powerful than constraints
  - Maintenance of derived data
    - Automatically update derived data and avoid anomalies due to redundancy
      - E.g., trigger to update the Total_sal in the earlier example

Triggers in SQL-99

- Can alias variables inside the REFERENCINFG

clause Trigger examples

```
T1:  CREATE TRIGGER Total_sal1
     AFTER UPDATE OF Salary ON EMPLOYEE
     REFERENCING OLD ROW AS O, NEW ROW AS N
     FOR EACH ROW
     WHEN  ( N.Dno IS NOT NULL )
     UPDATE DEPARTMENT
     SET Total_sal = Total_sal + N.salary – O.salary
     WHERE Dno = N.Dno;

T2:  CREATE TRIGGER Total_sal2
     AFTER UPDATE OF Salary ON EMPLOYEE
     REFERENCING OLD TABLE AS O, NEW TABLE AS N
     FOR EACH STATEMENT
     WHEN  EXISTS  ( SELECT * FROM N WHERE N.Dno IS NOT NULL ) OR
           EXISTS  ( SELECT * FROM O WHERE O.Dno IS NOT NULL )
     UPDATE DEPARTMENT AS D
     SET D.Total_sal = D.Total_sal
     + ( SELECT SUM (N.Salary) FROM N WHERE D.Dno=N.Dno )
     – ( SELECT SUM (O.Salary) FROM O WHERE D.Dno=O.Dno )
```

Temporal Database Concepts

Time Representation, Calendars, and Time Dimensions

- Time is considered ordered sequence of points in some granularity
  - Use the term chorononon instead of point to describe minimum granularity

- A calendar organizes time into different time units for convenience.
  - Accommodates various calendars
    - Gregorian (western)
    - Chinese
    - Islamic
    - Hindu
    - Jewish
    - Etc.

- Point events
  - Single time point event
    - E.g., bank deposit
  - Series of point events can form a time series data
- Duration events
  - Associated with specific time period
    - Time period is represented by start time and end time

- Transaction time
  - The time when the information from a certain transaction becomes valid

- Bitemporal database
  - Databases dealing with two time dimensions

Incorporating Time in Relational Databases Using Tuple

Versioning

- Add to every tuple
  - Valid start time
  - Valid end time

Figure 24.7
Different types of temporal relational databases. (a) Valid time database schema. (b) Transaction time database schema. (c) Bitemporal database schema.

Figure 24.8
Some tuple versions in the valid time relations EMP_VT and DEPT_VT.

EMP_VT

Name	Ssn	Salary	Dno	Supervisor_ssn	Vst	Vet
Smith	123456789	25000	5	333445555	2002-06-15	2003-05-31
Smith	123456789	30000	5	333445555	2003-06-01	Now
Wong	333445555	25000	4	999887777	1999-08-20	2001-01-31
Wong	333445555	30000	5	999887777	2001-02-01	2002-03-31
Wong	333445555	40000	5	888665555	2002-04-01	Now
Brown	222447777	28000	4	999887777	2001-05-01	2002-08-10
Narayan	666884444	38000	5	333445555	2003-08-01	Now

. . .

DEPT_VT

Dname	Dno	Manager_ssn	Vst	Vet
Research	5	888665555	2001-09-20	2002-03-31

Incorporating Time in Object-Oriented Databases Using Attribute Versioning

- A single complex object stores all temporal changes of the object
- Time varying attribute
  - An attribute that changes over time
    - E.g., age
- Non-Time varying attribute
  - An attribute that does not changes over time
    - E.g., date of birth

Introduction to Deductive Databases

Overview of Deductive Databases

- Declarative Language
  - Language to specify rules
- Inference Engine (Deduction Machine)
  - Can deduce new facts by interpreting the rules
  - Related to logic programming
    - Prolog language (Prolog => Programming in logic)
    - Uses backward chaining to evaluate
      - Top-down application of the rules
- Speciation consists of:

