

DATABASE MANAGEMENT SYSTEM

Prepared by Bhabani Shankar Pradhan;

Subject Code	Name of the Subject	L	T	P	C	QP
BCSPC4020	DATABASE MANAGEMENT SYSTEM	3	0	0	3	A

Course Educational Objectives

Pre -Requisite: A student should have basic idea on logic gates

CEO 1	Gain a good understanding of the architecture and functioning of Database Management Systems as well as associated tools and techniques.
CEO 2	Understand and apply the principles of data modeling using Entity Relationship and develop a good database design.
CEO3	Understand the use of Structured Query Language (SQL) and its syntax
CEO4	Apply Normalization techniques to normalize a database.

Course Outcomes: Upon successful completion of this course, students should be able to:

CO1	Identify and Classify the concepts of Database Management system, Data models and architecture of database, ER to Relational mapping concepts.
CO2	Applying the constraints in database using different query languages like:- relational algebra and calculus, SQL and QBE for the implementing the Data definition and data manipulate languages in Database.
CO3	Compare the different normal forms to Apply normalization process to construct the consistent Database.
CO4	Design and Develop the Database by inspecting concurrency control and recovery strategies to make complete Database without confliction and anomalies in concurrent access environment.

SYLLABUS

UNIT:1

(15 Hours)

Introduction to database Systems, advantages of database system over traditional file system, Basic concepts & Definitions, Database users, Database Language, Database System Architecture, Schemas, Sub Schemas, & Instances, database constraints, 3-level database architecture, Data Abstraction, Data Independence, Mappings, Structure, Components & functions of DBMS, Data models.

UNIT:2

(13 Hours)

Entity relationship model, Components of ER model, Mapping E-R model to Relational schema, Relational Algebra, Tuple & Domain Relational Calculus, Relational Query Languages: SQL and QBE. Database Design:- Database development life cycle (DDLC), Automated design tools, Functional dependency and Decomposition, Join strategies, Dependency Preservation & lossless Design, Normalization, Normal forms: 1NF, 2NF, 3NF, and BCNF, Multi-valued Dependencies, 4NF & 5NF. Query processing and optimization: Evaluation of Relational Algebra Expressions, Query optimization, Query cost estimation.

SYLLABUS

UNIT:3

(10 Hours)

Network and Object Oriented Data models, Storage Strategies: Detailed Storage Architecture, Storing Data, Magnetic Disk, RAID, Other Disks, Magnetic Tape, Storage Access, File & Record Organization, File Organizations & Indexes, Order Indices, B+ Tree Index Files, Hashing Data Dictionary.

UNIT:4

(12 Hours)

Transaction processing and concurrency control: Transaction concepts, properties of transaction, concurrency control, locking and Timestamp methods for concurrency control schemes. Database Recovery System, Types of Data Base failure & Types of Database Recovery, Recovery techniques, fundamental concepts on Object-Oriented Database, Object relational database, distributed database, Parallel Database, introduction to Data warehousing & Data Mining.

OUTLINE

- Introduction to database Systems
- Advantages of database system over traditional file system
- Database users
- Database Language
- Database System Architecture
- Schemas, Sub Schemas, & Instances, database constraints
- 3-level database architecture
- Data models.

INTRODUCTION

Data: data means a known fact that can be recorded and that have some implicit meaning. E.g., the names, telephone number, address of people etc.

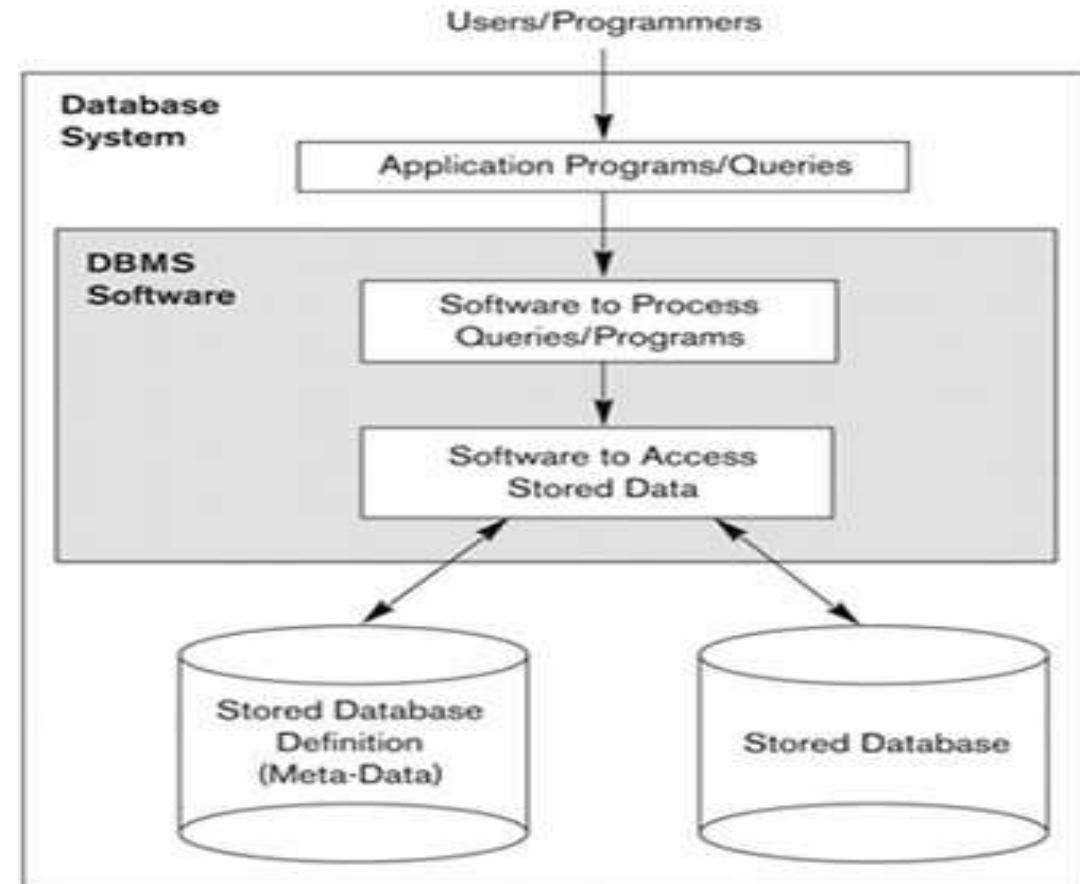
Database: a database is a collection of inter-related data with an implicit meaning.

DBMS: stands for Database Management System and it consist of 2 major components.

- The collection of inter related data, which is called as the database.
- A set of software packages or a set of software tools or programs that can access the data and process the data.

INTRODUCTION

Therefore the primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.



INTRODUCTION

1. Differentiate between **data** and **information**.
2. A **DBMS** is a _____.
 - a. Hardware
 - b. Software
 - c. Language
3. Differentiate between **Database** and **DBMS**.
4. Give some examples of **DBMS**.

INTRODUCTION

1. Differentiate between *data* and *information*.

When data are processed, interpreted, organized, structured or presented so as to make them meaningful or useful, they are called information. Information provides context for data. For example, a list of dates — data — is meaningless without the information that makes the dates relevant (dates of holiday).

2. A DBMS is a _____.

- a. Hardware
- b. Software
- c. Language

3. Differentiate between *Database* and *DBMS*.

A database is any collection of data, it can be words you write on a piece of paper or a digital file. A Database Management System (DBMS) is a piece of software that manages databases and lets you create, edit and delete databases, their tables and their data.

4. Give some examples of *DBMS*.

The DBMS manages incoming data, organizes it, and provides ways for the data to be modified or extracted by users or other programs. Some DBMS examples include MySQL, PostgreSQL, Microsoft Access, SQL Server, FileMaker, Oracle, RDBMS, dBASE, Clipper, and FoxPro.

DATABASE SYSTEM ENVIRONMENT:

The DBMS is a general purpose software system that facilitates the process of defining, constructing, manipulating and sharing databases among various users and applications.

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

Constructing: the database follows the process of storing the data in some storage medium controlled by the DBMS.

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

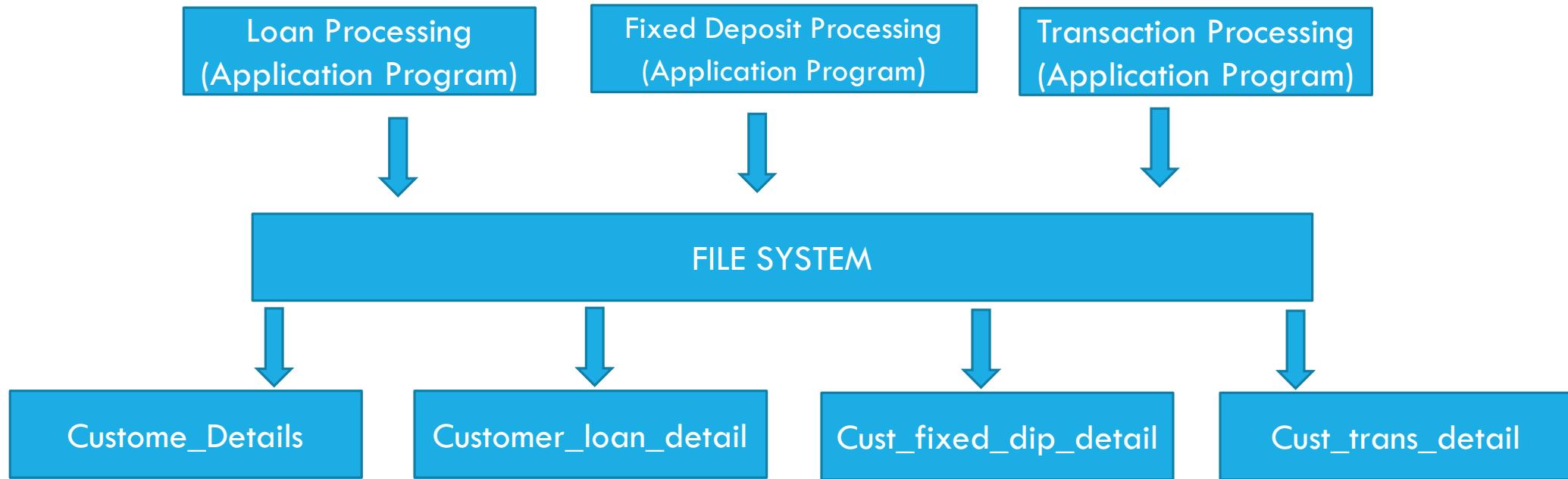
DATABASE SYSTEM ENVIRONMENT:

Sharing: a database allows multiple users and programs to access the database concurrently.

Protection: includes both system protection against hardware or software malfunction / crashes and security protection against unauthorized access.

Maintain: the database system by allowing the system to evolve as requirements change over time.

Traditional Method of Data Storage



- ❖ In the traditional approach, information is stored in flat files which are maintained by the file system of OS.
- ❖ Application programs go through the file system to access these flat files.

Traditional Approach: (Challenges)

Data Security: The data as maintained in the flat file(s) is easily accessible and therefore not secure.

Example: Consider the Banking System. The Customer transaction file has details about the total available balance of all customers. A Customer wants information about his account balance. In a file system it is difficult to give the Customer access to only his data. Thus enforcing security constraints for the entire file or for certain data items are difficult.

Data Redundancy: Often the same information is duplicated in two or more files.

Example: Assume the same data is repeated in two or more files. If change is made to data in one file, it is required that the change be made to the data in the other file as well. If this is not done, it will lead to error during access of the data.

Data Isolation: Data Isolation means that all the related data is not available in one file. Generally, the data is scattered in various files, and the files may be in different formats

Traditional Approach: (Challenges)

Program/Data Independence: Under the traditional file approach, application programs are dependent on the master and transaction file(s) and vice-versa. Changes in the physical format of the master file(s), such as addition of a data field requires that the change must be made in all the application programs that access the master file.

Lack of Flexibility: The traditional systems are able to retrieve information for predetermined requests for data.

Example: Consider the banking system. An application program is available to generate a list of customer names in a particular area of the city. The bank manager requires a list of customer names having an account balance greater than \$10,000.00 and residing in a particular area of the city. An application program for this purpose does not exist. The bank manager has two choices:

- To print the list of customer names in a particular area of the city and then manually find out those with an account balance greater than \$10,000.00
- Hire an application programmer to write the application program for the same. Both the solutions are cumbersome.

Traditional Approach: (Challenges)

Concurrent Access Anomalies: Many traditional systems allow multiple users to access and update the same piece of data simultaneously. But the interaction of concurrent updates may result in inconsistent data.

Database approach:

In the database approach a single repository of data is maintained that is defined once and is accessed by various users in many ways.

Characteristics: The following are the main characteristics of the database approach.

Self describing nature of a database system: The database system contains not only the database itself but also a complete definition / description of the database structure and constraints. The descriptions include the structure of each file, its type and storage format of each data item and various constraints imposed on them. This information is called as *metadata* and it is stored in the DBMS catalog.

Insulation between program and data, and data abstraction: in the DBMS approach if we change the structure of a file then it will not require to change all programs that access the file as the structure of the data files are stored in the DBMS catalog separately from the access programs. This property is called as program–data independence.

Database approach:

- **Data abstraction:** the characteristic that allows program – data independence and program – operation independence is called as data abstraction.
- **Support of multiple views of data:** different users of database 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.
- **Sharing of data and multiuser transaction processing:**
 - The DBMS must include **concurrency control software** to ensure that several users trying to update the same data must be done in a controlled manner so that the result of the updates is correct.
 - A **transaction** is an executing program or process that includes one or more database records.

Advantages of DBMS:

The following are the advantages of DBMS:

- Controlling redundancy:

- The DBMS avoids unnecessary duplication of data and effectively reduces the total amount of data storage required.
- It also eliminates the extra processing necessary to trace the required data in a large mass of data.
- Another advantage of avoiding duplication is the elimination of the inconsistencies that may be present in redundant data files.
- Any redundancies that exist in the DBMS are controlled and the system ensures that these multiple copies are consistent.

Advantages of DBMS:

Shared data: a database allows the sharing of data under its control by any number of application programs or users.

Restricting unauthorized access:

- The database administrator (DBA) who has the ultimate responsibility for the data in the DBMS can ensure that proper access procedures are followed including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data.
- Different levels of security can be implemented for the various types of data and operations.

Advantages of DBMS:

Providing backup and recovery: the backup and recovery subsystem of the DBMS is providing the facilities for recovering from hardware / software failures.

- For e.g., if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.

Integrity: data integrity means that the data contained in the database is both accurate and consistent. Therefore data values being entered for storage could be checked to ensure that they fall within a specified range and are of the correct format.

- For e.g., the value of the age of an employee may be in the range of 18 and 60. Another type of constraint specifies the uniqueness of data item values such as “every student must have a unique value for roll number”.

Advantages of DBMS:

- **Providing storage structure for efficient query processing:** DBMS provides a specialized data structures to speed up disk search for the required data. Indexes that are used for disk search are typically based on the tree data structure or hash data structure.
 - The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structure.
 - Representing complex relationships among data: a DBMS must have the capability to represent a variety of complex relationships of data that are interrelated in many ways as well as to retrieve and update related data easily and efficiently.

Database Users:

Database users are those who really use and take the benefits of database. There will be different types of users depending on their need and way of accessing the database.

➤ **Application Programmers** - They are the developers who interact with the database by means of DML queries.

- These DML queries are written in the application programs like C, C++, JAVA, Pascal etc.
- These queries are converted into object code to communicate with the database. (For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database.)
- It will include a embedded SQL query in the C Program.

➤ **Sophisticated Users:** They are database developers, who write SQL queries to select/insert/delete/update data.

- These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.

Database Users:

Specialized Users - These are also sophisticated users, but they write special database application programs. They are the developers who develop *the complex programs to the requirement.*

Stand-alone Users - These users will have *stand-alone database* for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.

Naive users - Any user who *does not have any knowledge about database* can be in this category. Their task is to just use the developed application and get the desired results. For example: Clerical staff in any bank is a naive user. They don't have any DBMS knowledge but they still use the database and perform their given task.

Database Users:

Database Administrators: The administration and maintenance of database is taken care by database Administrator—DBA. A good performing database is in the hands of DBA. A DBA has many responsibilities:

- **Installing and upgrading the DBMS Servers:** -
- Responsible for Installing a new DBMS server for the new projects.
- Responsible for Upgrading these servers as there are new versions comes in the market or requirement.
- If there is any failure in up gradation of the existing servers, he should be able revert the new changes back to the older version.
- Responsible for updating the service packs/ hot fixes/ patches to the DBMS servers.
- **Design and implementation:** - He should be able to decide proper memory management, file organizations, error handling, log maintenance etc for the database.

Database Users:

Performance tuning: - Since database is huge and it will have lots of tables, data, constraints and indices, there will be variations in the performance from time to time.

It is responsibility of the DBA to tune the *database performance* to make sure all the queries and programs works in fraction of seconds.

Migrate database servers: - DBA has to make sure that *migration* happens without any failure, and there is no data loss.(Ex: Sometimes, users using oracle would like to shift to SQL server.)

Backup and Recovery: -Proper backup and recovery programs needs to be developed by DBA and has to be maintained by him. This is one of the main responsibilities of DBA.

Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.

Security: - DBA is responsible for creating various database users and roles, and giving them different levels of access rights.

Documentation: - DBA should be properly documenting all his activities so that the database could be understood in his absence without any effort.

He should basically maintain reports of all his installation, backup, recovery, security methods and performance.

Introduction: Earlier architectures used mainframe computers to provide the main function for all functions of the system, including user application programs as well as all the DBMS functionality.

- **Data models:**

- A *data model* is a collection of concepts that can be used to describe the structure of a database where by structure of database, we mean the data types relationships and constraints that should hold for the data.
- Most data models also *include a set of basic operation or basic data model operation such as insert, delete, modify or retrieve any kind of objects.*

Categories of data models::

The data model is organized according to the types of concepts they use to describe the database structure.

Logical / conceptual / high level data model: it provides the concepts that are close to the way many users perceive data. It describes what data are stored in the database and what relationship exists among the data. It uses the concept such as entities, attributes and relationships.

Entity: it represents a real world object / concept such as an employee, student or project....

Attribute: it represents some properties that describe the entity such as employee's name, mobile etc...

Relationship: a relationship is the association among two or more entities.

Physical / low level data model: It provides the concept that describes the details of how data is stored in the computer.

Representational / Implementation data model: between the above two data model, this level provides concepts that may be understood by end users but are not too far removed from the way data is organized within the computer.

Categories of data models::

Data abstraction: It provides user a conceptual representation with an abstract view of the data.
(The system hides certain details of how data are stored and maintained that are not needed by most database users.)

Database schema: the description of the database or the overall design of the database is called as the database schema, which is specified during database design and is not expected to change frequently.

Database state / snapshot / Instance: the data in the database at a particular moment in time is called as the database state.

Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.

Metadata: it is the data about data means the description of schema constructs and constraints. Metadata is the information such as the structure of each file, the type and storage format of each data item and various constraints on the data.

Three-schema architecture:

The three schema architecture can be defined at the following three levels:

Internal level: the internal level has an internal schema, which is using the physical data model and describes the complete details of physical data storage and access paths of the database.

Conceptual level: the conceptual level has a conceptual schema which hides the details of physical storage structure and concentrates on describing entities, data types, relationships, user operations and constraints.

External / view level: it 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 the user group.