        - Similar to relation specification without the necessity of including attribute names
    - Rules
        - Similar to relational views (virtual relations that are not stored)

Prolog/Datalog Notation

- Predicate has
    - a name
    - a fixed number of arguments
        - Convention:
            - Constants are numeric or character strings
        - Variables start with upper case letters
    - E.g., SUPERVISE(Supervisor, Supervisee)
        - States that Supervisor SUPERVISE(s) Supervisee
- Rule
    - Is of the form head :- body
        - where :- is read as if and only iff
    - E.g., SUPERIOR(X,Y) :- SUPERVISE(X,Y)
    - E.g., SUBORDINATE(Y,X) :- SUPERVISE(X,Y)
- Query
    - Involves a predicate symbol followed by y some variable arguments to answer the question
        - where :- is read as if and only iff
    - E.g., SUPERIOR(james,Y)?
    - E.g., SUBORDINATE(james,X)?
- (a) Prolog notation (b) 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).
. . .

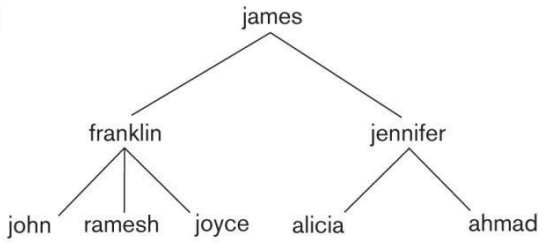
Rules

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)?

(b)



Datalog Notation

- Datalog notation
  - Program is built from atomic formulae
    - Literals of the form p(a1, a2, … an) where
      - p predicate name
      - n is the number of arguments
    - Built-in predicates are included
      - E.g., <, <=, etc.
  - A literal is either
    - An atomic formula
    - An atomic formula preceded by not

Clausal Form and Horn Clauses

- A formula can have quantifiers
  - Universal
  - Existential
- In clausal form, a formula must be transformed into another formula with the following characteristics
  - All variables are universally quantified
  - Formula is made of a number of clauses where each clause is made up of literals connected by logical ORs only
  - Clauses themselves are connected by logical ANDs only.
- Any formula can be converted into a clausal form

- A specialized case of clausal form are horn clauses that can contain no more than one positive literal
- Datalog program are made up of horn clauses

Interpretation of Rules

- There are two main alternatives for interpreting rules:
  - Proof-theoretic
  - Model-theoretic
- Proof-theoretic
  - Facts and rules are axioms
  - Ground axioms contain no variables
  - Rules are deductive axioms
  - Deductive axioms can be used to construct new facts from existing facts
  - This process is known as theorem proving

Proving a new fact

- Figure 24.12

1. SUPERIOR(X, Y) :– SUPERVISE(X, Y). (rule 1)
2. SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y). (rule 2)

3. SUPERVISE(jennifer, ahmad). (ground axiom, given)
4. SUPERVISE(james, jennifer). (ground axiom, given)
5. SUPERIOR(jennifer, ahmad). (apply rule 1 on 3)
6. SUPERIOR(james, ahmad). (apply rule 2 on 4 and 5)

Interpretation of Rules

- Model-theoretic
  - Given a finite or infinite domain of constant values, we assign the predicate every combination of values as arguments
  - If this is done fro every predicated, it is called interpretation

- Model
  - An interpretation for a specific set of rules
- Model-theoretic proofs
  - Whenever a particular substitution to the variables in the rules is applied, if all the predicated are true under the interpretation, the predicate at the head of the rule must also be true
- Minimal model

- Cannot change any fact from true to false and still get a model for these rules

- Figure 24.13

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

Datalog Programs and Their Safety

- Two main methods of defining truth values
  - Fact-defined predicates (or relations)
    - Listing all combination of values that make a predicate true
  - Rule-defined predicates (or views)
    - Head (LHS) of 1 or more Datalog rules, for example Figure 24.15

```
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), WORKS_ON(X, productx, Y), Y >=20.
```

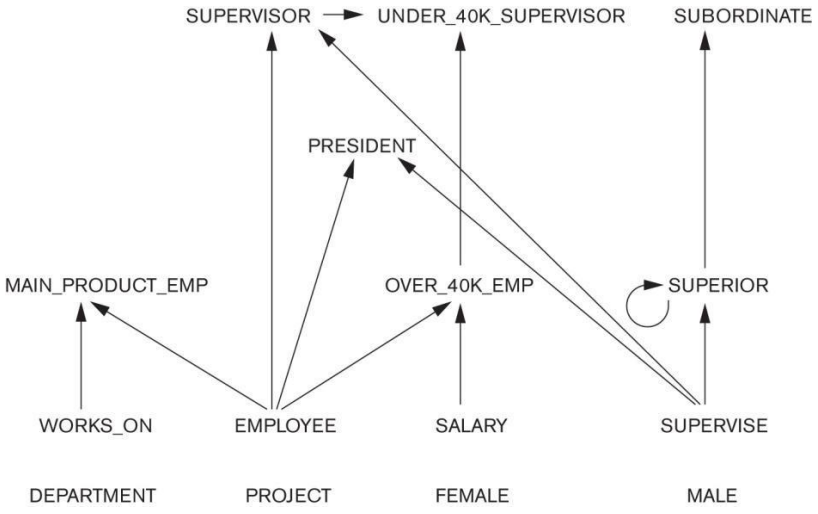
- A program is safe if it generates a finite set of facts
  - Fact-defined predicates (or relations)
  - Listing all combination of values that make a predicate true
    - Rule-defined predicates (or views)
- Head (LHS) of 1 or more Datalog rules, for example Figure 24.15

Use the Relational Operations

- Many operations of relational algebra can be defined in the for of Datalog rules that defined the result of applying these operations on database relations (fact predicates)
  - Relational queries and views can be easily specified in Datalog

Evaluation of Non-recursive Datalog Queries

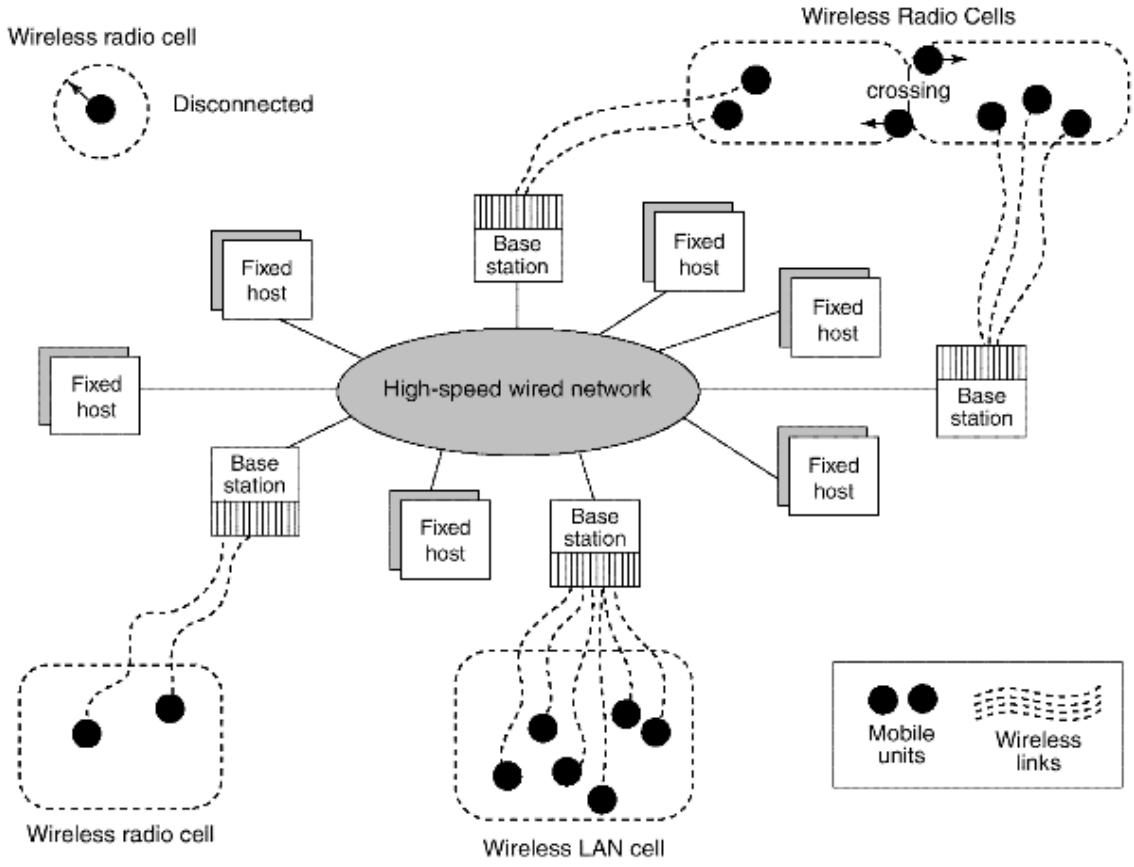
- Define an inference mechanism based on relational database query processing concepts
  - See Figure 24.17 on predicate dependencies for Figs 24.14 and 24.15