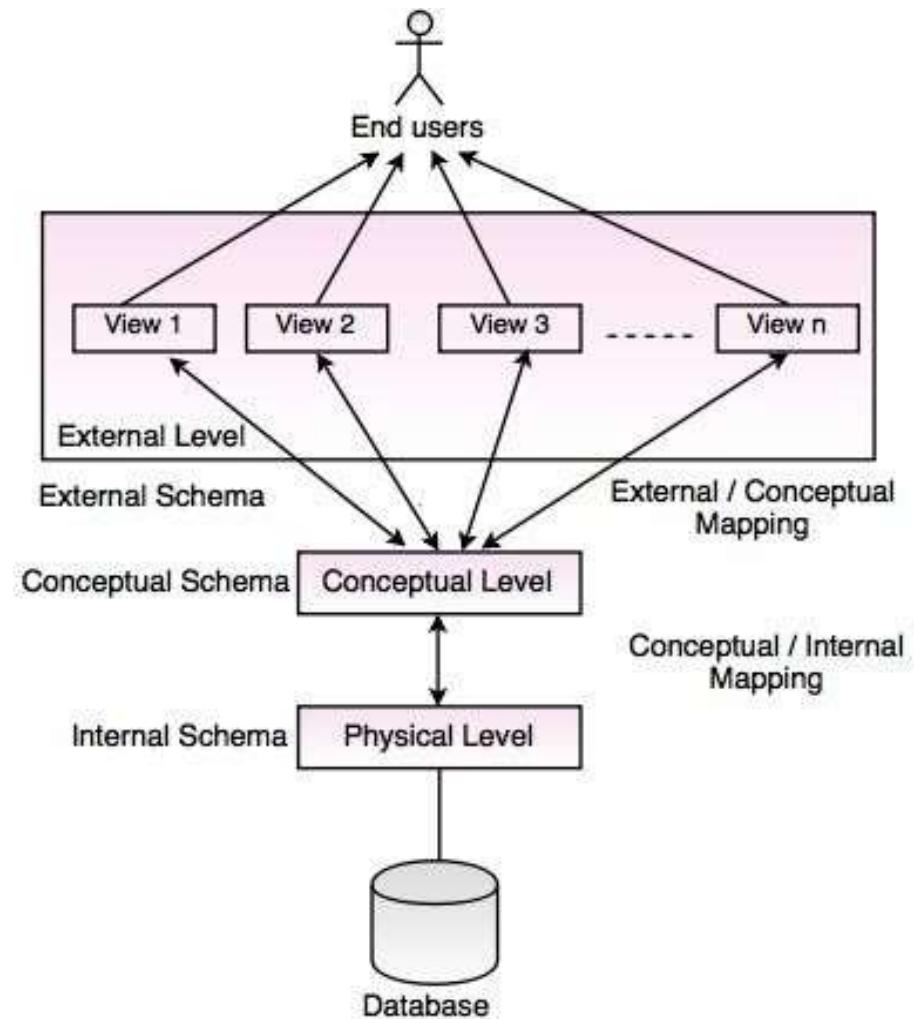
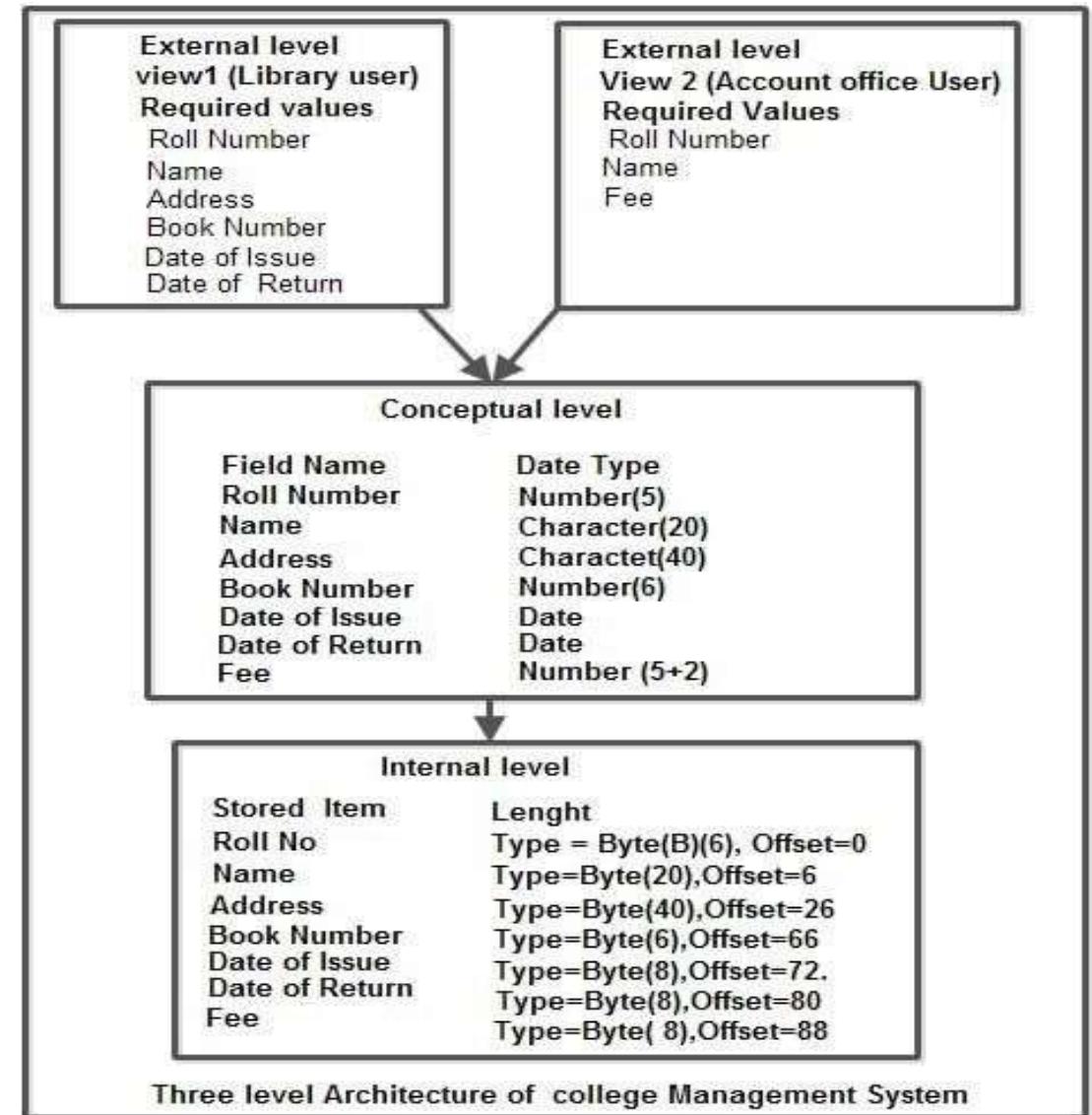


Fig. Three Level Architecture of DBMS

Three-schema architecture:

The DBMS must transform a request specified on an external schema into a request against the conceptual schema and then into a request on the internal schema for processing over the stored database and it must be reformatted to match the user's external view.



Three-schema architecture:

The process of transforming requests and result between levels are called mappings.

Data independence: It is the concept which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

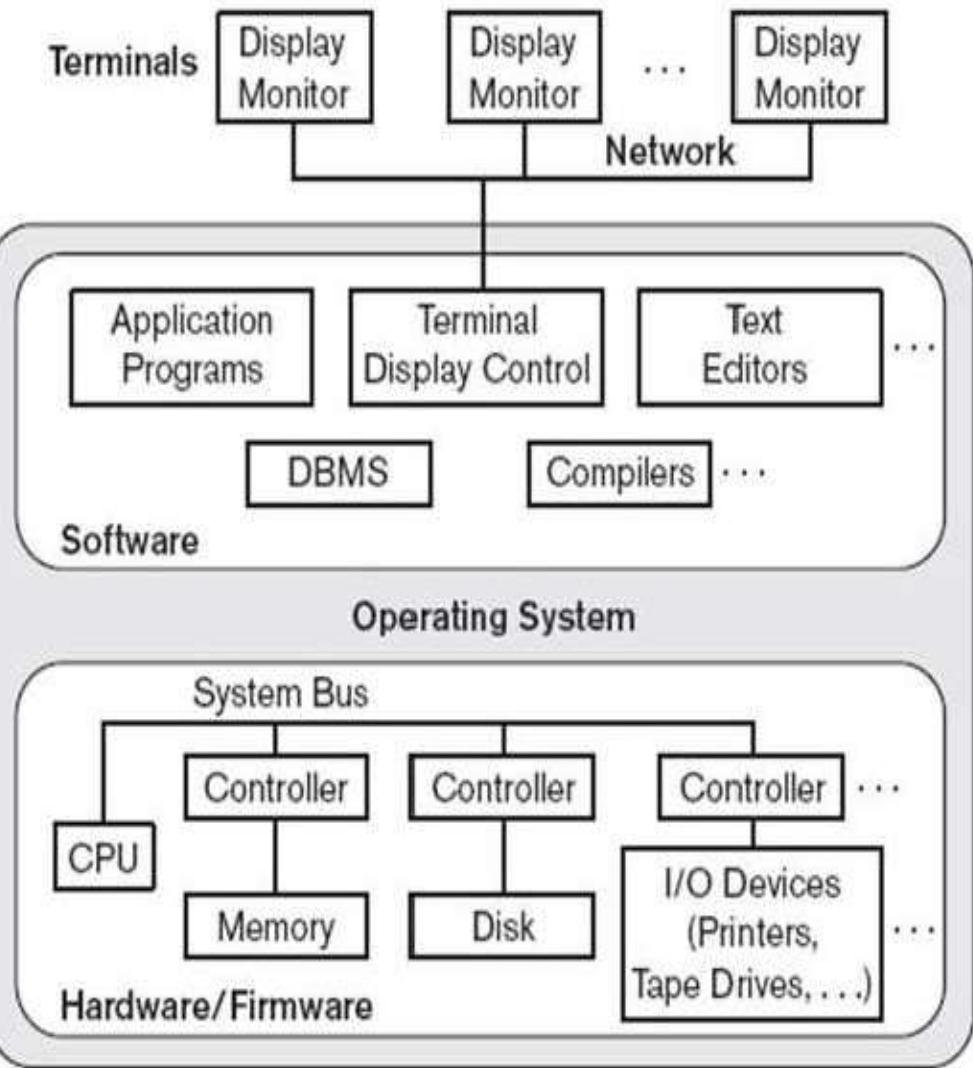
•**Logical data independence:** It is the capacity to change the conceptual schema without having to change the external schema or application programs. We may change the conceptual schema to expand the database by adding a data item, to change constraints or to reduce the database and that will not affect the external schema. Only the view definition and the mappings need to be changed.

•**Physical data independence:** It is the capacity to change the internal schema without having to change the conceptual schema. For e.g., the storage structure or devices used for storing the data could be changed without necessitating a change in the conceptual view / external view.

DBMS architecture:

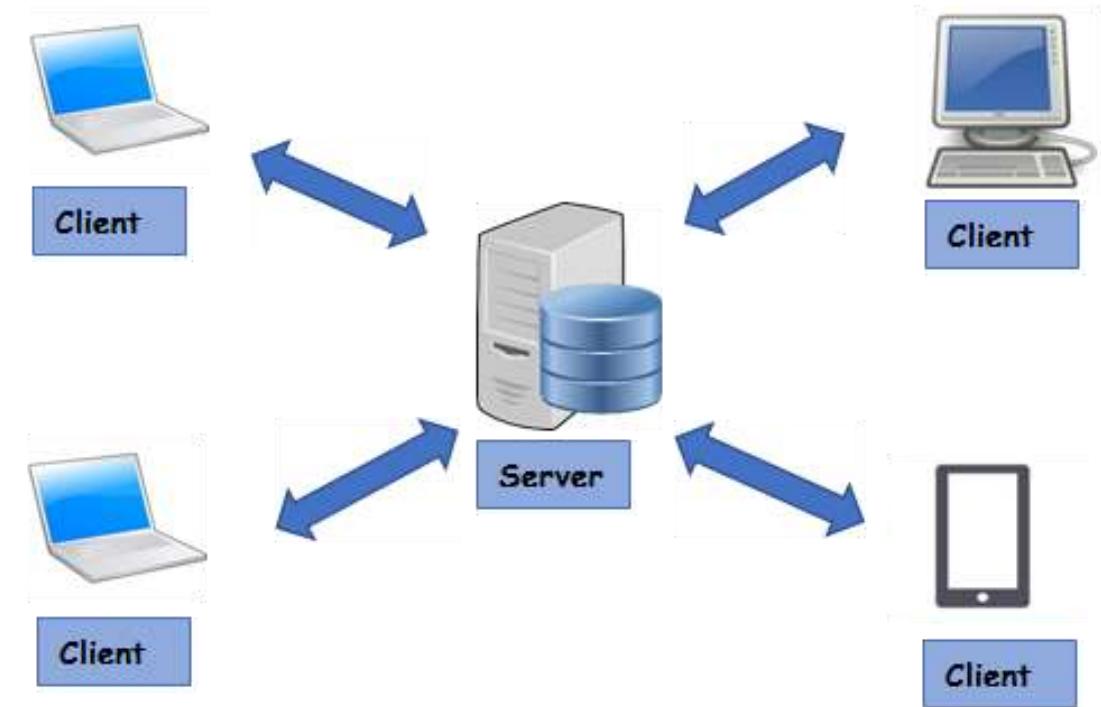
Earlier architectures used mainframe computers to provide the main processing for all functions of the system, including user application programs and user interface programs as well as the DBMS functionalities.

All the processing was performed remotely on the computer system and only display information and controls were sent from the computer to the display terminals through network.



Basic Client / Server Architecture:

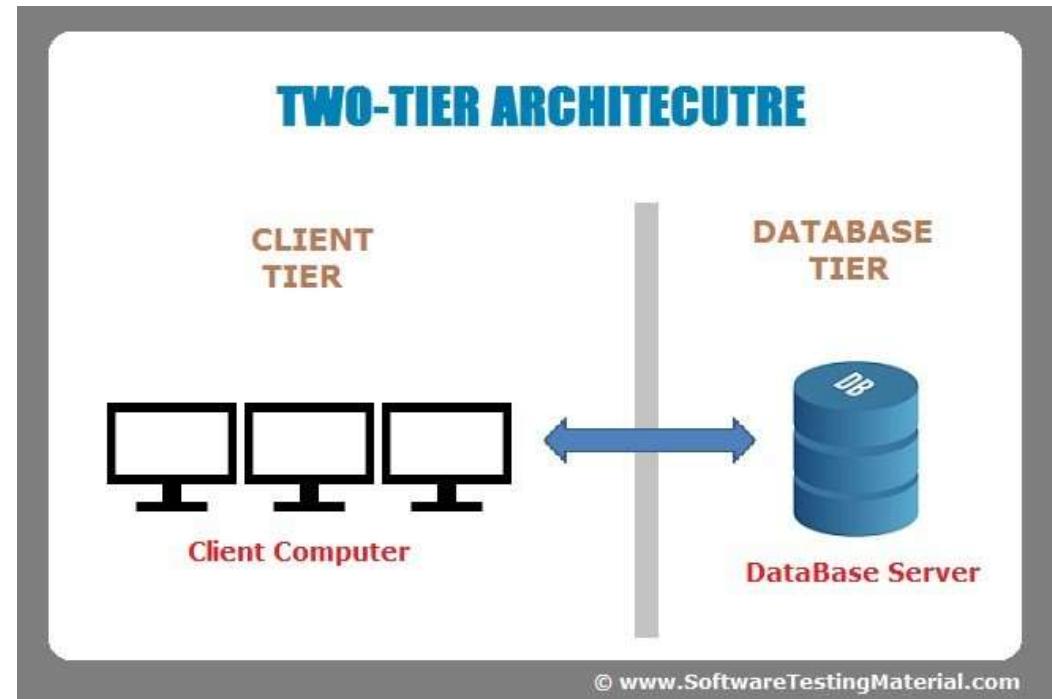
➤ The concept of *client / Server architecture* assumes a framework that consists of many PCs and workstations as well as a smaller number of mainframe computers connected via local area network and other types of networks. A client in this framework is typically a user machine that provides user interface capabilities and local processing.



➤ The *server* is a machine that can provide services to the client machines, such as file access, printing and database access etc...

Two tier Client / Server Architecture:

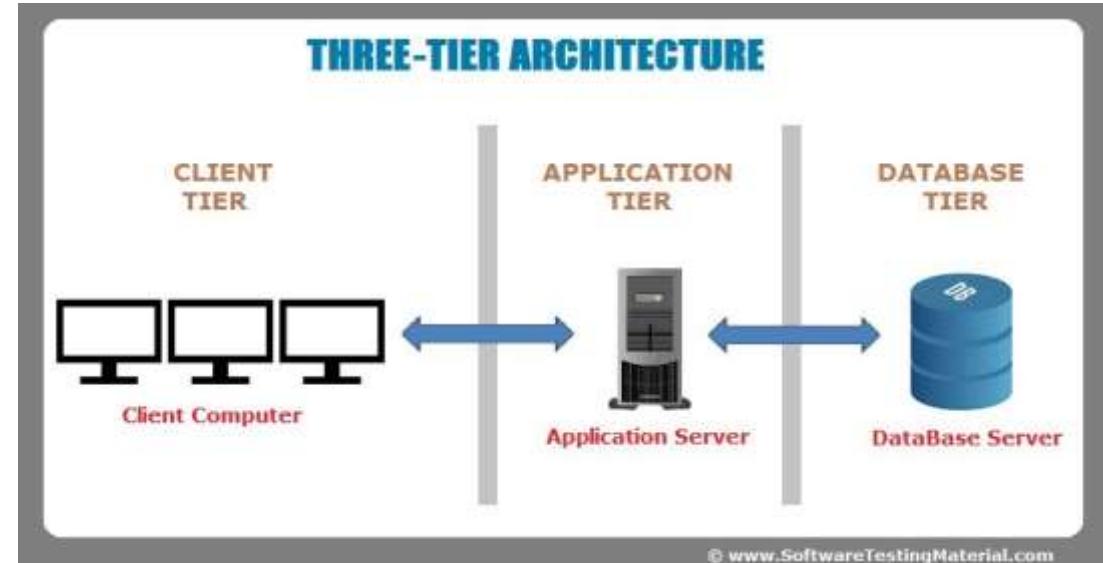
- It's called as two-tier architecture because the structure components are distributed over two systems client and server
- In this architecture the user interface program/application program can run on client machine.
- The query and transaction functionality remained on the server side.
- When DBMS access is required, the program establishes a connection to the DBMS .Once the connection is established ,the client program can communicate with the DBMS.
- A standard called open database connectivity(ODBC) provides an application programming interface(API),which allows the client side program to call the DBMS.



© www.SoftwareTestingMaterial.com

Three-tier Client/Server Architecture:-

- An intermediate layer called as *application server or web server* lies between the client and the database server.
- The *web server* play an intermediary role by storing business rules / procedures / constraints that are used to access the data from the database server.
- It can also *improves database security* by checking client's credentials before forwarding a request to the database server.
- The *web server* accepts request from the clients, process the request and sends database commands to the database server and then passing the processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format.



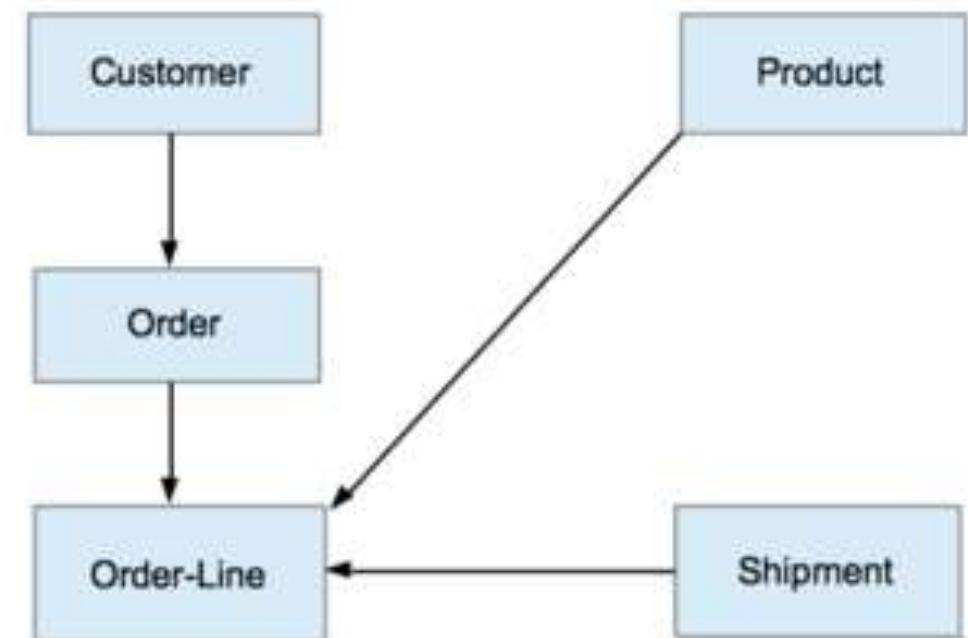
DATA MODELS: Hierarchical Model

- In the *hierarchical database approach/model* the data are in the hierarchical relationship.
- The data are stored in *paper and file*.
- Each customer section would contain *folders* for individual orders, and the orders would list each item being purchased.
- To store or retrieve *data*, the database system must start at top
- The *difficulty of searching* for item in the middle/bottom of the hierarchy.



NETWORK DATABASE APPROACH/MODELS:-

- The network model is named from the network of connection between the data elements.
- The primary goal is to solve the hierarchical problem of searching for data.
- The item are now physically separated, they are connected by arrows.
- The model solves the search problem but the cost is high.



Network Data Model

Object oriented database model :-

- An object oriented database model is a new and evolving method of organizing data.
- An object has 3 major components:-
 - A name
 - A set of properties/attributes
 - A set of methods/functions
- The properties describe the object just as attribute describe an entity in the relational database.
- Methods are short programs that define the actions that each can take.

For example, the code to add a new customer would be stored with the customer object.

Object-Oriented Model

Object 1: Maintenance Report	Object 1 Instance
Date	01-12-01
Activity Code	24
Route No.	I-95
Daily Production	2.5
Equipment Hours	6.0
Labor Hours	6.0

→

Object 2: Maintenance Activity	
Activity Code	
Activity Name	
Production Unit	
Average Daily Production Rate	

Object Oriented Database Approach

Relational data model :-

- The relational data model was first introduced by Ted Codd of IBM Research in 1970.
- The basic building blocks in the model is *tables/relations*.
- The relational model represents the database as a collection of relations, and each row in relations represents a collection of related data values.
- In the relational model, each row in the table represents a fact that typically corresponds to a real world entity/relationships.
- In the formal relationship relational model terminology, a row is called as tuple, a column header is called as attribute and the table is called as relation.
- The data types describing the type of data of values that can appear in each column is represented by a domain of possible values.
- Informally, a table is an entity set and a row is an entity.

Table also called Relation

The diagram illustrates a relational table with the following structure:

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

Annotations explain the components:

- Primary Key:** CustomerID
- Domain:** E.g. NOT NULL
- © guru99.com**
- Tuple OR Row:** Points to the rows (1, Google, Active), (2, Amazon, Active), and (3, Apple, Inactive).
- Column OR Attributes:** Points to CustomerID, CustomerName, and Status.
- Total # of rows is Cardinality:** Total # of rows is 3.
- Total # of column is Degree:** Total # of columns is 3.
- Relational data model:** Overall label for the table structure.

The Entity-Relationship(ER) Model:-

- The database is used to store information that is useful for an organization and represents this information through some means of modeling.
- The ER model was developed to facilitate the database design that represents the overall logical structure of a database.
- The ER model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.
- The database structure, employing the ER model is usually shown pictorially using entity-relationship(ER)diagram ,which shown how the schema for the database application can be displayed by means of the graphical notation .
- The ER data model employs 3 basic notations:
 - Entity sets
 - relationship sets
 - Attributes.