Emerging Database Technologies and Application

1. **Mobile Databases**

A. Mobile Computing Architecture



It is distributed architecture where a number of computers generally referred to as Fixed Hosts and Base Stations are interconnected through a high-speed wired network. Fixed hosts are general-purpose computers configured to manage mobile units. Base stations function as gateways to the fixed network for the Mobile Units.

Wireless Communications

The wireless medium have bandwidth significantly lower than those of a wired network.

The current generation of wireless technology has data rates range from the tens to hundreds of kilobits per second (2G cellular telephony) to tens of megabits per second (wireless Ethernet, popularly known as WiFi).

Modern (wired) Ethernet, by comparison, provides data rates on the order of hundreds of megabits per second.

The other characteristics distinguish wireless connectivity options:

- Interference,
- Locality of access,
- Range,
- Support for packet switching,
- Seamless roaming throughout a geographical region.

Some wireless networks, such as WiFi and Bluetooth, use unlicensed areas of the frequency spectrum, which may cause interference with other appliances, such as cordless telephones. Modern wireless networks can transfer data in units called packets, that are used in wired networks in order to conserve bandwidth.

Client/Network Relationships

Mobile units can move freely in a geographic mobility domain, an area that is circumscribed by wireless network coverage. To manage entire mobility domain is divided into one or more smaller domains, called cells, each of which is supported by at least one base station. In a MANET, co-located mobile units do not need to communicate via a fixed network, but instead, form their own using cost- effective technologies such as Bluetooth. In a MANET, mobile units are responsible for routing their own data, effectively acting as base stations as well as clients. Moreover, they must be robust enough to handle changes in the network topology, such as the arrival or departure of other mobile units.

MANET applications can be considered as peer-to-peer, meaning that a mobile unit is simultaneously a client and a server. Transaction processing and data consistency control become more difficult since there is no central control in this architecture. Resource discovery and data routing by mobile units make computing in a MANET even more complicated. Sample MANET applications are multi-user games, shared whiteboard, distributed calendars, and battle information sharing. Mobile units be unrestricted throughout the cells of domain, while maintaining information access contiguity.

The communication architecture described earlier is designed to give the mobile unit the impression that it is attached to a fixed network, emulating a traditional client-server architecture. Wireless communications, however, make other architectures possible. One alternative is a mobile ad-hoc network (MANET).

In a MANET, co-located mobile units do not need to communicate via a fixed network, but instead, form their own using cost- effective technologies such as Bluetooth. In a MANET, mobile units are responsible for routing their own data, effectively acting as base stations as well as clients. Moreover, they must be robust enough to handle changes in the network topology, such as the arrival or departure of other mobile units.

MANET applications can be considered as peer-to-peer, meaning that a mobile unit is simultaneously a client and a server. Transaction processing and data consistency control become more difficult since there is no central control in this architecture. Resource discovery and data routing by mobile units make computing in a MANET even more

complicated. Sample MANET applications are multi-user games, shared whiteboard, distributed calendars, and battle information sharing.

B. Characteristics of Mobile Environments

The characteristics of mobile computing include:

- Communication latency
- Intermittent connectivity
- Limited battery life
- Changing client location

The server may not be able to reach a client. A client may be unreachable because it is dozing – in an energy- conserving state in which many subsystems are shut down – or because it is out of range of a base station. In either case, neither client nor server can reach the other, and modifications must be made to the architecture in order to compensate for this case. Proxies for unreachable components are added to the architecture.

For a client (and symmetrically for a server), the proxy can cache updates intended for the server. Mobile computing poses challenges for servers as well as clients. The latency involved in wireless communication makes scalability a problem. Since latency due to wireless communications increases the time to service each client request, the server can handle fewer clients.

One way servers relieve this problem is by broadcasting data whenever possible. A server can simply broadcast data periodically. Broadcast also reduces the load on the server, as clients do not have to maintain active connections to it. Client mobility also poses many data management challenges. Servers must keep track of client locations in order to efficiently route messages to them.

Client data should be stored in the network location that minimizes the traffic necessary to access it. The act of moving between cells must be transparent to the client.

The server must be able to gracefully divert the shipment of data from one base to another, without the client noticing. Client mobility also allows new applications that are location-based.

C. Data Management Issues

Data management issues as it is applied to mobile databases:

- Data distribution and replication
- Transactions models
- Query processing
- Recovery and fault tolerance
- Mobile database design
- Location-based service
- Division of labor

- Security

D. Application: Intermittently Synchronized Databases

Whenever clients connect – through a process known in industry as synchronization of a client with a server – they receive a batch of updates to be installed on their local database. The primary characteristic of this scenario is that the clients are mostly disconnected; the server is not necessarily able reach them. This environment has problems similar to those in distributed and client-server databases, and some from mobile databases. This environment is referred to as Intermittently Synchronized Database Environment (ISDBE).

The characteristics of Intermittently Synchronized Databases (ISDBs) make them distinct from the mobile databases are:

- A client connects to the server when it wants to exchange updates.
- The communication can be unicast –one-on-one communication between the server and the client– or multicast– one sender or server may periodically communicate to a set of receivers or update a group of clients.
- A server cannot connect to a client at will.
- Issues of wireless versus wired client connections and power conservation are generally immaterial.
- A client is free to manage its own data and transactions while it is disconnected. It can also perform its own recovery to some extent.
- A client has multiple ways connecting to a server and, in case of many servers, may choose a particular server to connect to based on proximity, communication nodes available, resources available, etc.

2. Multimedia Databases

A. The Nature of Multimedia Data and Applications

In the years ahead multimedia information systems are expected to dominate our daily lives. Our houses will be wired for bandwidth to handle interactive multimedia applications. Our high-definition TV/computer workstations will have access to a large number of databases, including digital libraries, image and video databases that will distribute vast amounts of multisource multimedia content. DBMSs have been constantly adding to the types of data they support. Today many types of multimedia data are available in current systems.

Types of multimedia data are available in current systems

Text: May be formatted or unformatted. For ease of parsing structured documents, standards like SGML and variations such as HTML are being used.

Graphics: Examples include drawings and illustrations that are encoded using some descriptive standards (e.g. CGM, PICT, postscript).

Images: Includes drawings, photographs, and so forth, encoded in standard formats such as bitmap, JPEG, and MPEG. Compression is built into JPEG and MPEG.

These images are not subdivided into components.