Entity sets:-

- An entity is a “thing” or “object” in the real world that is distinguished from all other objects. For example, each student in the student database is an entity.
- An entity has a set of properties and the values of the set of properties may uniquely identify an entity.
 - For example , a student may have a roll number property whose value uniquely identifies that student.
- Entity set:- An entity set is a set of entities of the same type that share the same properties, or attributes.
 - For example the set of all students in the student database can be defined as an entity set student.
- A database thus includes a collection of entity sets, each of which contains any number of entities of the same type.

Attributes:-

- An entity is represented by a set of attributes.
- Each entity has a value for each of its attributes. For example possible attributes for the student entity are roll-no, name, age etc.
- *Domain/value set*:- For each attribute, there is a set of permitted values, called the domain or value set, of that attribute.
 - For example , the domain of the attribute roll-no might be the set of all integers from 0 to 9.

Types of attributes:-

The following are the different types of attributes:-

Simple and composite attributes:-

The simple attributes couldn't be further divided into subparts.

For example: the roll number, age can not be divided into subparts, hence called as simple or atomic attributes.

The composite attributes can be divided into subparts and can form a hierarchy.

For ex: An attribute name could be structured as a composite attribute consisting of first_name, Middle_name and last_name as 3 sub-parts.

Single-valued and multi-valued attributes:-

Single-valued attributes have single value for a particular entity.

For example , the roll number attributes for a specific student refers to only one roll number.

Multi-valued attributes means an attribute has a set of values for a specific entity.

For example, a student may have zero , one or several phone numbers and different employees may have different number of phones. This type of attribute is said to be multi-valued.

Types of attributes:-

Stored and derived attributes:-

- **Stored attribute are the attributes from which we can derive another attribute.**
 - For example , the date_of_birth attribute from which we can derive the attribute age.

Derived attribute are the attributes that is derived from another attribute.

For example, the age attribute can be derived from the attribute date_of_birth.

Complex attributes:-

Composite and multi-valued attribute nested to form a complex attribute.

For example , if a person can have more than one residence and for each residence can have multiple phones, or attributes address phone can be specified as complex attribute.

Keys :-

- A key is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of a set.
- A key allows us to identify a set of attributes that is used to distinguish entities from each other.
- The following are the different types of keys used:-

Primary key:-

- It's a key attribute that uniquely identifies an entity within an entity set.

For ex: The attribute roll-number is unique and will identify an instance of the entity set STUDENT. Such an unique entity identifier as STUDENT roll-number is referred to as a **primary key**.

Candidate key:-

- There may be two or more attributes or combinations of attributes those uniquely identify an instance of entity set. These attributes or combinations of attributes are called as candidate keys.

For example ,the attributes roll-number and regd-no both can uniquely identifies an STUDENT entity.

Keys :-

Alternate key:-

- One of the candidate key can be used as primary key.
- The remaining candidate key could be considered alternate key.

Super key:-

- If we add/remove additional attributes to a primary key, the resulting combination would still uniquely identifies an instance of the entity set. Such keys are called as super keys.

For example , the attribute set {roll_no} is a primary key, that uniquely identifies an entity. However {roll_no, name, age} is a super key , because by removing name and age or both from the set still leaves with a super key.

Secondary key:-

- A secondary key is an attribute or combination of attributes that may not be a candidate key but that classifies the entity set on a particular characteristic.

For example, an entity set EMPLOYEE having the attribute department which identifies by its value all instances of EMPLOYEE who belong to a given department.

Foreign Key

Foreign key

- A Foreign Key is a set of attribute (s) whose values are required to match values of a column in the same or another table.

Foreign key

• Usually a foreign key is a “copy” of a primary key that has been exported from one relation into another to represent the existence of a relationship between them.

• Foreign key values do not (usually) have to be unique.

• Foreign keys can also be *null*.

DEPT

(Parent /Master/Referenced Table)

DeptNo	DName
D1	IVS
D2	ENR

EMP

(Child /Referencing Table)

EmpNo	EName	EDeptNo
1001	Elsa	D1
1002	John	D2
1003	Maria	Null
1004	Maida	D1

Points to remember

- Foreign key values do not (usually) have to be unique.
- Foreign keys can also be *null*.
- To enter the data in child table corresponding data must be present in master table or NULL is the default entry in child table in the referenced column (FK column)

QUIZ



QUIZ

Trainee(Empno, FirstName, LastName, Email, PhoneNo)

Assumptions:

- I. Empno for each trainee is different.
- II. Email for each trainee is different
- III. PhoneNo for each trainee is different
- IV. Combination of FirstName and LastName for each trainee is different

Candidate key:

{Empno},{Email},{PhoneNo},{FirstName,LastName}

Primary key:

{Empno}

Alternate Key:

{Email},{PhoneNo},{FirstName,LastName}

QUIZ

Given a relation R1(X,Y,Z,L) and the following attribute(s) can uniquely identify the records of relation R1.

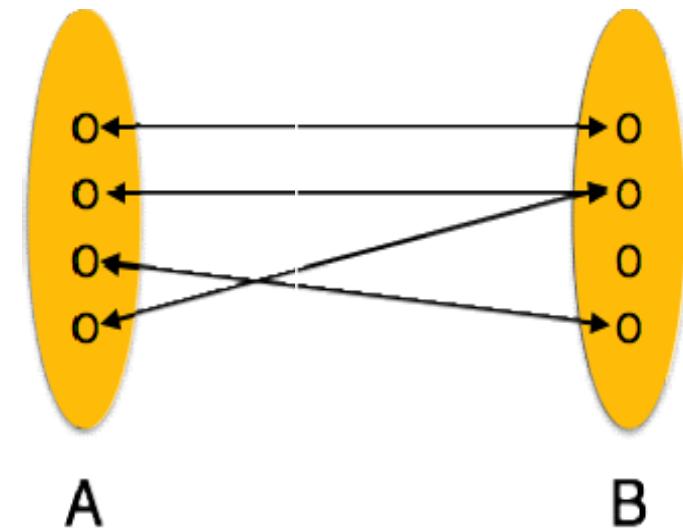
- 1)X
- 2)X,L
- 3)Z,L

Identify the following in relation R1?

- Candidate Key(s)
- Primary Key
- Alternate Key
- Key attribute(s)
- Non-key attribute(s)

Relationship sets:-

- A *relationship* is an association among several entities .Relationship type(R) between n entity. E1,E2,....En.
- A relationship set is a set of relationships of the same type. Formally, it is a mathematical notation of $n \geq 2$ entity sets.
 - For example ,for the two entity sets EMPLOYEE and DEPARTMENT , we can define the relationship type works-for, which associates each employee with the department for which the employee works.
- The association between the entity sets is referred to as *participation* i.e the entity sets EMPLOYEE , DEPARTMENT participate in the relationship set works_for.
 - Mathematically , the entity sets E1,E2,E3,....En participate in the relationship set R.



Relationship sets:-

- The *function* that an entity plays in a relationship is called the entity's *role*. The role name signifies the role that a participating entity from the entity type plays in each relationship instance ,and helps to explain what the relationship means.
- *Recursive relationship set*:- When the entity sets of a relationship set are not distinct, that is the same entity set participating in relationship set more than once in different roles. In this type of relationship set ,it is called as the recursive relationship set.
- *Degree of relationship*:- Degree of the relationship type is the name of participating entity type in the relationship.
- *Mapping cardinalities*-(*Cardinality ratios*): Mapping cardinality express the number of entity to which another entity can express the number via a relationship set.
 - For example , a binary relationship set R between entity sets A and B , the mapping cardinalities must be one of the following:-

Relationship sets:-

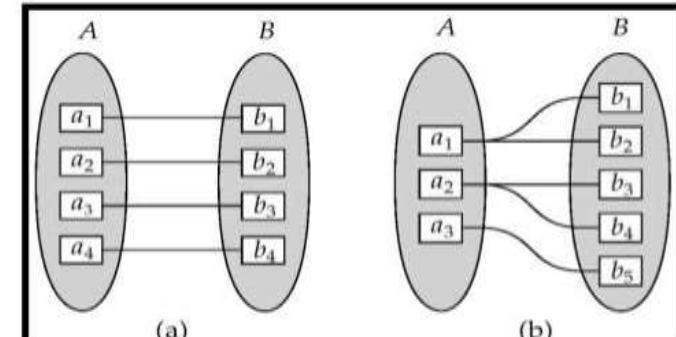
One-to-One(1:1): An entity associated in A is with at most one entity in B, and an entity in B is associated with at most one entity in A.

One-to-Many: An entity in A is associated with any number (zero or more) of entity in B, An entity in B ,however can be associated with at most one entity in A.

Many-to-One: An entity in A is associated with at most one entity in B. An entity in B , however can be associated with any number of entities in A.

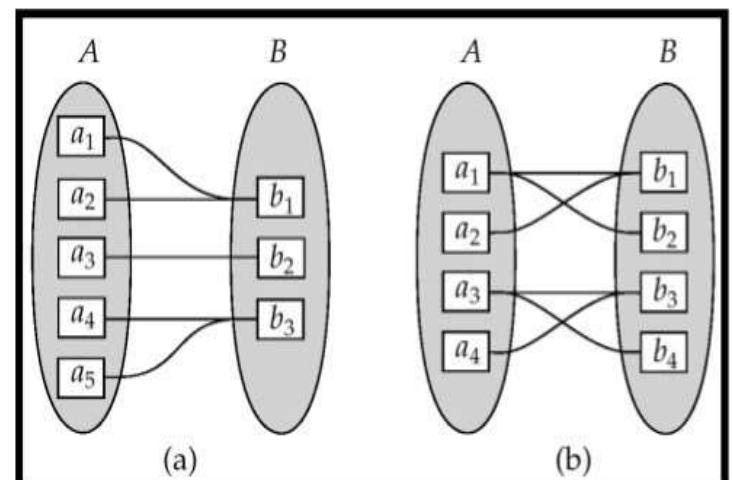
Many-to-Many: An entity in A is associated with any number(zero/more) of entities in B, and an entity in B is associated with any number (zero/more) of entities in A.

Mapping cardinalities



One to
One to many
Note: Some elements in A and B may not be mapped to any elements in the other set

15



Many to one
Many to many
Note: Some elements in A and B may not be mapped to any elements in the other set

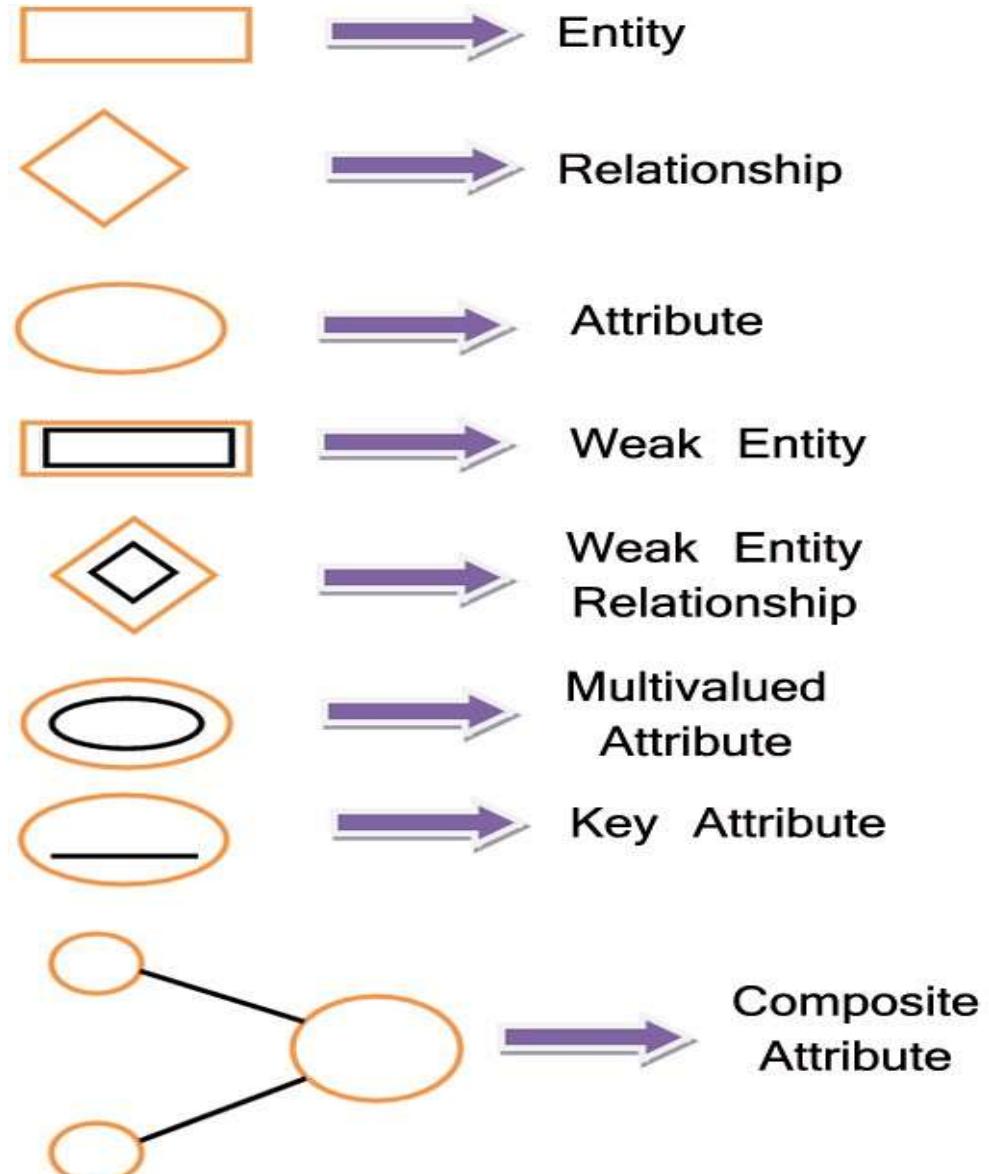
16

Participation Constraints:-

- The participation of an entity set E in a relationship set R is said to be *total* if every entity in E participates in at least one relationship in R.
- The participation of an entity set E in a relationship set R is said to be *partial* if only some in E participate in relationship in R.

Entity –Relationships Diagrams:-

- An ER diagram can express the overall logical structure of a database graphically. The following are the different notations/components that are used in the ER diagram.



Participation Constraints:-

Strong Entity Type and Weak Entity type:-

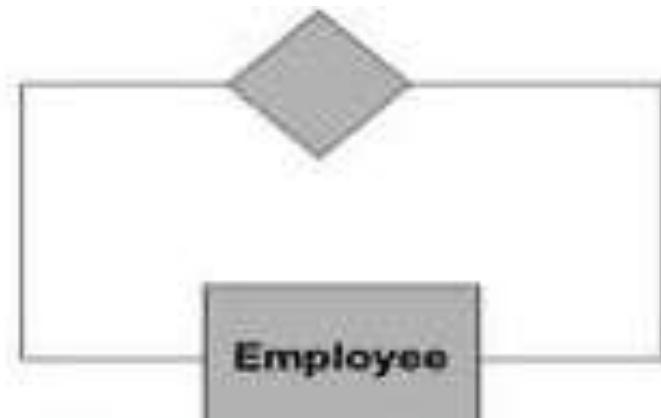
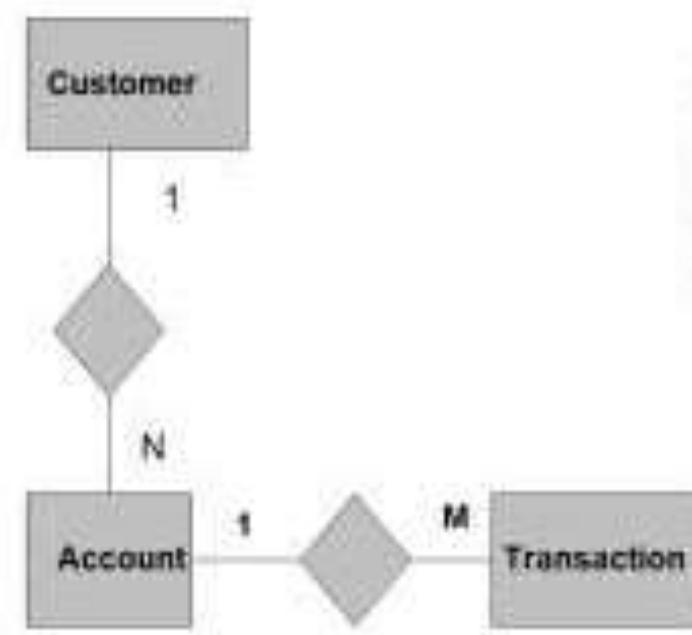
- A regular entity type or strong entity type does have sufficient attribute to form a key attribute to form a key attribute /primary key .
- A weak entity type do not have sufficient attribute to form a key attribute .
- Entities belonging to weak entity type are identified by being related to a another entity type known as owner entity type.
- A weak entity type always has a total participation with respect to its identification relationship, because a weak entity cannot be identified without an owner entity.
- For example ,consider the entity type **DEPENDENT**, related to **EMPLOYEE**, which is used to keep the details of the employee's department. The attributes of **DEPENDENT** are Name, DOB, Sex and relationship to the employee.

Participation Constraints:-

- A weak entity type (in our ex. DEPENDENT) has a partial key(suppose name) which is a set of attribute that can uniquely identify the weak entities with the help of the strong entity type(Ex. EMPLOYEE)
- In ER-diagram, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines.
- The partial key attribute in underline with a dashed or dotted lines.

ER Modeling Notations

Cardinality of Relationship tells how many instances of an Entity type is relate to one instance of another Entity type. M,N both represent “MANY ” and 1 represents “ONE” Cardinality.



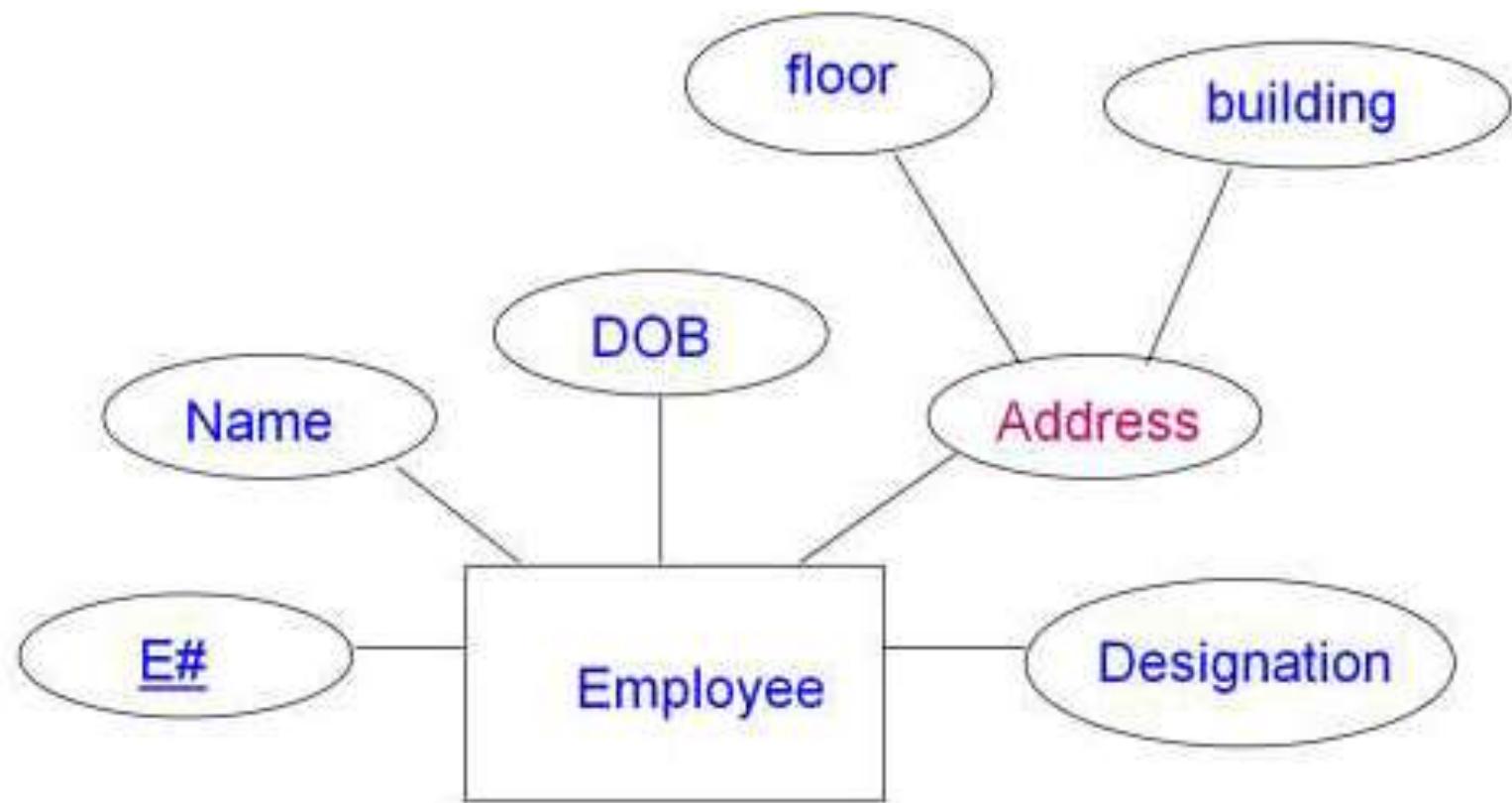
An Entity can be self linked.

For example, employees can supervise other employees

ER Modeling Notations : Composite Attribute

Represented by an ellipse from which other ellipses produced and represent the component attributes.

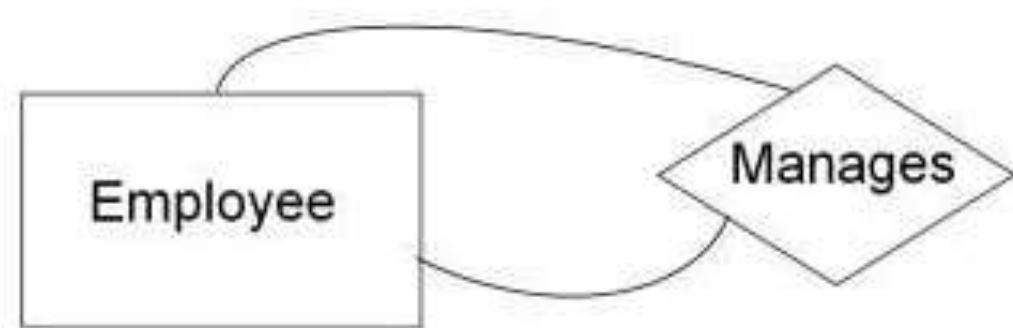
E.g Address



ER Modeling Notations :

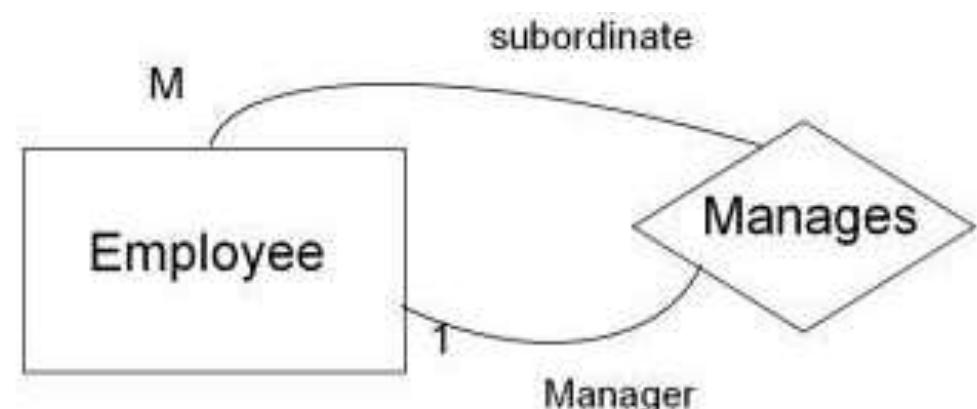
Unary Relationship:

- A unary relationship is represented as a diamond which connects one entity to itself as a loop.
- The relationship above means, some instances of employee manage other instances of Employee



Role Names:

- Role names may be added to make the meaning more explicit.



ER Modeling Notations :

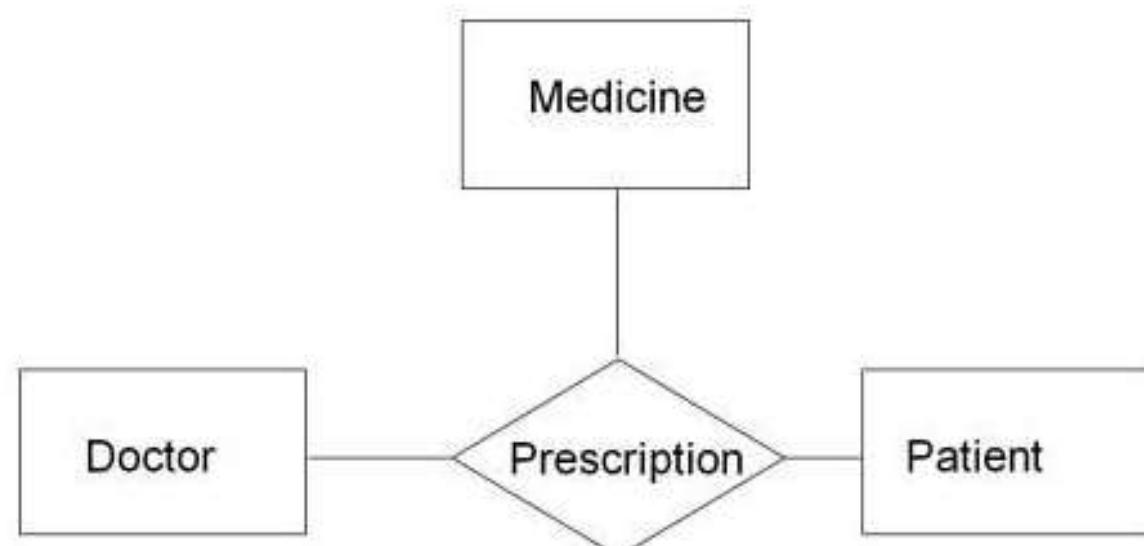
Binary Relationship:

- A relationship between two entity types



Ternary Relationship:

- A relationship connecting three entity types.

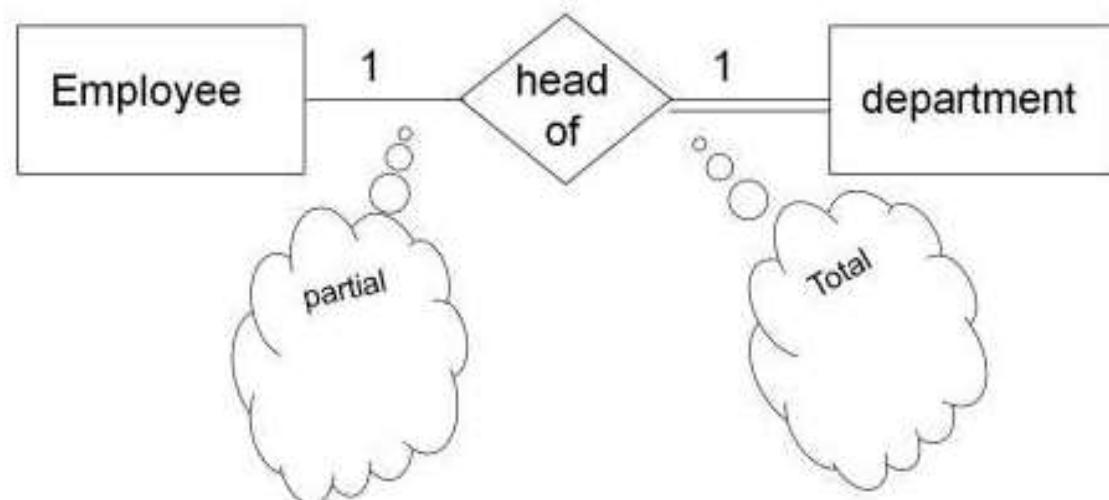


ER Modeling Notations :

Relationship Participation:

All instances of the entity type Employee don't participate in the relationship, Head-of.

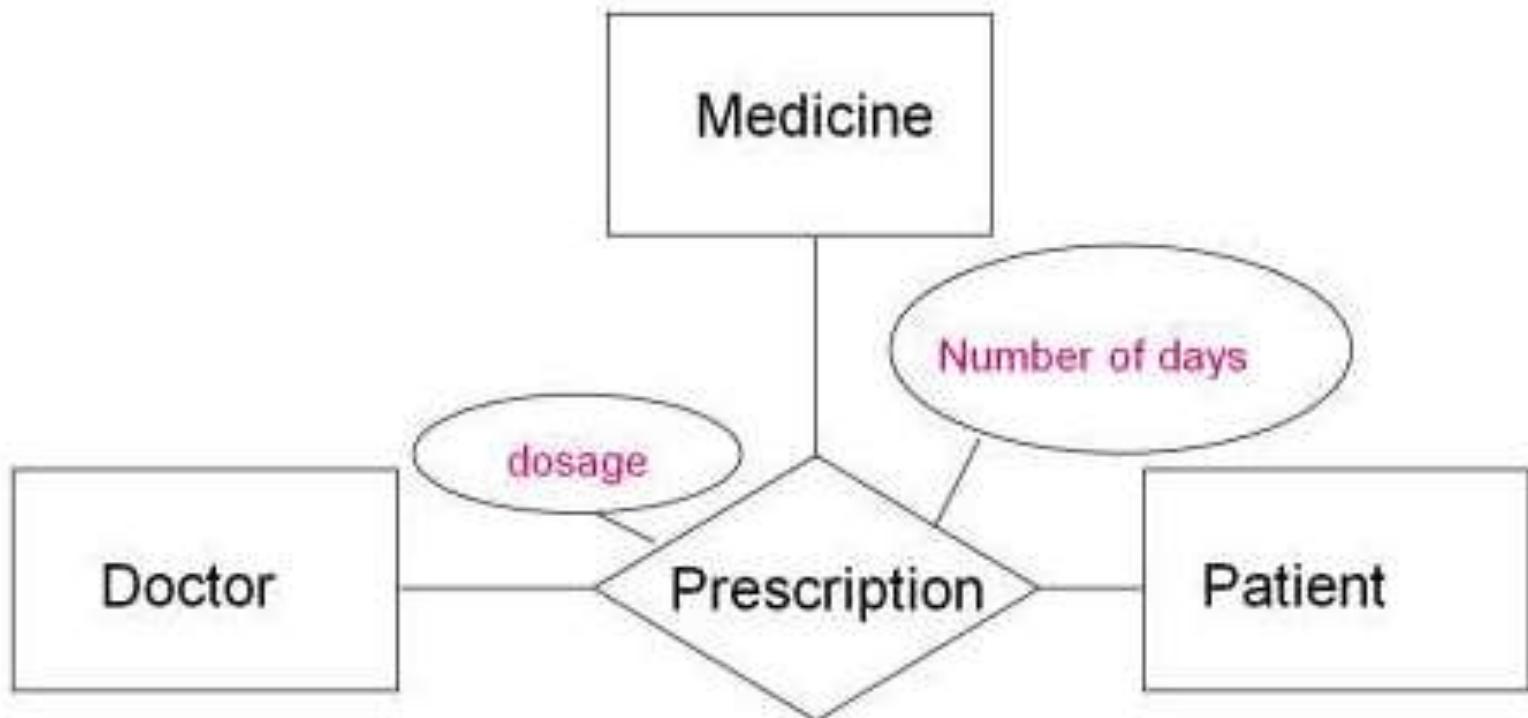
- Every employee doesn't head a department. So, employee entity type is said to partially participate in the relationship.
- But, every department would be headed by some employee.
- So, all instances of the entity type Department participate in this relationship. So, we say that it is total participation from the department side.



ER Modeling Notations :

Attributes of a Relationship:

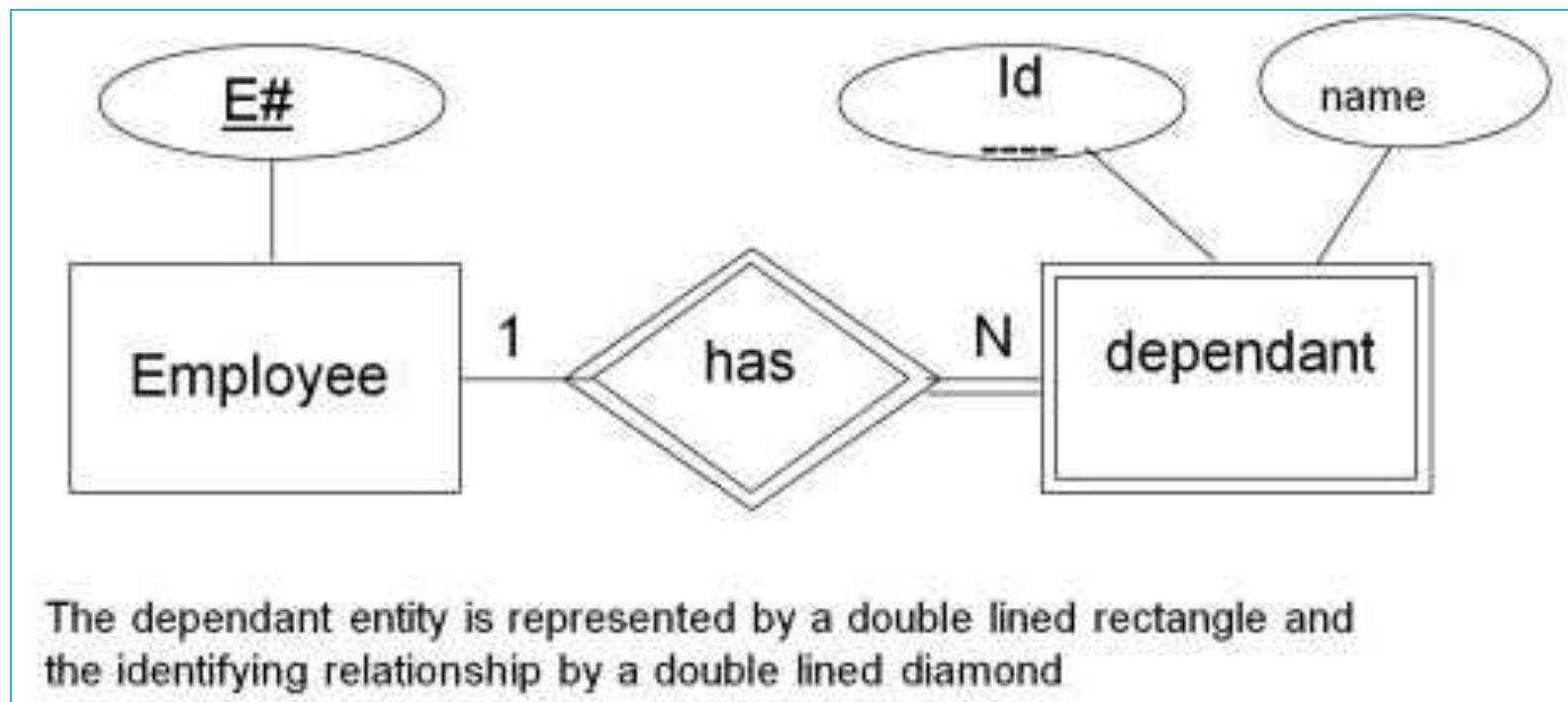
These attributes best describe the relationship prescription rather than any individual entity Doctor, Patient or Medicine.



ER Modeling Notations :

Weak Entity:

- The identifying relationship is the one which relates the weak entity (dependant) with the strong entity (Employee) on which it depends.
- Id is underlined with a dotted line because it is used to form composite key of dependent entity along with E#.



Case Studies (ER Model for a College DB)

- A College contains many departments.
- Each department can offer any number of courses
- Many Instructors can work in a department.
- An Instructor can work only in one department.
- For each department there is a head.
- An instructor can be head of only one department.
- Each Instructor can take any number of courses.
- A course can be taken only by one instructor.
- A student can enroll for any number of courses.
- Each course can have any number of students.

Steps in ER Modeling

- Identify the Entities
- Find Relationships
- Identify the key attributes for every entities
- Identify other relevant attributes
- Draw complete ER diagram with all attributes including key attributes
- Review your results with your business users

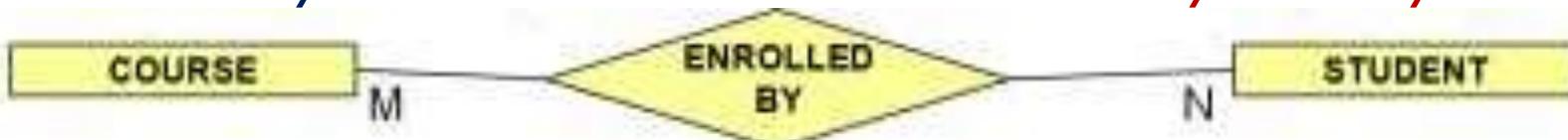
Steps in ER Modeling

➤ Identify the Entities

- *Department*
- *Student*
- *Course*
- *Instructor*

Steps in ER Modeling (Find the Relationships)

- One course is enrolled by multiple users and one student enrolls for multiple courses, hence the cardinality between course and student is **Many to Many**.



- The department offers many courses and each course belongs to only one department. Hence the cardinality between the department and course is **One to Many**.



- One department has multiple instructors and one instructor belongs to only one department. Hence the cardinality between the department and Instructor is **One to Many**.



Steps in ER Modeling (Find the Relationships)

- In each department there is a “Head of department” and one instructor is “Head of department”. Hence the cardinality is **One to One**



- One course is taught by one instructor but the instructor teaches many courses. Hence the cardinality between course and instructor is **Many to One**.



Steps in ER Modeling

➤ Identify the Key attributes

- *Dept_name* is the key attribute for the Entity “**Department**”, as it identifies the department uniquely.
- *Course#(Course_Id)* is the key attribute for “**Course**” Entity.
- *Student#(Student_Number)* is the key attribute for “**Student**” Entity.
- *Instructor_Name* is the key attribute for “**Instructor**” Entity.

➤ Identify other relevant attributes

- For the department entity, the relevant attribute is *location*.
- For course Entity, the relevant attributes are *course_name*, *duration*, *Prerequisites*....
- For Instructor Entity, the relevant attributes are *room_no*, *telephone_no*....
- For Student Entity, the relevant attributes are *S_name*, *DOB*, *Email*.....

- Draw an ER diagram of Online Retail Application which allows customer to purchase items from a Retail shop.
- A customer can register to purchase an item. The customer will provide bank account number and bank name (the customer may have multiple account no).
- After registration each customer will have unique customer Id, user id and password.
- Customer can purchase one or more items in different quantities . The items can be of different classes based on their prices.
- Based on the quantity , price of item and discount(if any) on the purchased items, the bill will be generated. A bank account number is required to settle the bill.
- The application also mentions the information of suppliers who supply the items to the retail shop. The retail shop may give orders to supply the items based on some statistics they maintain about different items.

Case Studies (Online Retail Application)

- Draw an ER diagram of Online Retail Application which allows customer to purchase items from a Retail shop.
- A customer can register to purchase an item. The customer will provide bank account number and bank name (the customer may have multiple account no).
- After registration each customer will have unique customer Id, user id and password.
- Customer can purchase one or more items in different quantities . The items can be of different classes based on their prices.
- Based on the quantity , price of item and discount(if any) on the purchased items, the bill will be generated. A bank account number is required to settle the bill.
- The application also mentions the information of suppliers who supply the items to the retail shop. The retail shop may give orders to supply the items based on some statistics they maintain about different items.

Step 1: Identify the Entities

- CUSTOMER
- ITEM
- SUPPLIER
- BILL

Step 2: Find the relationships

- Customer can purchase an item and each purchase will be corresponding to a bill. So it is a ternary relationship.
- Items can be ordered to one or more suppliers. One supplier may take order of many items. So **many to many** relationship between **item** and **supplier**.
- One customer can pay many bill and one bill can be paid by only one customer. So **one to many** relationship between **customer** and **bill**.

Step 3: Identify the key attributes

- Customer entity will be identified by CustomerId
- Item entity will be identified by ItemId
- Supplier entity will be identified by SupplierId
- Bill entity will be identified by BillId

Case Studies (Online Retail Application)

Step 4: Identify other relevant attributes of Entities and Relationships

- For **Customer** entity the relevant attributes will be
(CustomerId, CustomerName, DateOfRegistration, UserId,
Password, AccountNo)
- For **Item** entity the relevant attributes will be
(ItemId, ItemName, UnitOfMeasurement, UnitPrice, Discount,
QuantityOnHand, SupplierId, ReOrderLevel, ReOrderQuantity, Class)
- For **Supplier** entity the relevant attributes will be
(SupplierID, SupplierName, SupplierContactNo)
- For **Bill** entity the relevant attributes will be
(BillId, AccountNo, BillAmount, BillDate)



Case Studies (Online Retail Application)

Step 4: Identify other relevant attributes of entities and Relationships (Cont..)

- For **Purchase** Relation the relevant attributes will be
(QtyPurchased, NetPrice)
- For **OrderedTo** relation the relevant attributes will be
(QtyOfOrder, OrderDate, DeliveryDate, DeliveryStatus)
- For **Pays** relation the relevant attributes will be
(AccountNo)

Converting ER Diagram to Relational Schema

Converting strong entity types:

- Each *Entity type* becomes a *table*
- Each *Single valued attribute* becomes a *column*
- *Derived attributes* are ignored
- *Composite attributes* are represented by its equivalent parts
- *Multi-valued attributes* are represented by a *separate table*
- The *key attribute* of the Entity type becomes the *primary key* of the table.

Entity Example

- Here address is a composite attribute
- Years of service is a derived attribute (can be calculated from date of joining and current date)
- Skill_Set is a multi-valued attribute

As per the rules:

- Derived attributes are ignored.
- Composite attributes are represented by components.
- Multi-valued attributes are represented by a separate table.



Employee(*E#*, *Name*, *Door_No*, *Street*, *City*, *Pincode*, *Date_of_joining*)

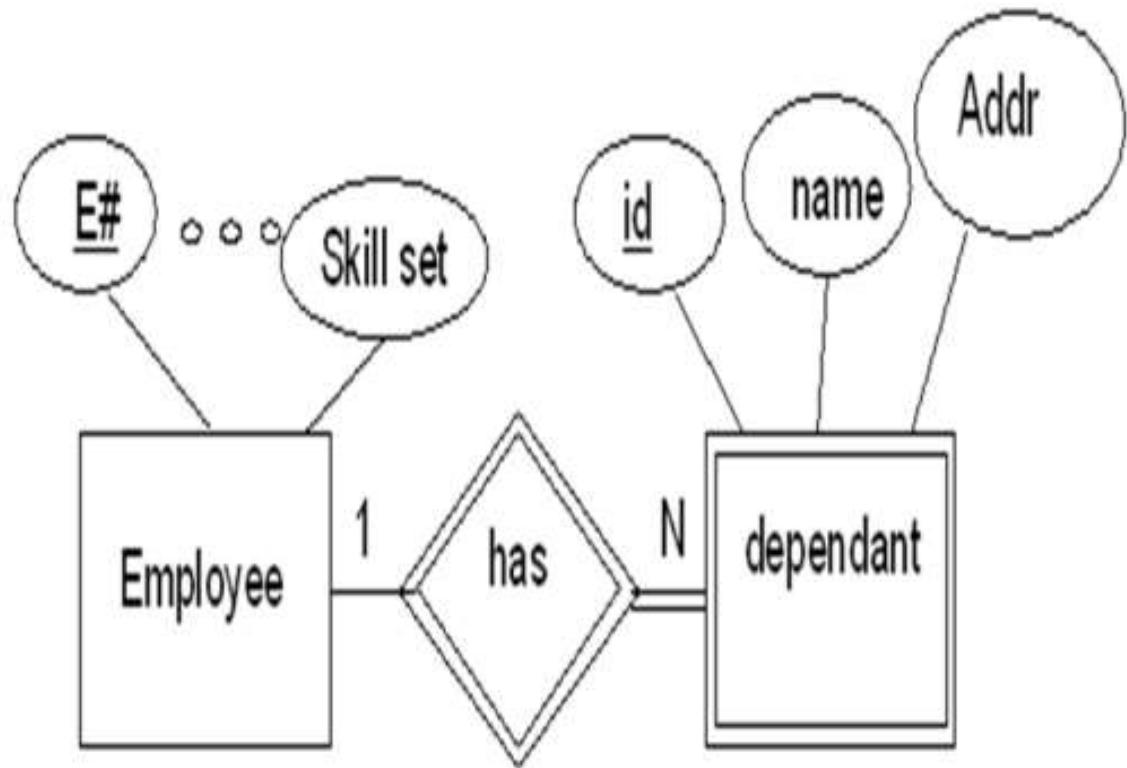
And

Emp_skillset(*E#*, *Skillset*)

Converting Weak Entity Types

- Weak Entity types are converted into a table of their own, with the primary key of the strong entity acting as a foreign key in the table.
- This foreign key along with the key of the weak entity forms the composite primary key of this weak table.