Hence querying them by content (e.g., find all images containing circles) is nontrivial.

Animations: Temporal sequences of image or graphic data.

Video: A set of temporally sequenced photographic data for presentation at specified rates– for example, 30 frames per second.

Structured audio: A sequence of audio components comprising note, tone, duration, and so forth.

Audio: Sample data generated from aural recordings in a string of bits in digitized form. Analog recordings are typically converted into digital form before storage.

Composite or mixed multimedia data: A combination of multimedia data types such as audio and video which may be physically mixed to yield a new storage format or logically mixed while retaining original types and formats. Composite data also contains additional control information describing how the information should be rendered.

Nature of Multimedia Applications:

Multimedia data may be stored, delivered, and utilized in many different ways.

Applications may be categorized based on their data management characteristics.

B. Data Management Issues

Characterization of applications based on their data management characteristics:

Repository applications: A large amount of multimedia data as well as metadata is stored for retrieval purposes. Examples include repositories of satellite images, engineering drawings and designs, space photographs, and radiology scanned pictures.

Presentation applications: A large amount of applications involve delivery of multimedia data subject to temporal constraints; simple multimedia viewing of video data, for example, requires a system to simulate VCR-like functionality. Complex and interactive multimedia presentations involve orchestration directions to control the retrieval order of components in a series or in parallel. Interactive environments must support capabilities such as real-time editing analysis or annotating of video and audio data.

Collaborative work using multimedia information: This is a new category of applications in which engineers may execute a complex design task by merging drawings, fitting subjects to design constraints, and generating new documentation, change notifications, and so forth. Intelligent healthcare networks as well as telemedicine will involve doctors collaborating among themselves, analyzing multimedia patient data and information in real time as it is generated. Multimedia applications dealing with thousands of images, documents, audio and video segments, and free text data depend critically on

- Appropriate modeling of the structure and content of data
- Designing appropriate database schemas for storing and retrieving multimedia information.
- Multimedia information systems are very complex and embrace a large set of issues:

- Modeling
- Complex objects
- Design
- Conceptual, logical, and physical design of multimedia has not been addressed fully.
- Storage

Multimedia data on standard disklike devices presents problems of representation, compression, mapping to device hierarchies, archiving, and buffering during the input/output operation.

Queries and retrieval

"Database" way of retrieving information is based on query languages and internal index structures.

Performance

Multimedia applications involving only documents and text, performance constraints are subjectively determined by the user. Applications involving video playback or audio-video synchronization, physical limitations dominate.

C. Multimedia Database Applications

Large-scale applications of multimedia databases can be expected encompasses a large number of disciplines and enhance existing capabilities.

- Documents and records management
- Knowledge dissemination
- Education and training
- Marketing, advertising, retailing, entertainment, and travel
- Real-time control and monitoring

3. Geographic Information Systems (GIS)

Geographic information systems (GIS) are used to collect, model, and analyze information describing physical properties of the geographical world.

The scope of GIS broadly encompasses two types of data:

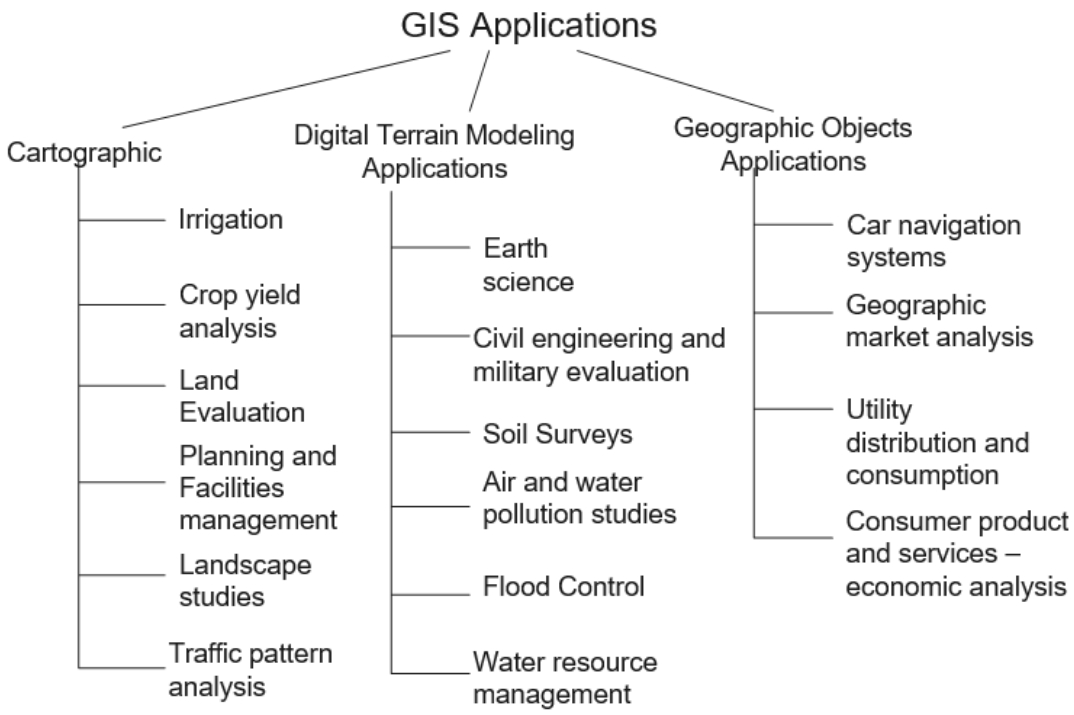
Spatial data, originating from maps, digital images, administrative and political boundaries, roads, transportation networks, physical data, such as rivers, soil characteristics, climatic regions, land elevations, and

Non-spatial data, such as socio-economic data (like census counts), economic data, and sales or marketing information. GIS is a rapidly developing domain that offers highly innovative approaches to meet some challenging technical demands.

A. GIS Applications

It is possible to divide GISs into three categories:

- Cartographic applications
- Digital terrain modeling applications
- Geographic objects applications



B. Data Management Requirements of GIS

The functional requirements of the GIS applications above translate into the following database requirements.

Data Modeling and Representation

GIS data can be broadly represented in two formats:

- **Vector** data represents geometric objects such as points, lines, and polygons.
- **Raster** data is characterized as an array of points, where each point represents the value of an attribute for a real-world location.

Informally, raster images are n-dimensional array where each entry is a unit of the image and represents an attribute. Two-dimensional units are called pixels, while three-dimensional units are called voxels. Three-dimensional elevation data is stored in a raster-

based digital elevation model (DEM) format. Another raster format called **triangular irregular network (TIN)** is a topological vector-based approach that models surfaces by connecting sample points as vector of triangles and has a point density that may vary with the roughness of the terrain.

Rectangular grids (or elevation matrices) are two- dimensional array structures.

In **digital terrain modeling (DTM)**, the model also may be used by substituting the elevation with some attribute of interest such as population density or air temperature.

GIS data often includes a temporal structure in addition to a spatial structure.

Data Analysis

GIS data undergoes various types of analysis.

For example, in applications such as soil erosion studies, environmental impact studies, or hydrological runoff simulations,

DTM data may undergo various types of geomorphometric analysis – measurements such as slope values, gradients (the rate of change in altitude), aspect (the compass direction of the gradient), profile convexity (the rate of change of gradient), plan convexity (the convexity of contours and other parameters).

Data Integration

GISs must integrate both vector and raster data from a variety of sources.

Sometimes edges and regions are inferred from a raster image to form a vector model, or conversely, rasterimages such as aerial photographs are used to update vector models.

Several **coordinate systems** such as Universal Transverse Mercator (UTM), latitude/longitude, and local cadastral systems are used to identify locations.

Data originating from different coordinate systems requires appropriate transformations.