Here dependant is a **weak entity**. Dependant doesn't mean anything to the problem without the information on for which employee the person is a dependant.



The Relation Schema

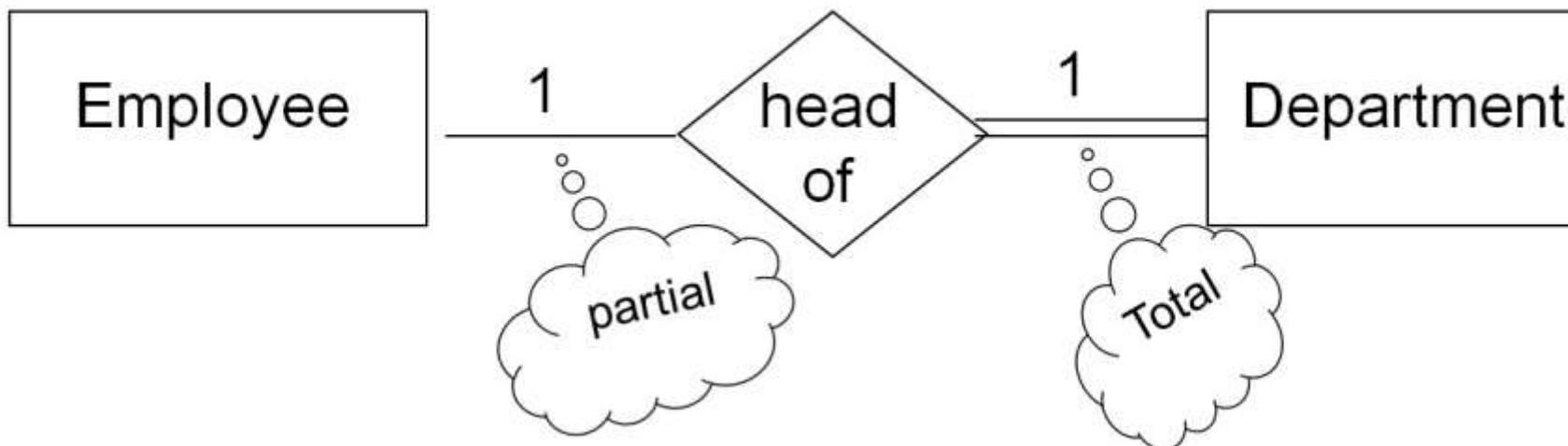
Employee(E#, EmpName, Date_of_joining, Skillset)

Dependent(Employee, Dependent_id, Name, Address)

Converting Relationship

- The way relationships are represented depends on the cardinality and the degree of the relationship.
- The possible cardinalities are
 - 1:1
 - 1:M
 - M:N
- The degrees are:
 - *Unary*
 - *Binary*
 - *Ternary*

Converting Relationship(Binary 1:1)



- Case:1 Combination of participation type
- *The primary key of the partial participant will become the foreign key in the total participant*

Employee(E#, EmpName, Date_of_joining, Skillset)

Department(Dept#, Dname, Location, Head)

Converting Relationship(Binary 1:1)



- Case 2: Uniform participation types
- The primary key of either of the participants can become a foreign key in the other

Employee (EmpCode, EmpName, DateOfJoining)

Chair (Item#, Model, Location, Used_by)

(OR)

Employee (EmpCode, EmpName, DateOfJoining, Sits_on)

Chair (Item#, Model, Location)

Converting Relationship(Binary 1:N)

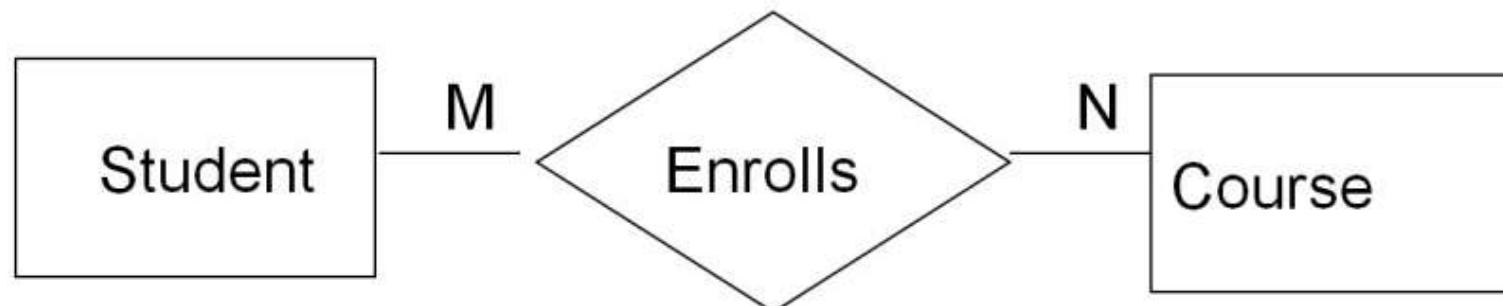


The primary key of the relation on the “1” side of the relationship becomes a foreign key in the relation on the “N” side.

Teacher (TeacherID, Name, Telephone, Cabin)

Subject (SubCode, SubName, Duration, TeacherID)

Converting Relationship(Binary M:N)



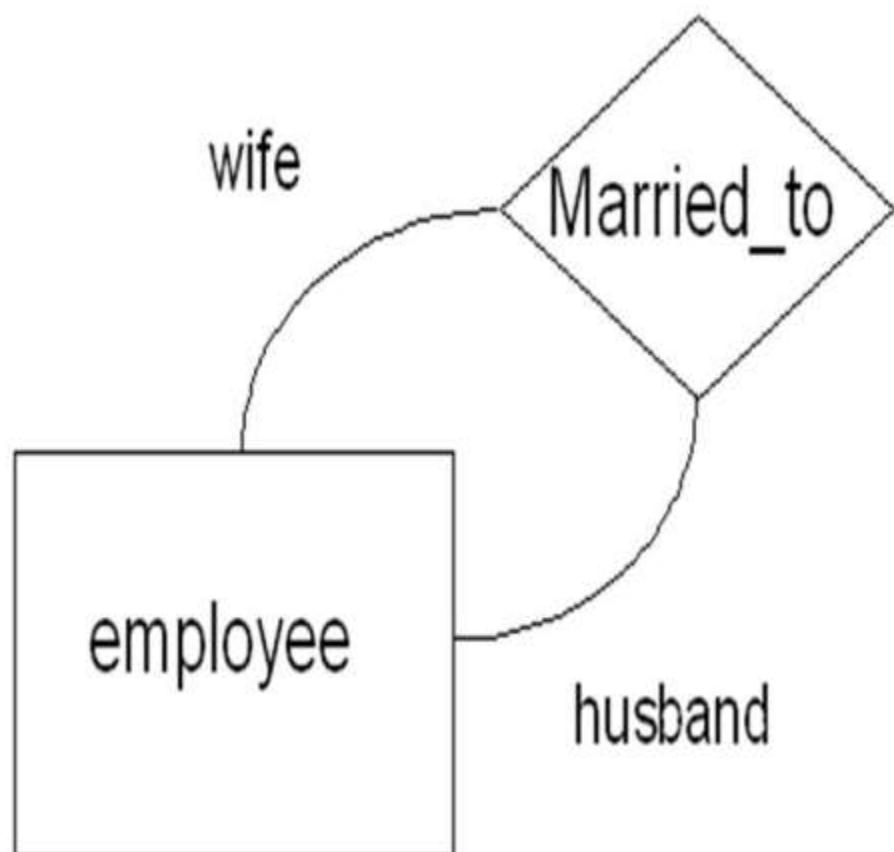
- A new table is created to represent the relationship which contains two foreign keys - one from each of the participants in the relationship.
- The primary key of the new table is the combination of the two foreign keys.
- Student (StudentID, SName, DOB, Address) Course(CourseID, CName)



Converting Relationship(Unary 1:1)

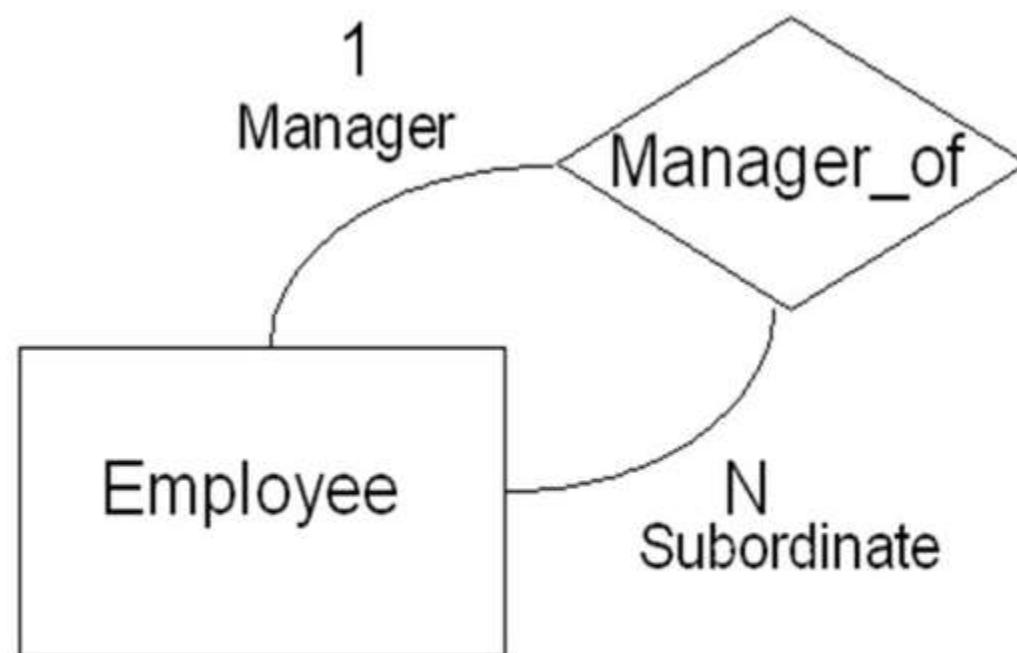
- Consider employees who are also a couple
- The primary key field itself will become foreign key in the same table

Employee(E#, EName, ... **Spouse**)



Converting Relationship(Unary 1:N)

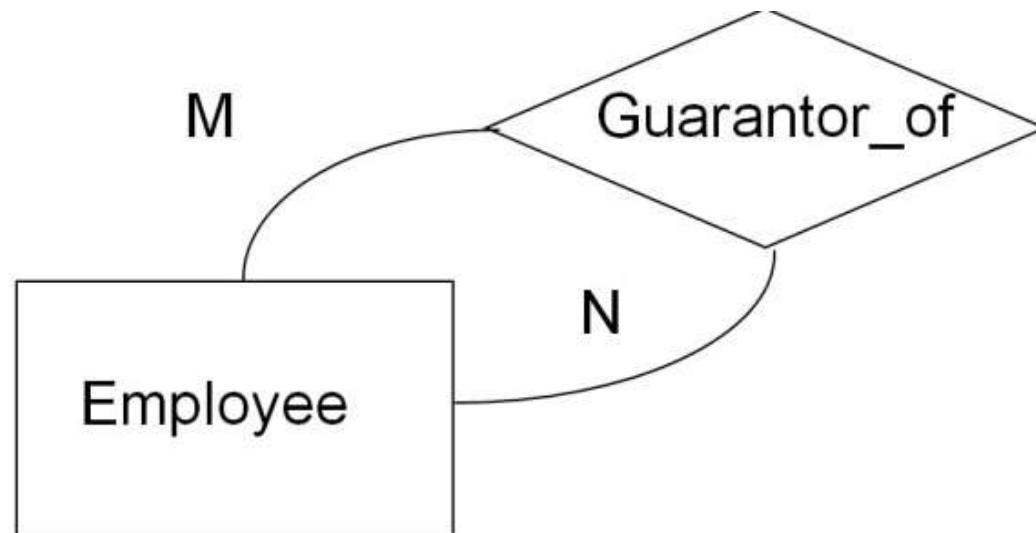
- The primary key field itself will become foreign key in the same table.
- Same as unary 1:1



Employee(E#, EName, DateOfJoining, SkillSet, Manager)



Converting Relationship(Unary M:N)



- There will be two resulting tables. One to represent the entity and another to represent the M:N relationship as follows

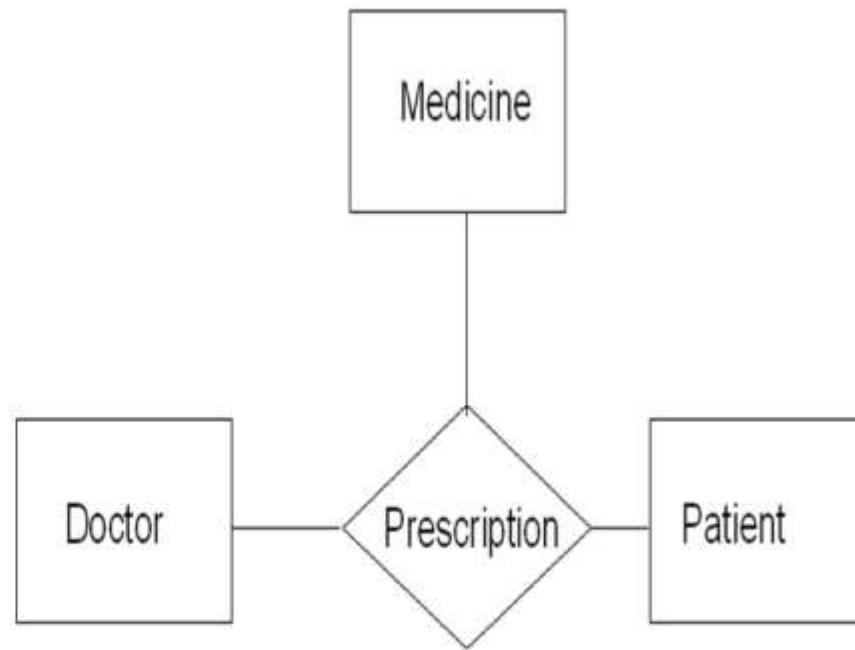
Employee(E#, EName, DateOfJoining, SkillSet)

Guaranty(Guarantor, Beneficiary)



Converting Ternary Relationship

- Represented by a new table.
- The new table contains **three** foreign keys - one from each of the participating Entities.
- The primary key of the new table is the combination of all three foreign keys.
- Prescription (DocID, PatCode, MedName)



Deriving logical schema for online retail application

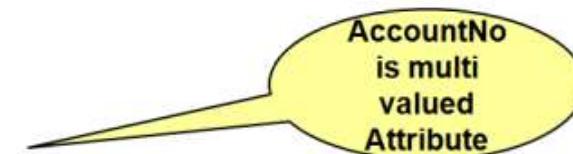
- There are four strong entities in the ER Diagram of Online Retail Application.
 - Customer
 - Item
 - Bill
 - Supplier
- One binary 1:N relationship
 - Pays
- One binary M:N relationship
 - OrderedTo
- One Ternary relationship
 - Purchase

Deriving logical schema for online retail application

So we can think of the following relations

Customer(CustomerId, CustomerName, DateOfRegistration, UserId, Password)

BankInfo(CustomerId, AccountNo)



- CustomerId is the Primary key of the Customer relation.
- (CustomerId, AccountNo) is the primary key of BankInfo relation
- CustomerId of BankInfo relation is foreign key and refers to CustomerId attribute of Customer relation.

For Many to many relationship and ternary relationship we create separate table otherwise the existing table (created for Entities) is modified to express the relationship.

Deriving logical schema for online retail application

Billing(BillId, BillDate, CustomerId, AccountNo)

- **BillId** is the Primary key of the Billing relation
- **(CustomerId, AccountNo)** are composite foreign key and refers (CustomerId, AccountNo) column of BankInfo relation.

Pay relationship is 1:N so no need to create separate relation

Item(ItemId, ItemName, QtyOnHand, UnitPrice, UnitOfMeasurement, ReOrderLevel, ReOrderQty, Discount)

ItemSupplier(ItemId, SupplierId)

SupplierId is multi valued in Item entity

- **ItemId** is the Primary key of the Item relation
- **(ItemId, SupplierId)** is primary key of ItemSupplier relation
- **ItemId** of ItemSupplier relation is foreign key to itemId of Item relation

For Many to many relationship and ternary relationship we create separate table otherwise the existing table (created for Entities) is modified to express the relationship.

Deriving logical schema for online retail application

Supplier(SupplierId, SupplierName,SupplierContactNo)

ItemOrdered(ItemId,SupplierId,QtyOrdered,DeliveryStatus,OrderDate, DeliveryDate)

OrderedTo
relationship
is M:N so
new relation
is required

- **SupplierId** is Primary key of Supplier Relation
- **(ItemId,SupplierId)** is composite primary key of ItemOrdered relation
- **ItemId** of ItemOrdered relation is foreign key to Item relation
- **SupplierId** of ItemOrdered relation is foreign key to Supplier relation

For Many to many relationship and ternary relationship we create separate table otherwise the existing table (created for Entities) is modified to express the relationship.

Deriving logical schema for online retail application

CustomerPurchase(CustomerId, ItemId, BillId, QtyPurchased, NetPrice)



Purchase relationship is ternary so new relation is required.

- (**CustomerId**,**ItemId**,**BillId**) are composite Primary key of CustomerPurchase relation
- **CustomerId** of CustomerPurchase relation is foreign key to Customer relation
- **ItemId** of CustomerPurchase relation is foreign key to Item relation
- **BillId** of CustomerPurchase relation is foreign key of Billing relation

For Many to many relationship and ternary relationship we create separate table otherwise the existing table (created for Entities) is modified to express the relationship.

Extended ER Modeling:-

- *ER modeling concepts are sufficient for representing many traditional database application, but there are some more complex application are present such as telecommunication, GIS (Geographic Information System) ,CAD/CAM . These type of data database required complex requirement.*
- **The extended features are:**
 - **Specialization**
 - **Generalization**
 - **Aggregation.**

Super Class and Sub Class relationship:-

➤ This is the inheritance concept, where we can say that all unity that is a member of a subclass which inherits all the attributes of the entity as a member of the super class and it also inherits all the relationship in which the super class participates.

■ For example, the entity type EMPLOYEE describes the type of each employee entity, and also refers to the COMPANY database. The entity type EMPLOYEE (super class) has some sub groupings such as ENGINEER, TECHNICAN, SECRETARY (subclass).

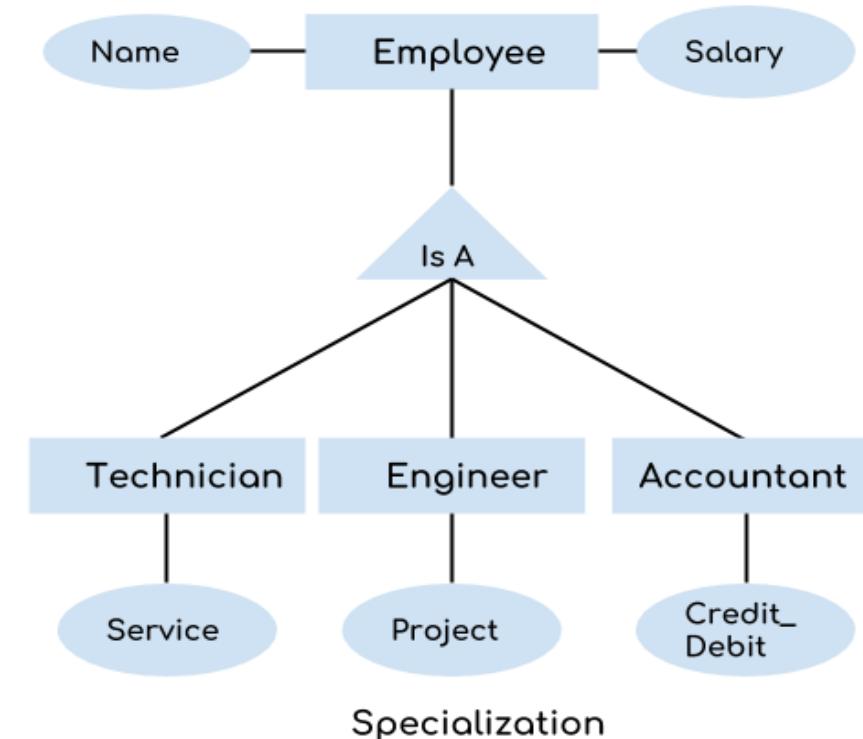
■ Here we can say that EMPLOYEE/ENGINEER, EMPLOYEE/TECHNICIAN and EMPLOYEE/SECRETARY ARE the super class/subclass relationships.

SPECIALIZATION:-

➤ Specialization is the process of defining a set of subclasses of an entity type, this entity type is called as the super class of the specialization.

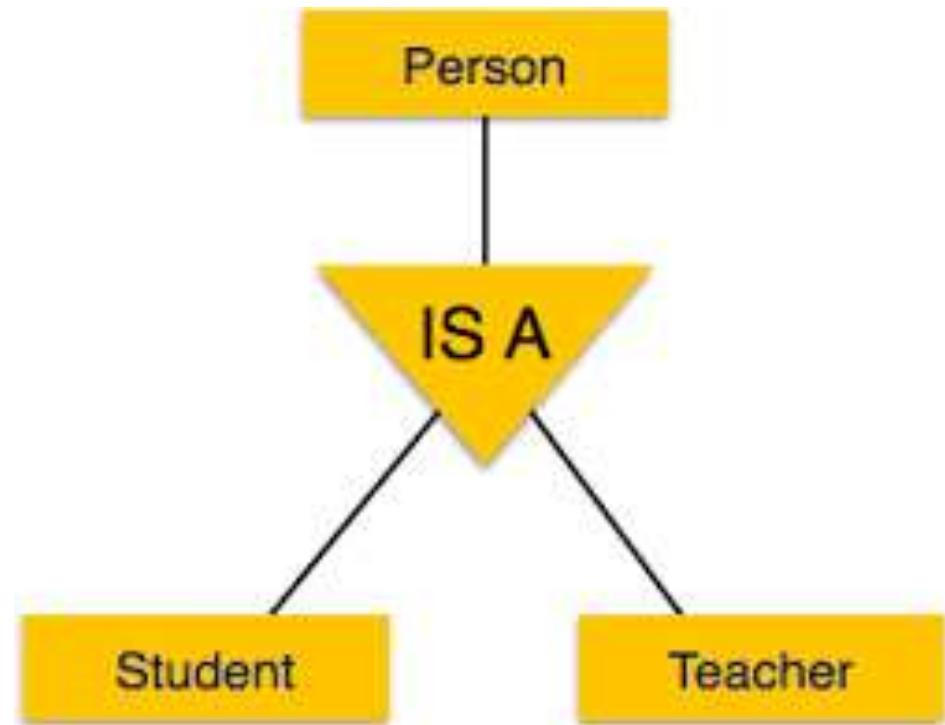
■ For example, {ENGINEER, TECHNICIAN, SECRETARY} is a specialization of the EMPLOYEE super class that distinguishes among EMPLOYEE entities based on the job type of each EMPLOYEE entity.

➤ In ER-diagram, specialization is depicted by a triangle component labeled ISA .The label ISA stands for “is a” and represents, for example, that a TECHNICIAN “is an” EMPLOYEE.



GENERALIZATION:-

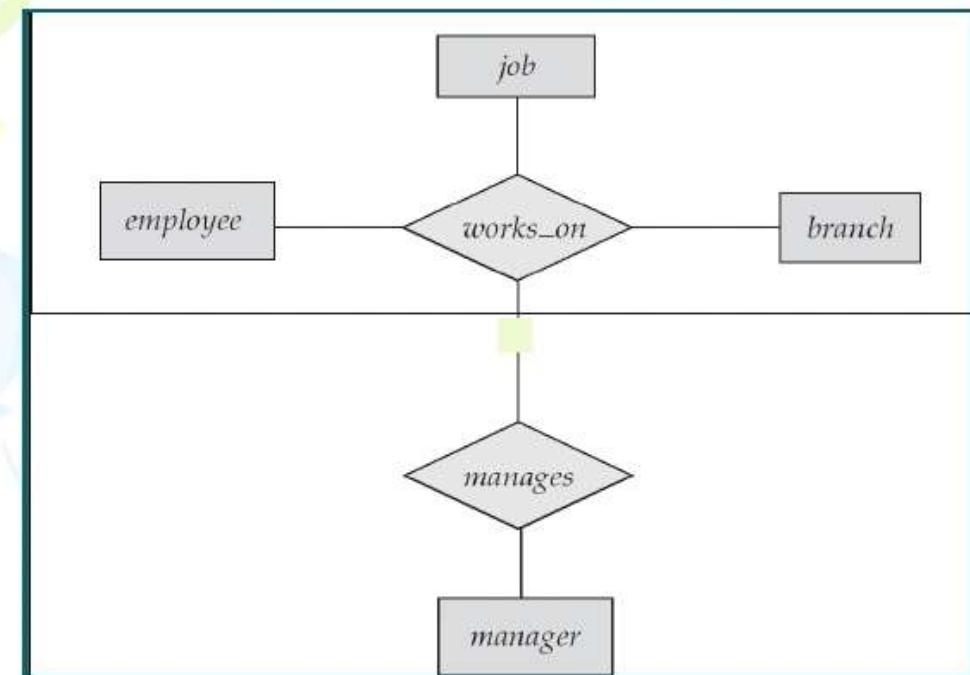
- Generalization is the process of identifying the common features of the different entities i.e the subclass and generalizes them into a single super class.
 - For example, consider the two entity types ENGINEER, TECHNICIAN which has several common attributes and they can be generalized to form the super class EMPLOYEE.
- A double line pointing to the generalized super class represents a generalization, where as simply arrow pointing to the specialized subclasses represents specialization.
- Generalization is a bottom up approach.



AGGREGATION

- The aggregation is the concept which expresses relationships among relationships.
- Aggregation is an abstraction through which relationships are treated as higher level entities.
 - For example, the relationship set works-on (relating the entity sets EMPLOYEE, DEPARTMENT and JOB) as a higher level entity set called works-on. Such an entity set is treated in the same marks as in any other entity set. We can then create a binary relationship manages between works-on and manager to represent who manages what task.

E-R Diagram With Aggregation



Relational Algebra

A Relational data model includes a set of queries to manipulate the database, for which there are two formal languages are in use

- Relational Algebra
- Relational calculus

- Relational algebra is a *procedural query language*, which takes instances of relations as input and yields instances of relations as output.
- It uses *operators* to perform queries. An operator can be either *unary or binary*.
- They accept *relations* as their input and yield *relations* as their output.

Relational Algebra

The fundamental operations of relational algebra are as follows :

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

Select Operation (σ)

- It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma p(r)$

Where σ stands for selection predicate and r stands for relation. p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like – $=, \neq, \geq, <, >, \leq$.

- Example-1

$\sigma_{\text{subject} = \text{"database"}}(\text{Books})$

Output – Selects tuples from books where subject is 'database'.

Select Operation (σ)

Example-2

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"}}(\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

Example-3

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"} \text{ or } \text{year} > \text{"2010}}(\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Select Operation (σ)

Roll No.	Name	Dept.	Age	Address	Gender
101	Sachin	CSE	22	Gunupur	Male
102	Rahul	IT	24	Rayagada	Male
103	Sourav	ECE	23	Gunupur	Male
104	Laxman	CSE	24	Bhubaneswar	Male

Query:

Find all the tuples from Student relation of CSE department

$\sigma_{\text{Dept.} = \text{"CSE"}}(\text{STUDENT})$

Select Operation (σ)

➤ We can combine several conditions by using the connections \wedge (and), \vee (or), \neg (not)

Query:

Find the tuples from Student relation of CSE department who are staying at Gunupur.

$\sigma_{\text{Dept.} = \text{"CSE"} \wedge \text{Address} = \text{"Gunupur"}}$ (STUDENT)

Find the tuples from Student relation whose age is greater than 22 and are staying at Gunupur.

$\sigma_{\text{Age} > 22 \wedge \text{Address} = \text{"Gunupur"}}$ (STUDENT)

Project Operation (Π)

➤ It projects column(s) that satisfy a given predicate.

Notation – $\Pi_{A_1, A_2, A_3, \dots, A_n} (r)$

Where A_1, A_2, A_n are attribute names of relation r .

➤ Duplicate rows are automatically eliminated, as relation is a set.

➤ The Project operation selects certain columns from the table/Relation and discards the other column/attributes.

➤ Projection is denoted by the upper case Greek letter Pi (Π)

Select Operation (σ)

Example-1

$\Pi_{\text{subject, author}} (\text{Books})$

Selects and projects columns named as subject and author from the relation Books.

Example-2

Find out the Roll no, name and address of the relation student

$\Pi_{\text{Roll_no, Name, Address}} (\text{STUDENT})$

Composition of Select and Project Operation

➤ Combination of Select and Project operation is required when there is a query which requires some condition and for selecting some particular attributes which satisfies the condition.

Query: Retrieve the Roll and Name of the student having age greater than 23.

$\prod_{\text{Roll}, \text{Name}, \text{Address}} (\sigma_{\text{Age} > 23} (\text{STUDENT}))$

OR

$\text{Temp} \leftarrow \sigma_{\text{Age} > 23} (\text{STUDENT})$

$\text{Result} \leftarrow \prod_{\text{Rollno}, \text{Name}} (\text{Temp})$

Rename Operation (ρ)

- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ .
- Allows us to name, and therefore to refer to, the results of relational-algebra expressions, allows us to refer to a relation by more than one name.

Example:

$$\rho_x(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n , then

$$\rho_{x(A_1, A_2, \dots, A_n)}(E)$$

- Returns the result of expression E under the name X , and with the attributes renamed to A_1, A_2, \dots, A_n .

Binary Operation

- Binary operation between two relations must be *union compatible*.
- The relation R (A₁, A₂,....A_n) and S(B₁, B₂,....B_n) are said to be union compatible if they have the same degree n and if $\text{dom}(A_i) = \text{dom}(B_i)$ for $1 \leq i \leq n$.
- This means that two relations must have *same number of attributes* and should have *compatible pair of attributes with same type of domains*.

Union Operation (U)

- The result of this operation denoted by $R \cup S$ between two relation R and S that includes all tuples that are either in R or in S or in both R and S.
- Duplicate tuples are eliminated.

Example

Query: find out all the customer name of the bank who have either an account or both account and loan.

$$\Pi_{\text{Cust_name}} (\text{ACCOUNT}) \cup \Pi_{\text{Cust_name}} (\text{LOAN})$$

ACCOUNT	
AC_NO	CUST_NAME
A1001	Sachin
A1002	Rahul
A1003	Saurav
A1004	Sachin
A1005	Laxman

LOAN	
LOAN_NO	CUST_NAME
L2001	Sachin
L2002	Rahul
L2003	Rahul
L2004	Saurav
L2005	Saurav



CUST_NAME
Sachin
Rahul
Saurav
Laxman

Intersection Operation (\cap)

- The intersection operation denoted by $R \cap S$ between two relation R and S that includes all tuples that are in both R and S.

Example

Query: find out all the customer name of the bank who have both account and loan.

$$\Pi_{Cust_name} (ACCOUNT) \cap \Pi_{Cust_name} (LOAN)$$

ACCOUNT	
AC_NO	CUST_NAME
A1001	Sachin
A1002	Rahul
A1003	Saurav
A1004	Sachin
A1005	Laxman

LOAN	
LOAN_NO	CUST_NAME
L2001	Sachin
L2002	Rahul
L2003	Rahul
L2004	Saurav
L2005	Saurav



CUST_NAME
Sachin
Rahul
Saurav

Set Difference Operation

Notation $r - s$

Defined as:

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

Set differences must be taken between **compatible** relations.

r and s must have the same arity

attribute domains of r and s must be compatible

Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\prod_{Course_id} (\sigma_{semester="Fall"} \wedge year=2009 (SECTION)) - \prod_{Course_id} (\sigma_{semester="Spring"} \wedge year=2010 (SECTION))$$

The result of this operation, denoted by $R-S$, is a relation that includes all tuples that are in R but not in S

$$\prod_{Cust_name} (ACCOUNT) - \prod_{Cust_name} (LOAN)$$

CUST_NAME
Laxman

Cartesian Product/Cross Product/Cross Join (X)

- It's a binary operation but the Relations on which it is applied need not have to be Union compatible.
- This operation is used to combine tuples from two relations.
- The Cartesian Product is denoted by a cross symbol (X)

Mathematically the relation R and S with the Cartesian Product will be:

$$R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$$

than the resultant relation Q with n + m attributes :

$$Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$$

Cartesian Product/Cross Product/Cross Join (X)

Let us consider two relations CUSTOMER and LOAN

CUSTOMER		LOAN		
AC_NO	CUST_NAME	LOAN_NO	BRANCH	AMOUNT
A1	Sachin	A1	Gunupur	50000
A2	Rahul	A2	BBSR	40000
A3	Saurav	A4	Gunupur	30000

The Cartesian Product of CUSTOMER and LOAN contains total of 9 tuples which associates every tuple of CUSTOMER with every tuple of LOAN and the resultant relation would be

Cartesian Product/Cross Product/Cross Join (X)

Query: Find out the details of CUSTOMER who have account at Gunupur.

Temp 1 \leftarrow CUSTOMER X LOAN

Temp2 \leftarrow $\sigma_{\text{Branch} = \text{"Gunupur"}}(\text{Temp1})$

Temp3 \leftarrow $\sigma_{\text{CUSTOMER.AC_NO} = \text{LOAN.AC_NO}}(\text{Temp2})$

RESULT \leftarrow $\Pi_{\text{AC_NO, NAME, BRANCH, AMOUNT}}(\text{Temp3})$

AC_NO	NAME	AC_NO	BRANCH	AMOUNT
A1	Sachin	A1	Gunupur	50000
A1	Sachin	A2	BBSR	40000
A1	Sachin	A4	Gunupur	30000
A2	Rahul	A1	Gunupur	50000
A2	Rahul	A2	BBSR	40000
A2	Rahul	A4	Gunupur	30000
A3	Saurav	A1	Gunupur	50000
A3	Saurav	A2	BBSR	40000
A3	Saurav	A4	Gunupur	30000

Additional Relational Algebra Operation

The additional algebra operation is used to simplify common queries

JOIN OPERATION

- ❖ The JOIN operation denoted by \bowtie is to combine related tuples from two relations into single tuple.
- ❖ It's a binary operation which takes two relations as input and produces a resultant relation as an output
- ❖ The JOIN operation will check the key attribute is present in both the relation or not, if present than it will retrieve the required information from both the tables by comparing the value of the common key attribute.

The following are the type of JOIN operations:

Natural Join Operation

- ❖ The NATURAL JOIN operation first forms the Cartesian product than it forms a selection forcing equality on those attributes that appear in both relation and finally removes the duplicate attributes.
- ❖ Mathematically, for the relation R and S the join operation will be $R \bowtie S$

Example 1: find out the name of the customer who have account at Gunupur branch

Query: $R1 \leftarrow \sigma_{\text{Branch} = \text{"Gunupur}} (\text{CUSTOMER} \bowtie \text{LOAN})$

$\text{RESULT} \leftarrow \Pi_{\text{NAME}}(R1)$

OR

$\Pi_{\text{NAME}}(\sigma_{\text{Branch} = \text{"Gunupur}} (\text{CUSTOMER} \bowtie \text{LOAN}))$

OUTER JOIN

- ❖ The OUTER JOIN operation is an extension of the join operation to deal with missing information.
- ❖ In our previous example, we have lost the branch name and the loan amount information of Sourav in the LOAN relation. Similarly the name of the account number A4 is absent in the ACCOUNT relation.
- ❖ We can use the outer join operation to avoid this loss of information.

There are three types of outer join are there:

- Left outer join
- Right outer join
- Full outer join

LEFT OUTER JOIN

❖ The LEFT OUTER JOIN takes all tuples in the left relation that did not match with any tuple in the right relation, fills the tuples with NULL values for all other attributes from the right relation.

➤ For example the result of the left outer join of

CUSTOMER  LOAN

AC_NO	NAME	BRANCH	AMOUNT
A1	Sachin	Gunupur	50000
A2	Rahul	BBSR	40000
A3	Saurav	NULL	NULL

RIGHT OUTER JOIN

❖ The RIGHT OUTER JOIN takes all tuples in the right relation that did not match with any tuple in the left relation, fills the tuples with NULL values for all other attributes from the left relation.

➤ For example the result of the Right outer join of
CUSTOMER  LOAN

AC_NO	NAME	BRANCH	AMOUNT
A1	Sachin	50000	Gunupur
A2	Rahul	40000	BBSR
A4	NULL	30000	Gunupur

FULL OUTER JOIN

- ❖ The FULL OUTER JOIN takes all tuples from both the relation and fills the missing values with NULL values.
- For example the result of the full outer join of
CUSTOMER  **LOAN**

AC_NO	NAME	BRANCH	AMOUNT
A1	Sachin	50000	Gunupur
A2	Rahul	40000	BBSR
A3	Saurav	NULL	NULL
A4	NULL	30000	Gunupur

Generalized Projection

❖ The Generalized Projection Operation extends the projection operation by allowing arithmetic functions to be used in the projection list

❖ Syntax: $\Pi_{F_1, F_2, \dots, F_n}(E)$

Where E is an relational algebra expression

F_1, F_2, \dots, F_n are the arithmetic expression

EMP_NO	NAME	SALARY
E_001	Sachin	30000
E_002	Rahul	40000

$\Pi_{NAME, (SALARY * 0.40) \text{ as } INCREMENT}(EMPLOYEE)$

NAME	INCREMENT
Sachin	12000
Rahul	16000

Aggregate Functions

- ❖ The Aggregate function takes a collection of values and return a single value as a result.
- ❖ It is denoted by G. (G)

For example: the aggregate function sum takes a collection of values and returns the sum of values.

EX: $G \text{ SUM(SALARY)} (\text{EMPLOYEE})$

SUM(SALARY)
70000

- ❖ The Aggregate functions are SUM, AVG, COUNT, MIN, MAX.

Examples

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Write the following queries in relational algebra.

1. “Find the names of suppliers who supply some red part.”

$$\pi_{\text{sname}}((\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \text{Catalog}) \bowtie \text{Supplier}))$$

2. “Find the IDs of suppliers who supply some red or green part.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}=\text{'red'} \vee \text{colour}=\text{'green'}}(\text{Part}) \bowtie \text{Catalog})$$

OR

$$\pi_{\text{sid}}(\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \text{Catalog} \cup \sigma_{\text{colour}=\text{'green'}}(\text{Part}) \bowtie \text{Catalog}).$$

Examples

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Write the following queries in relational algebra.

❖ “Find the IDs of suppliers who supply some red part or are based at 21 George Street.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \text{Catalog}) \cup \pi_{\text{sid}}(\sigma_{\text{address}=\text{'21G.S.'}}(\text{Supplier})).$$

❖ “Find the names of suppliers who supply some red part or are based at 21 George Street.”

$$\begin{aligned} & \pi_{\text{sname}}(\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \text{Catalog} \bowtie \text{Supplier}) \\ & \quad \cup \pi_{\text{sname}}(\sigma_{\text{address}=\text{'21G.S.'}}(\text{Supplier})). \end{aligned}$$

Examples

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Write the following queries in relational algebra.

❖ “Find the IDs of suppliers who supply some red part and some green part.”

$$\pi_{\text{sid}}(\sigma_{\text{colour}=\text{'red'}}(\text{Part}) \bowtie \text{Catalog}) \cap \pi_{\text{sid}}(\sigma_{\text{colour}=\text{'green'}}(\text{Part}) \bowtie \text{Catalog})$$

❖ “Find the IDs of suppliers who supply only red parts.”

$$\pi_{\text{sid}}(\text{Supplier}) \setminus \pi_{\text{sid}}(\text{Catalog} \bowtie \sigma_{\text{colour} \neq \text{'red'}}(\text{Part})).$$

Examples

Give the following queries in the relational algebra using the relational schema

student(id, name)

enrolledIn(id, code)

subject(code, lecturer)

1. What are the names of students enrolled in cs3020?

Solution: $\Pi_{\text{name}}(\sigma_{\text{cs3020}=\text{code}}(\text{student} \bowtie \text{enrolledIn}))$

2. Which subjects is Hector taking?

Solution: $\Pi_{\text{code}}(\sigma_{\text{name}=\text{Hector}}(\text{student} \bowtie \text{enrolledIn}))$

3. Who teaches cs1500?

Solution: $\Pi_{\text{lecturer}}(\sigma_{\text{code}=\text{cs1500}}(\text{subject}))$

Examples

Give the following queries in the relational algebra using the relational schema

student(id, name)

enrolledIn(id, code)

subject(code, lecturer)

4. Who teaches cs1500 or cs3020?

Solution: $\Pi_{\text{lecturer}}(\sigma_{\text{code}=\text{cs1500} \text{ OR } \text{code}=\text{cs3020}}(\text{subject}))$

5. Who teaches at least two different subjects?

Solution: For this query we have to relate subject to itself. To disambiguate the relation, we will call the subject relation R or S.