Data Capture

The first step in developing a spatial database for cartographic modeling is to capture the two-dimensional or three-dimensional geographical information in digital form

This process that is sometimes impeded by source map characteristics such as resolution, type of projection, map scales, cartographic licensing, diversity of measurement techniques, and coordinate system differences.

Spatial data can also be captured from remote sensors in satellites such as Landsat, NORA, and Advanced Very High Resolution Radiometer(AVHRR) as well as SPOT HRV (High Resolution Visible Range Instrument.

C. Specific GIS Data Operations

GIS applications are conducted through the use of special operators such as the following:

- Interpolation
- Interpretation
- Proximity analysis
- Raster image processing
- Analysis of network

The functionality of a GIS database is also subject to other considerations:

- Extensibility
- Data quality control
- Visualization

Such requirements clearly illustrate that standard RDBMSs or ODBMSs do not meet the special needs of GIS. Therefore it is necessary to design systems that support the vector and raster representations and the spatial functionality as well as the required DBMS features.

4. GENOME Data Management

A. Biological Sciences and Genetics

The biological sciences encompass an enormous variety of information.

- **Environmental** science gives us a view of how species live and interact in a world filled with natural phenomena.
- **Biology** and **ecology** study particular species.
- **Anatomy** focuses on the overall structure of an organism, documenting the physical aspects of individual bodies.

Traditional medicine and physiology break the organism into systems and tissues and strive to collect information on the workings of these systems and the organism as a whole. Histology and cell biology delve into the tissue and cellular levels and provide knowledge about the inner structure and function of the cell. This wealth of information that has been generated, classified, and stored for centuries has only recently become a major application of database technology.

Genetics has emerged as an ideal field for the application of information technology.

In a broad sense, it can be taught of as the construction of models based on information about genes and population and the seeking out of relationships in that information.

Genes can be defined as units of heredity. The study of genetics can be divided into three branches:

Mendelian genetics is the study of the transmission of traits between generations.

Molecular genetics is the study of the chemical structure and function of genes at the molecular level.

Population genetics is the study of how genetic information varies across populations of organisms.

The origins of **molecular genetics** can be traced to two important discoveries:

In 1869 when Friedrich Miescher discovered nuclein and its primary component, deoxyribonucleic acid (DNA). In subsequent research DNA and a related compound, ribonucleic acid, were found to be composed of nucleotides (a sugar, a phosphate, and a base combining to form nucleic acid) linked into long polymers via the sugar and phosphate.

The second discovery was the demonstration in 1944 by Oswald Avery that DNA was indeed the molecular. Genes were shown to be composed of chains of nucleic acids arranged linearly on chromosomes and to serve three primary functions:

- Replicating genetic information between generations,
- Providing blueprints for the creation of polypeptides, and
- Accumulating changes– thereby allowing evolution to occur.

Watson and Crick found the double-helix structure of the DNA in 1953, which gave molecular biology a new direction.

Characteristics of Biological Data

1. Biological data exhibits many special characteristics that make management of biological information a particularly challenging problem.
2. The characteristics related to biological information, and focusing on a multidisciplinary field called bioinformatics that has emerged.
3. Bioinformatics addresses information management of genetic information with special emphasis on DNA sequence analysis.
4. Applications of bioinformatics span design of targets for drugs, study of mutations and related diseases, anthropological investigations on migration patterns of tribes and therapeutic treatments.
5. Biological data is highly complex when compared with most other domains or applications.
6. The amount and range of variability in data is high.
7. Schemas in biological databases change at a rapid pace.
8. Representations of the same data by different biologists will likely be different (even using the same system).
9. Most users of biological data do not require write access to the database; read-only access is adequate.

10. Most biologists are not likely to have knowledge of the internal structure of the database or about schema design.
11. The context of data gives added meaning for its use in biological applications.
12. Defining and representing complex queries is extremely important to the biologist.
13. Users of biological information often require access to “old” values of the data – particularly when verifying previously reported results.