$\Pi_{\text{lecturer}}(\sigma_{R.\text{lecturer} = S.\text{lecturer} \text{ AND } R.\text{code} <> S.\text{code}}(R \bowtie S))$

6. What are the names of students in cs1500 or cs3010?

Solution: $\Pi_{\text{name}}(\sigma_{\text{code}=\text{cs1500}}(\text{student} \bowtie \text{enrolledIn})) \cap \Pi_{\text{name}}(\sigma_{\text{code}=\text{cs3010}}(\text{student} \bowtie \text{enrolledIn}))$

Examples

Give the following queries in the relational algebra using the relational schema

student(id, name)

enrolledIn(id, code)

subject(code, lecturer)

7. What are the names of students in both cs1500 and cs1200?

Solution: $\Pi_{\text{name}}(\sigma_{\text{code}=cs1500}(\text{student} \bowtie \text{enrolledIn})) \cap \Pi_{\text{name}}(\sigma_{\text{code}=cs3010}(\text{student} \bowtie \text{enrolledIn}))$

8. What are the codes of all the subjects taught?

Solution: $\Pi_{\text{code}}(\text{subject})$

9. What are the names of all the students in cs1500?

Solution: $\Pi_{\text{name}}(\sigma_{\text{code}=cs1500}(\text{student} \bowtie \text{enrolledIn}))$

Examples

Give the following queries in the relational algebra using the relational schema

student(id, name)

enrolledIn(id, code)

subject(code, lecturer)

10. What are the names of students taking a subject taught by Roger.

Solution: $\Pi_{\text{name}}(\sigma_{\text{lecturer}=\text{Roger}}(\text{student} \bowtie \text{enrolledIn} \bowtie \text{subject}))$

11. What are the names of students who are taking a subject not taught by Roger?

Solution: $\Pi_{\text{name}}(\sigma_{\text{lecturer} <> \text{Roger}}(\text{student} \bowtie \text{enrolledIn} \bowtie \text{subject}))$

Assignment

Solve the following queries in the relational algebra using the relational schema

`lives(person-name, street, city)`

`works(person-name, company-name, salary)`

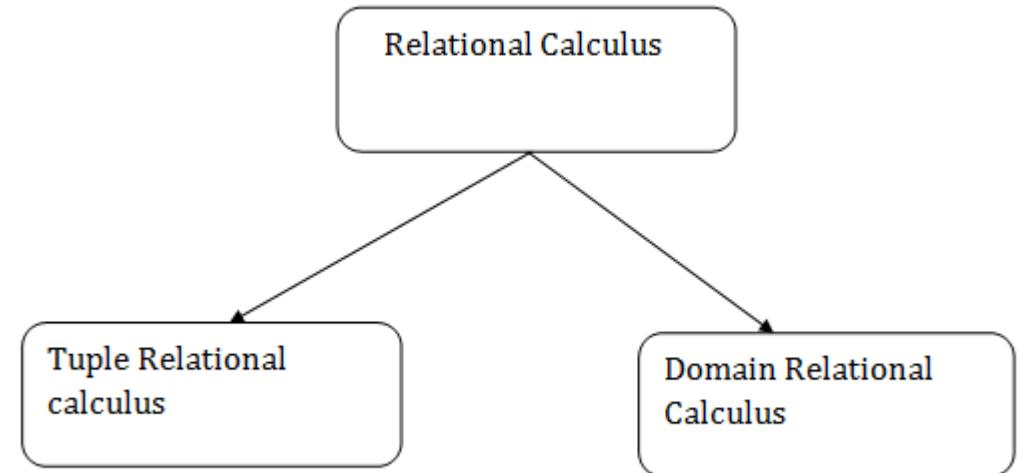
`located-in(company-name, city)`

`manages(person-name, manager-name)`

1. Find the name of all employees (i.e., persons) who work for the City Bank Company (which is a specific company in the database).
2. Find the name and city of all employees who work for City Bank. Similar to previous query, except we have to access the lives table to extract the city of the employee. The join condition is the same person name in the two tables Lives and Works.
3. Find the name, street and city of all employees who work for City Bank and earn more than \$10,000. Similar to previous query except an additional condition on salary attribute.
4. Find all persons who do not work for City Bank.

Relational Calculus

- ❖ A Relational Calculus is a Non Procedural Language
- ❖ In relational calculus there is no description of how to evaluate a query, it only specifies what is to be retrieved rather than how to retrieve it.



Tuple relational calculus

The tuple relational calculus is based on specifying a number of tuple variable

Syntax: $\{t \mid \text{COND}(t)\}$

where t is a tuple variable

$\text{COND}(t)$ is a conditional expression involving t.

Tuple Relational Calculus

❖ Display all the employee details or all the tuple of EMPLOYEE relation

$$\{t \mid \text{EMPLOYEE}(t)\}$$

❖ Display all the employee details whose salary is above 30000

$$\{t \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{SALARY} > 30000\}$$

❖ Retrieve the name of the employee whose salary is above 30000

$$\{t.\text{NAME} \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{SALARY} > 30000\}$$

In tuple relational calculus, we have to specify the requested attributes for each selected t , then we specify the condition for selecting a tuple following the bar (\mid).

Tuple Relational Calculus

❖ $\{t1.A1, t2.A2, \dots, tn.An \mid \text{COND } (t1, t2, \dots, tn)\}$

Where,

$t1, \dots, tn$ are tuple variables

Ai is the attribute on which ti ranges and

COND is a condition or formula of the expression.

- ❖ A formula / condition is made up of one or more atoms connected via the logical operator AND, OR, NOT.
- ❖ There are two quantifiers, which may appear in the formulas that is Existential / there exist (\exists) and universal / for all (\forall) quantifier.
- ❖ A tuple variable t is said to be bound if it appears with $(\exists t)$ or $(\forall t)$, otherwise it is free.

Existential Quantifier: (\exists)

- ❖ If F is a formula, then so is $(\exists t) (F)$, where t is a tuple variable.
- ❖ The formula $(\exists t) (F)$ is TRUE if the formula evaluates to TRUE for some (at least one) tuple, otherwise $(\exists t) (F)$ is false.

Query : find the name and account number of the relation CUSTOMER and LOAN who have loan account

$$\{t.\text{NAME}, t.\text{AC_NO} \mid \text{CUSTOMER}(t) \text{ and } (\exists d) (\text{LOAN}(d)) \text{ and } t.\text{AC_NO} = d.\text{AC_NO}\}$$

Client(ID, fName, lName, Age)
Matches(Client1, Client2, Type)

Query: List the first and last names of clients that appear as client1 in a match of any type.

$$\{c.\text{fName}, c.\text{lName} \mid \text{CLIENT}(c) \text{ AND } (\exists m) (\text{MATCHES}(m) \text{ AND } c.\text{ID} = m.\text{Client1})\}$$

Universal Quantifier: (\forall)

- ❖ If F is a formula, than so is $(\forall t) (F)$, where t is a tuple variable.
- ❖ The formula $(\forall t) (F)$ is TRUE if the formula evaluates to TRUE for every tuple, otherwise $(\forall t) (F)$ is false.

Query : find the name and account number of the relation CUSTOMER and LOAN who have no loan account

$\{t.\text{NAME}, t.\text{AC_NO} \mid \text{CUSTOMER } (t) \text{ and } (\forall d) (\text{NOT } (\text{LOAN}(d))) \text{ or NOT } (t.\text{AC_NO} = d.\text{AC_NO})\}$

Domain Relational Calculus

❖ The domain relational calculus uses domain variable that take an value from an attribute domain, rather than values for an entire tuple.

❖ The domain relational calculus is in the form of

$$\{<x_1, x_2, \dots, x_n> \mid \text{COND}(x_1, x_2, \dots, x_n)\}$$

Where,

x_1, x_2, \dots, x_n represents the domain variable

COND represents the formula composed of atoms.

Domain Relational Calculus

CUSTOMER				
AC_NO	NAME	AGE	GENDER	ADDRESS

LOAN		
AC_NO	BRANCH	AMOUNT

❖ Find the name of the CUSTOMER whose age is greater than 50

$$\{b \mid (\exists a) (\exists c) (\exists d) (\exists e) (\text{CUSOMER}(abcde) \text{ AND } c > 50)\}$$

❖ Find the NAME, AC_NO, BRANCH and AMOUNT whose branch = “Gunupur”.

$$\{abyz \mid (\exists c) (\exists d) (\exists e) (\exists x) (\text{CUSOMER}(abcde) \text{ AND } \text{LOAN}(xyz) \text{ AND } y = \text{"Gunupur"} \text{ AND } a = x)\}$$

QUERY BY EXAMPLE: (QBE)

- ❖ QBE queries are expressed “by example” instead of giving a procedure for obtaining the desire answer, the user gives an example of what is desired.
- ❖ The queries in QBE is expressed by skeleton tables which shows the relation schema.
- ❖ The user selects those skeletons needed for a given query and fills in the skeleton with example rows.
- ❖ An example row consists of constants and example elements which are domain variables, and QBE uses an underscore character before domain variable as _x.

QUERY BY EXAMPLE: (QBE)

❖Query: Print the name of the customer whose age is greater than 50.

AC_NO	NAME	AGE	GENDER	ADDRESS
	P.	>50		

❖Query: find the account number and address where the account is jointly on Ram and Shyam.

AC_NO	NAME	AGE	GENDER	ADDRESS
P. _x	Ram Shyam			P. _y P. _z

QUERY BY EXAMPLE: (QBE) – CONDITION BOX

- ❖ It is very much difficult to express all the constraints in the domain variables within the skeleton tables.
- ❖ To overcome this difficulty, QBE includes a condition box feature that allows the logical expression to appear in a condition box.
- ❖ find the account number of the customers within an amount between 25000 to 30000 but not exactly 28000.

AC_NO	BRANCH	AMOUNT
P.		_x

CONDITION
$_x = (>=25000 \text{ AND } <=30000 \text{ AND } \neg 28000)$

QUERY BY EXAMPLE: (QBE) – RESULT RELATION

- ❖ This is the mechanism where the result of a query includes attributes from several relation schema, where it displays the desired result in a single table.
- ❖ For this purpose we can declare a temporary result relation that includes all the attributes of the result of the query.
- ❖ we print the desired result by including the command P. in only the result skeleton table.

Display the name and amount where branch = “Delhi”

CUSTOMER				
AC_NO	NAME	AGE	GENDER	ADDRESS
_x	_y			

LOAN		
AC_NO	BRANCH	AMOUNT
_x	Delhi	_z

RESULT	NAME	AMOUNT
P.	_y	_z

Introduction:

- E.F. Codd (Edgar Frank Codd) of IBM had written an article “*A relational model for large shared data banks*” in June 1970 in the Association of Computer Machinery (ACM) Journal
- One of the most significant implementations of the relational model was “*System R*,” which was developed by IBM during the late 1970s. System R was intended as a “proof of concept” to show that relational database systems could really build and work efficiently. It gave rise to major developments such as a *structured query language* called SQL which has since become an ISO standard and de facto standard relational language.
- Various commercial relational DBMS products were developed during the 1980s such as *DB2*, *SQL/DS*, and *Oracle*. In relational data model the data are stored in the form of tables.

CODD'S 12 Rules:

➤ In 1985, Codd published a list of rules that became a standard way of evaluating a relational system. After publishing the original article Codd stated that there are no systems that will satisfy every rule. Nevertheless the rules represent relational ideal and remain a goal for relational database designers.

Rule-1: The Information Rule: All information in a relational database is represented explicitly at the logical level and in exactly one way-by values in tables:

- *Data should be presented to the user in the tabular form.*

Rule-2: Guaranteed Access Rule: Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name:

- *Every data element should be unambiguously accessible.*

CODD'S 12 Rules:

Rule-3: Systematic Treatment of Null Values: Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Rule-4: Dynamic On-line Catalogue Based on the Relational Model: The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data:

- The database description should be accessible to the users.

Rule-5: Comprehensive Data Sublanguage Rule: A relational system may support several languages and various modes of terminal use (for example the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all the following is comprehensive: data definition, view definition, data manipulation (interactive and by program), integrity constraints, and transaction boundaries:

- A database supports a clearly defined language to define the database, view the definition, manipulate the data, and restrict some data values to maintain integrity.

CODD'S 12 Rules:

Rule-6: View Updating Rule: All views that are theoretically updatable are also updatable by the system:

- Data should be able to be changed through any view available to the user.

Rule-7: High-level Insert, Update, and Delete: The capacity of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data:

- All records in a file must be able to be added, deleted, or updated with singular commands

Rule-8: Physical Data Independence: Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods:

- Changes in how data are stored or retrieved should not affect how a user accesses the data.

CODD'S 12 Rules:

Rule-9: Logical Data Independence: Application programs and terminal activities remain logically unimpaired whenever information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables:

- A user's view of data should be unaffected by its actual form in files.

Rule-10: Integrity Independence: Integrity constraints specific to a particular relational database must be definable in a relational data sublanguage and storable in the catalogue, not in the application programs.

- Constraints on user input should exist to maintain data integrity.

Rule-11: Distribution Independence: A relational DBMS has distribution independence. Distribution independence implies that users should not have to be aware of whether a database is distributed.

- A database design should allow for distribution of data over several computer sites.

CODD'S 12 Rules:

Rule-12: Nonsubversion Rule: Nonsubversion Rule. If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time):

- Data fields that affect the organization of the database cannot be changed.

There is one more rule called Rule Zero which states that “For any system that is claimed to be a relational database management system, that system must be able to manage data entirely through capabilities.”

Concept of Key

Key is an attribute or group of attributes, which is used to identify a row in a relation.
Key can be broadly classified into

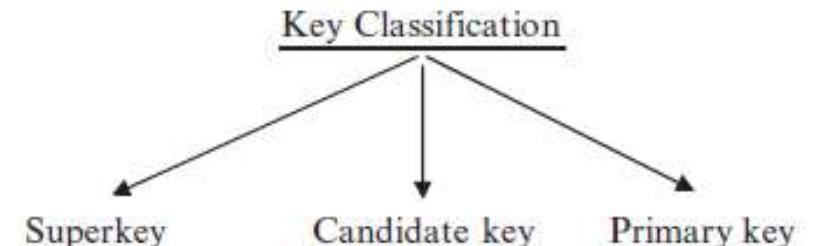
- (1) Super-key
- (2) Candidate key and
- (3) Primary key

Super-key:

A super-key is a subset of attributes of an entity-set that uniquely identifies the entities.
Super-keys represent a constraint that prevents two entities from ever having the same value for those attributes

Candidate Key:

Candidate key is a minimal super-key. A candidate key for a relation schema is a minimal set of attributes whose values uniquely identify tuples in the corresponding relation.



Concept of Key

Primary Key:

The primary key is a designated candidate key. It is to be noted that the primary key should not be null.

Example

Consider the employee relation, which is characterized by the attributes, employee ID, employee name, employee age, employee experience, employee salary, etc. In this employee relation:

Superkeys can be employee ID, employee name, employee age, employee experience, etc.

Candidate keys can be employee ID, employee name, employee age.

Primary key is employee ID.

Note: If we declare a particular attribute as the primary key, then that attribute value cannot be NULL. Also it has to be distinct.

Foreign Key

Foreign key is set of fields or attributes in one relation that is used to “refer” to a tuple in another relation.

Relational Integrity:

Data integrity constraints refer to the accuracy and correctness of data in the database. Data integrity provides a mechanism to maintain data consistency for operations like INSERT, UPDATE, and DELETE. The different types of data integrity constraints are Entity, NULL, Domain, and Referential integrity.

Entity Integrity:

Entity integrity implies that a primary key cannot accept null value. The primary key of the relation uniquely identifies a row in a relation. Entity integrity means that in order to represent an entity in the database it is necessary to have a complete identification of the entity's key attributes.

Null Integrity:

Null implies that the data value is not known temporarily. Consider the relation PERSON. The attributes of the relation PERSON are name, age, and salary. The age of the person cannot be NULL.

Domain Integrity Constraint:

The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values.

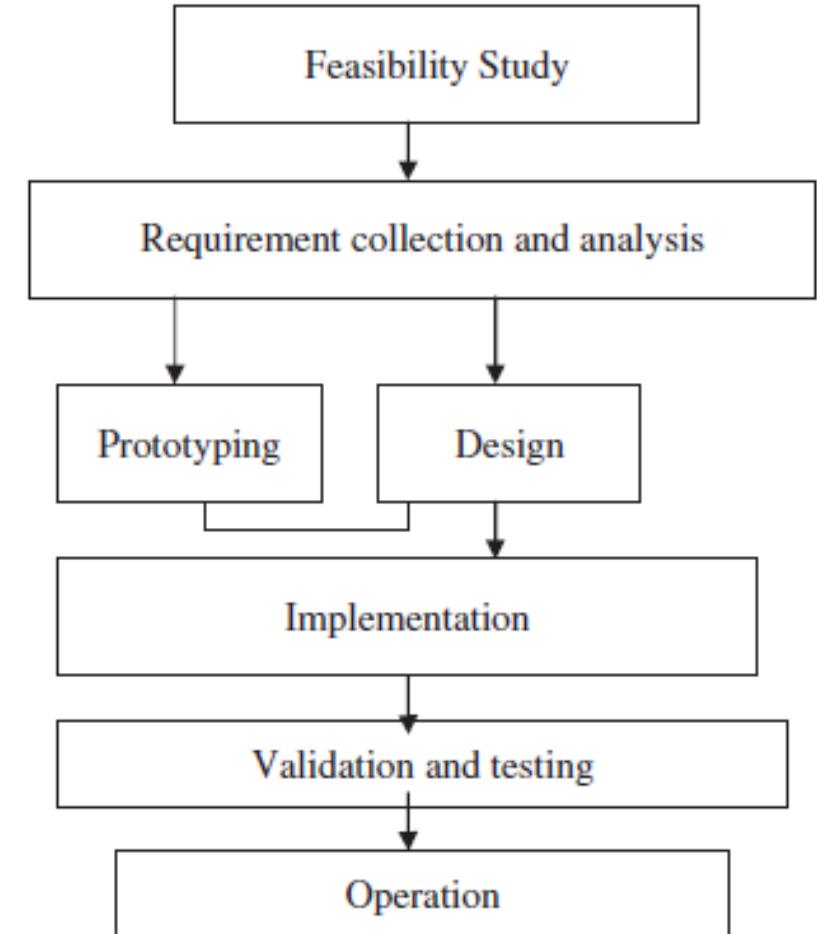
Relational Integrity:

Referential Integrity:

- ❖ In the relational data model, associations between tables are defined through the use of foreign keys. The referential integrity rule states that a database must not contain any unmatched foreign key values.
- ❖ It is to be noted that referential integrity rule does not imply a foreign key cannot be null.
- ❖ There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null.
- ❖ A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

Database design process

- Database design process integrates relevant data in such a manner that it can be processed through a mechanism for recording the facts.
- The database design is a complex process and the complexity arises mainly because of the identification of relationship among individual components and their representation for maintaining correct functionality are highly involved.
- The degree of complexity increases if there are many-to-many relationships among individual components. The process of database design usually requires a number of steps



Steps in database design

Requirement Collection and Analysis:

In requirement collection, one has to decide what data are to be stored, and to some extent, how that data will be used.

Prototyping and Design

Design implies a procedure for analyzing and organizing data into a form suitable to support business requirements and makes use of strategic technology. The three phases in relational database design are *conceptual design, logical design, and physical design*.

Implementation

Database implementation involves development of code for database processing, and also the installation of new database contents, usually from existing data sources.

Objectives of Database Design

Efficiency: Efficiency is generally considered to be the most important. The design should make full and efficient use of the facilities provided. If the database is made online, then the users should interact with the database without any time delay.

Integrity: The term integrity means that the database should be as accurate as possible.

Privacy: The database should not allow unauthorized access to files.

Security: The database, once loaded, should be safe from physical corruption whether from hardware or software failure or from unauthorized access.

Objectives of Database Design

Implementation: The conceptual model should be simple and effective so that mapping from conceptual model to logical model is easy.

Flexibility: The database should not be implemented in a rigid way that assumes the business will remain constant forever. Changes will occur and the database must be capable of responding readily to such change.

From the ER model a good relational database can be designed. But in the ER model there may be some amount of inconsistency, ambiguity and redundancy. To reduce this type of problem the normalization process is required.

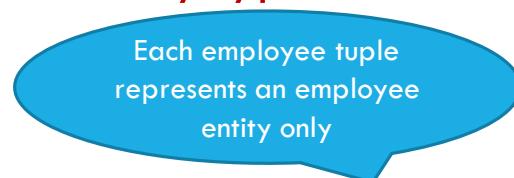
INFORMAL DESIGN GUIDELINES FOR RELATIONAL SCHEMAS :

Why normalization is required?

The following are the some informal design guidelines for the relational schemas:-

Semantics of relation attributes (guideline -1):

Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relation types into a single relation.



EMPLOYEE				
ENAME	ENO	DOB	ADDRESS	DNUMBER



DEPARTMENT		
DNAME	DNUMBER	DMGRNO

INFORMAL DESIGN GUIDELINES FOR RELATIONAL SCHEMAS :

Redundant information in tuples and update anomalies (irregularities):-

- Design the relation schema so that there should not be any redundant (duplicate) information.
- Design the relation schema so that insertion of new data, deletion / updation of existing data should not be difficult.

GUIDELINE-2: Design the relation schema so that no insertion, deletion or modification anomalies (irregularities) are present in the relations. If any anomalies are present, note them clearly, and make sure that the programs that update the database will operate correctly.

Null value in tuples (guideline -3): As far as possible, avoid placing attributes in a relation whose values may frequently be null. If nulls are unavoidable, make sure that they apply in exceptional cases and do not apply to a majority of tuples in the relation.

INFORMAL DESIGN GUIDELINES FOR RELATIONAL SCHEMAS :

Generation of spurious tuples(guideline -4):

- Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys are generated.
- Avoid relations that contain matching attributes that are not primary key or foreign key combinations, because joining on such attributes may produce spurious tuples.

FUNCTIONAL DEPENDENCIES

- ❖ Functional dependencies are the relationships among the attributes within a relation. Functional dependencies provide a formal mechanism to express constraints between attributes.
- ❖ If attribute A functionally depends on attribute B, then for every instance of B you will know the respective value of A.
- ❖ Attribute “B” is functionally dependent upon attribute “A” (or collection of attributes) if a value of “A” determines or single value of attributes “B” at only one time functional dependency helps to identify how attributes are related to each other.

FUNCTIONAL DEPENDENCIES

Suppose a relation / table R contains two attributes X and Y , we can say that Y is functional depends upon attributes X if and only if X uniquely determines the value of Y.

for the two tuple t1 and t2 in the relation R

$$\text{if } t1[X] = t2[X]$$

$$\text{then } t1[Y] = t2[Y]$$

- So, this means the values of Y component of a tuple is determined by the values of X component of a tuple.
- We can say that there is a functional dependency from X to Y is functionally dependent on X.
- The abbreviation for functional dependency is FD or f.d and it can be represented by $X \rightarrow Y$.
- The set of attributes X is called the left-hand side of the FD, and Y is called the right –hand side.

FUNCTIONAL DEPENDENCIES

Example

lets us consider the following example :-

EMPLOYEE				
ENAME	ENO	DOB	ADDRESS	DNUMBER

Here, by using the key attributes ENO we can derive the other attributes ENAME, ADDRESS and DOB and by using the ENO and DOB we can derive the attribute AGE.

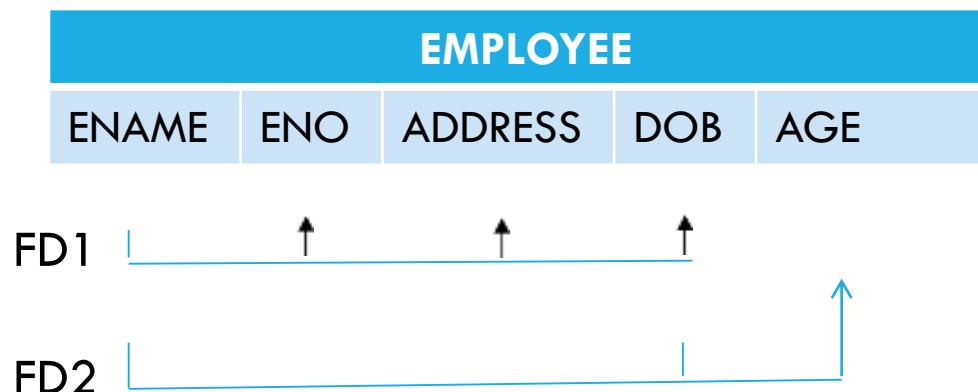
It can be represented by

FD1: $\text{ENO} \rightarrow \{\text{ENAME}, \text{ADDRESS}, \text{DOB}\}$

FD2: $\{\text{ENO}, \text{DOB}\} \rightarrow \text{AGE}$

FUNCTIONAL DEPENDENCIES

The functional dependencies FD1 and FD2 can be represented graphically as :-



- Each FD is displayed as a horizontal line. The left hand side attributes of the FD are connected by vertical lines to the lines represented the FD, while the right-hand side attributes are connected by arrows pointing towards the attributes.
- Definition :- A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subset of R, specifies a constraint and that is for any two tuples t_1 and t_2 that have $t_1[X] = t_2[X]$ then they must also have $t_1[Y] = t_2[Y]$.

FULL FUNCTIONAL DEPENDENCY:-

- A functional dependency (FD) $X \rightarrow Y$ is a full functional dependency (FFD) if removal of any attributes A from X means that the dependency does not hold any more.

$\{ENO, DOB\} \rightarrow AGE$

- Here, the combination of ENO and DOB uniquely identifies the AGE of the employee. Here age cannot be determined by the use of DOB or ENO only.

PARTIAL FUNCTIONAL DEPENDENCY:-

- A Functional dependency (FD) $X \rightarrow Y$ is partial functional dependency if some attributes $A \in X$ can be removed from X and the dependency still holds.

- For example, $\{ENAME, ENO\} \rightarrow ADDRESS$

Here, if we remove ENAME then $ENO \rightarrow ADDRESS$ still holds.

Example Database

Movies

title	director	myear	rating
Fargo	Coen	1996	8.2
Raising Arizona	Coen	1987	7.6
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

Actors

actor	ayear
Cage	1964
Hanks	1956
Maguire	1975
McDormand	1957

Acts

actor	title
Cage	Raising Arizona
Maguire	Spiderman
Maguire	Wonder Boys
McDormand	Fargo
McDormand	Raising Arizona
McDormand	Wonder Boys

Directors

director	dyear
Coen	1954
Hanson	1945
Raimi	1959

Some Queries

- Find movies made after 1997
- Find movies made by Hanson after 1997
- Find all movies and their ratings
- Find all actors and directors
- Find Coen's movies with McDormand
- Find movies with Maguire but not McDormand
- Find actors who have acted in some Coen's movie
- Find (director, actor) pairs where the director is younger than the actor
- Find actors who have acted in all of Coen's movies

Solutions Using Relational Algebra

Query: Find movies made after 1997

$$\sigma \text{ myear} > 1997 (\text{Movies})$$

Query: Find movies made by Hanson after 1997

$$\sigma \text{ myear} > 1997 \wedge \text{director} = \text{'Hanson'} (\text{Movies})$$

Query: Find all movies and their ratings

$$\pi \text{ title, rating} (\text{Movies})$$

Query: Find all actors & directors

$$\pi \text{ actor} (\text{Actors}) \cup \pi \text{ director} (\text{Directors})$$

Solutions Using Relational Algebra

Query: Find Coen's movies with McDormand

```
e1 = π title(σactor='McDormand' (Acts))
e2 = π title(σdirector='Coen' (Movies))
result = e1 ∩ e2
```

Query: Find movies with Maguire but not 'McDormand'

```
σactor='Maguire'(Acts) – σ actor='McDormand'(Acts)
```

Query: Find actors who have acted in some Coen's movies

```
e1 = ρT(title2)(πtitle(σdirector='Coen(Movies)))
```

Solutions Using Relational Algebra

Query: Find (director, actor) pairs where the director is younger than the actor

$$\pi \text{ director, actor}(\text{Directors} \bowtie \text{dyear} > \text{ayear} \text{ Actors})$$

Query: Find actors who have acted in all Coen's movies

$$\pi \text{ title}(\sigma \text{director} = \text{'Coen'}(\text{Movies}))$$

INFERENCE RULES FOR FUNCTIONAL DEPENDENCY: -

- Suppose, F is set of functional dependencies that are specified on relation schema R. It is not possible to specify all possible functional dependencies for a given situation .

For example if each department has one manager so that DEPT_NO uniquely determines MANAGER_ENO ($DEPT_NO \rightarrow MGR_ENO$), and a manager has a unique phone number called MGR_PHONE ($MGR_ENO \rightarrow MGR_PHONE$), then these two dependencies together imply that $DEPT_NO \rightarrow MGR_PHONE$.

- This is an *inferred FD* and need not be explicitly stated in addition to the two given FDs.
- Therefore formally it is useful to define a concept called *closure* that includes all possible dependencies that can be inferred from the given set F.

INFERENCE RULES FOR FUNCTIONAL DEPENDENCY: -

DEFINITION

- The set of all dependencies that includes F as well as all dependencies that can be inferred from F is called the closure of F , it is denoted by F^+ .

Example:

$$F = ENO \rightarrow \{ENAME, DOB, ADDRESS, DNUM\} \quad DNUM \rightarrow \{DNAME, ADDRESS\}$$

$$F^+ = ENO \rightarrow \{DNAME, DMGRNO\}$$

- To determine a systematic way to infer dependencies we must discover a set of inference rules that can be used to infer new dependencies from a given set of dependences F .

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR1(REFLEXIVE RULE): -

For a relation R having attributes X and Y.

If $X \supseteq Y$, then $X \rightarrow Y$

Proof: - Suppose $X \supseteq Y$ and two tuples t_1 and t_2 exist in some relation,

such that, $t_1[X] = t_2[X]$ then, $t_1[Y] = t_2[Y]$,

because $X \supseteq Y$

Hence, $X \rightarrow Y$ must hold in this relation.

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR2(AUGMENTATION RULE): -

For a relation R having attributes X and Y

If $\{X \rightarrow Y\}$ then $XZ \rightarrow YZ$

Proof: - Let $t_1[X] = t_2[X]$ (1)

Then, $t_1[Y] = t_2[Y]$ (2)

And we can say that $t_1[XZ] = t_2[XZ]$ (3)

From equation 1 and 2, we can say that, $t_1[Z] = t_2[Z]$ (4)

From equation 2 and 4 we can say that, $t_1[YZ] = t_2[YZ]$

So from the above equation we can say that $\{X \rightarrow Y\}$, then $XZ \rightarrow YZ$.

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR3(TRANSITIVE RULE): -

For a relation R having attributes X and Y ,

If $\{X \rightarrow Y, Y \rightarrow Z\}$ then, $X \rightarrow Z$

Proof: -

Assume that $X \rightarrow Y$ and $Y \rightarrow Z$ both hold in relation R then

we can say that, $t1[X]=t2[X]$ and $t1[Y]=t2[Y]$

and we can say that $t1[Z]=t2[Z]$

hence, $X \rightarrow Z$ must hold in relation R.

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR4(DECOMPOSITION /PROJECTIVE) :-

For a relation R having attributes X and Y, If $\{X \rightarrowYZ\}$ then, $X \rightarrow Y$

Proof: -

Given, $X \rightarrowYZ$ 1

$YZ \rightarrow Y$ from IR1 2

Thus from 1 & 2 , we can say that $X \rightarrow Y$

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR5(UNION / ADDITIVE RULE): -

For a relation R having attributes X and Y, If $\{X \rightarrow Y, X \rightarrow Z\}$ then, $X \rightarrow YZ$

Proof

Given $X \rightarrow Y$ and $X \rightarrow Z$ 1

$X \rightarrow XY$ from IR2 2

$XY \rightarrow YZ$ from IR3 3

$X \rightarrow YZ$ using IR3 AND EQUATION 2 AND 3

ARMSTRONG'S AXIOMS OR INFERENCE RULES: -

IR6 (PSEUDOTRANSITIVE RULE) :-

For a relation R having attributes X and Y, If $\{X \rightarrow Y, WY \rightarrow Z\}$ then $WX \rightarrow Z$

PROOF

Given

$$X \rightarrow Y \quad 1$$

$$WY \rightarrow Z \quad 2$$

$$WX \rightarrow WY \text{ (using IR2)} \quad 3$$

$WX \rightarrow Z$ (USING IR3 AND EQUATION 2 AND 3)

CLOSURE:-

- ❖ A systematic way to determine the additional functional dependencies is first to determine each set of attributes X that appears in the left hand side of some functional dependencies in F and then to determine the set of all attributes that are dependent on X.
- ❖ Thus, for each set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F, X^+ is called the closure of X under F.

Example-

$$F = ENO \rightarrow ENAME$$

$$DNO \rightarrow \{DNAME, DLOCATION\}$$

$$X^+ (\text{CLOSURE SET}) = \{ENO\} + = \{ENO, ENAME\}$$

$$\{DNO\} + = \{DNO, DNAME, DLOCATION\}$$

CLOSURE

R(ABC)

$A \rightarrow B$

$B \rightarrow C$

R(ABCDEFG)

$A \rightarrow B$

$BC \rightarrow DE$

$AEG \rightarrow G$

$(AC)^+ = ?$

R(ABCDE)

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$(B)^+ = ?$

R(ABCDEF)

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$(AB)^+ = ?$

R(ABCDEFGH)

$A \rightarrow BC$

$CD \rightarrow E$

$E \rightarrow C$

$D \rightarrow AEH$

$ABH \rightarrow BD$

$DH \rightarrow BC$

$BCD \rightarrow H ?$

COVER

COVERED BY FUNCTIONAL DEPENDENCY: -

A set of functional dependency F is said to cover another set of functional dependencies E if every FD in E is also in F^+ that is if every dependency in E can be inferred from F then we can say that E is covered by F.

EQUIVALENT: -

Two sets of functional dependencies E and F are said to be equivalent if $E^+ = F^+$. Hence equivalence means that every FD in E can be inferred from F and every FD in F can be inferred from E that is E is equivalent to F if both the condition E covers F and F covers E hold.

EQUIVALENCE OF FUNCTIONAL DEPENDENCY

R(ACDEH)

F: $A \rightarrow C$

$AC \rightarrow D$

$E \rightarrow AD$

$E \rightarrow H$

F: $(A)^+ = ACD$

$(AC)^+ \rightarrow ACD$

$(E)^+ \rightarrow ACDEH$

G: $A \rightarrow CD$

$E \rightarrow AH$

G: $(A)^+ = ACD$

$(E)^+ \rightarrow ACDEH$

$F \supseteq G$

$F \subseteq G$

$F = G$

$F \neq G$

Here we can say that F is equivalent to G

EQUIVALENCE OF FUNCTIONAL DEPENDENCY

F: {

$A \rightarrow B$

$AB \rightarrow C$

$D \rightarrow AC$

$D \rightarrow E$

}

G: {

$A \rightarrow BC$

$D \rightarrow AB$

}

$F \supseteq G$

$F \subseteq G$

$F = G$

$F \neq G$

Findout wheather F covers G or G covers F or both are equivalent

Canonical Cover of Functional Dependencies:

- ❖ Whenever a user updates the database, the system must check whether any of the functional dependencies are getting violated in this process. If there is a violation of dependencies in the new database state, the system must roll back. Working with a huge set of functional dependencies can cause unnecessary added computational time. This is where the canonical cover comes into play.
- ❖ A canonical cover of a set of functional dependencies F is a simplified set of functional dependencies that has the same closure as the original set F.
- ❖ **Extraneous attributes:** An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

Canonical Cover of Functional Dependencies:

Canonical cover: A canonical cover F_c of a set of functional dependencies F such that ALL the following properties are satisfied:

- ❖ F logically implies all dependencies in F_c .
- ❖ F_c logically implies all dependencies in F .
- ❖ No functional dependency in F_c contains an extraneous attribute.

Each left side of a functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in such that $\alpha_1 \rightarrow \alpha_2$.

Canonical Cover of Functional Dependencies:

Problem-

The following functional dependencies hold true for the relational scheme R (W , X , Y , Z)

$$X \rightarrow W$$

$$WZ \rightarrow XY$$

$$Y \rightarrow WXZ$$

Write the irreducible equivalent for this set of functional dependencies.

Solution

Step-01:

Write all the functional dependencies such that each contains exactly one attribute on its right side-

$$X \rightarrow W$$

$$WZ \rightarrow X$$

$$WZ \rightarrow Y$$

$$Y \rightarrow W$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Step-02:

Check the essentiality of each functional dependency one by one.

For $X \rightarrow W$:

Considering $X \rightarrow W$, $(X)^+ = \{ X, W \}$

Ignoring $X \rightarrow W$, $(X)^+ = \{ X \}$

Now, clearly, the two results are different.

Thus, we conclude that $X \rightarrow W$ is essential and can not be eliminated.

Solution

For $WZ \rightarrow X$:

Considering $WZ \rightarrow X$, $(WZ)^+ = \{ W, X, Y, Z \}$

Ignoring $WZ \rightarrow X$, $(WZ)^+ = \{ W, X, Y, Z \}$

Now,

Clearly, the two results are same.

Thus, we conclude that $WZ \rightarrow X$ is non-essential and can be eliminated.

Eliminating $WZ \rightarrow X$, our set of functional dependencies reduces to-

$$\begin{aligned} X &\rightarrow W \\ WZ &\rightarrow X \\ WZ &\rightarrow Y \\ Y &\rightarrow W \\ Y &\rightarrow X \\ Y &\rightarrow Z \end{aligned}$$

Now, we will consider this reduced set in further checks.

$$\begin{aligned} X &\rightarrow W \\ WZ &\rightarrow Y \\ Y &\rightarrow W \\ Y &\rightarrow X \\ Y &\rightarrow Z \end{aligned}$$

Solution

For $WZ \rightarrow Y$:

Considering $WZ \rightarrow Y$, $(WZ)^+ = \{ W, X, Y, Z \}$

Ignoring $WZ \rightarrow Y$, $(WZ)^+ = \{ W, Z \}$

$$X \rightarrow W$$

$$WZ \rightarrow X$$

$$WZ \rightarrow Y$$

$$Y \rightarrow W$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Now,

Clearly, the two results are different.

Thus, we conclude that $WZ \rightarrow Y$ is essential and can not be eliminated.

Solution

For $Y \rightarrow W$:

Considering $Y \rightarrow W$, $(Y)^+ = \{W, X, Y, Z\}$

Ignoring $Y \rightarrow W$, $(Y)^+ = \{W, X, Y, Z\}$

$$\begin{aligned}X &\rightarrow W \\WZ &\rightarrow X \\WZ &\rightarrow Y \\Y &\rightarrow W \\Y &\rightarrow X \\Y &\rightarrow Z\end{aligned}$$

Now,

Clearly, the two results are same.

Thus, we conclude that $Y \rightarrow W$ is non-essential and can be eliminated.

$$\begin{aligned}X &\rightarrow W \\WZ &\rightarrow Y \\Y &\rightarrow X \\Y &\rightarrow Z\end{aligned}$$

Eliminating $Y \rightarrow W$, our set of functional dependencies reduces to-

Solution

For $Y \rightarrow X$:

Considering $Y \rightarrow X, (Y)^+ = \{ W, X, Y, Z \}$

Ignoring $Y \rightarrow X, (Y)^+ = \{ Y, Z \}$

Now,

Clearly, the two results are different.

Thus, we conclude that $Y \rightarrow X$ is essential and can not be eliminated.

For $Y \rightarrow Z$:

Considering $Y \rightarrow Z, (Y)^+ = \{ W, X, Y, Z \}$

Ignoring $Y \rightarrow Z, (Y)^+ = \{ W, X, Y \}$

Now,

Clearly, the two results are different.

Thus, we conclude that $Y \rightarrow Z$ is essential and can not be eliminated.

From here, our essential functional dependencies are-

$X \rightarrow W$

$WZ \rightarrow Y$

$Y \rightarrow X$

$Y \rightarrow Z$

Solution

Step-03:

Consider the functional dependencies having more than one attribute on their left side.
Check if their left side can be reduced.

In our set,

Only $WZ \rightarrow Y$ contains more than one attribute on its left side.

Considering $WZ \rightarrow Y$, $(WZ)^+ = \{ W, X, Y, Z \}$

Now,

Consider all the possible subsets of WZ .

Check if the closure result of any subset matches to the closure result of WZ .

$$(W)^+ = \{ W \}$$

$$(Z)^+ = \{ Z \}$$

Solution

Clearly,

None of the subsets have the same closure result same as that of the entire left side.

Thus, we conclude that we can not write $WZ \rightarrow Y$ as $W \rightarrow Y$ or $Z \rightarrow Y$.

Thus, set of functional dependencies obtained in step-02 is the canonical cover.

Finally, the canonical cover is-

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Canonical Cover

Canonical Cover of Functional Dependencies:

Example1:

Consider the following set F of functional dependencies:

$$\begin{aligned} F = & \{ \\ & A \rightarrow BC \\ & B \rightarrow C \\ & A \rightarrow B \\ & AB \rightarrow C \\ & \} \end{aligned}$$

Example2:

Consider another set F of functional dependencies:

$$\begin{aligned} F = & \{ \\ & A \rightarrow BC \\ & CD \rightarrow E \\ & B \rightarrow D \\ & E \rightarrow A \\ & \} \end{aligned}$$

KEYS

- ❖ Generally a column has a name but the row does not
- ❖ To find any specific data in any row we need to refer some value
- ❖ A key is a attribute or a set of attribute which is ~~A→B~~ capable of finding data in any row without any conflict.
- ❖ For ex: in the given table we hold some functional dependencies like

$A \rightarrow BC$

$BC \rightarrow A$

- ❖ There is no difference between the key and a super key

A	B	C
1	a	x
2	b	Y
3	b	X
4	C	y

KEYS

- ❖ A super-key is a key if it can find every attribute of the relation.
- ❖ If a proper subset of a super-key is a super-key than its not a candidate key.
- ❖ A super-key is a key if it can find every attribute of the relation.
- ❖ If a proper subset of a super-key is a super-key than its not a candidate key.
- ❖ A minimal super-key is a candidate key

R(ABCD)		
FD	SK	CK
$A \rightarrow BCD$	✓	✓
$AB \rightarrow CD$	✓	X
$ABC \rightarrow D$	✓	X
$BD \rightarrow AC$	✓	✓
$C \rightarrow AD$	X	X

FINDING CANDIDATE KEYS

R(ABCDEFGH)

AB→C
A→DE
B→F
F→GH

R(ABCDEFGH)

AB→C
BD→EF
AD→G
A→H

R(ABCDE)

BC→ADE
D→B

R(ABCDE)

AB→CD
D→E
BC→DE

R(WXYZ)

Z→W
Y→XZ
WX→Y

R(ABCDEFGH)

CH→G
A→BC
B→CFH
E→A

Reason for Normalization

- ✓ Insertion, deletion and updation anomalies.

STUDENT						
ROLL NO	NAME	AGE	BRANCH ID	BRANCH NAME	HOD NAME	HOD PH
1	A	20	121	CS	XYZ	123
2	B	21	121	CS	XYZ	123
3	C	19	121	CS	XYZ	123

Normalization

- ❖ Normalization is the process of efficiently organizing data by analyzing and decomposing the complex relation to form a simple relation.
- ❖ Normalization is the process of analyzing the given relation schemas based on their functional dependencies and primary keys to achieve the desired properties like
 - ✓ Minimizing redundancy
 - ✓ Minimizing the insertion, deletion and updation anomalies.
- ❖ Normalization is carried out for the following reasons: -
 - ✓ To structure the data between relations as maintenance is simplified.
 - ✓ To allow data retrieval at optimal speed.
 - ✓ To reduce the need to restructure relations as application requirements arise.
 - ✓ To improve the quality of design for an application
 - ✓ To minimize the insertion, deletion, and update anomalies.
 - ✓ To structure the data between relations so that data redundancy can be minimized

FIRST NORMAL FORM (1NF): -

❖ A relation is said to be in first normal form, if the domain of an attribute must include only atomic (simple, individual) values.

❖ It disallows multi-valued and composite attribute.

❖ Steps to convert the relation into 1NF: -

- ✓ Eliminate duplicate columns from the same relation.
- ✓ Create a separate relation for each set of related data and identify each set of related data with a primary key.

ROLL	NAME	COURSE
001	RAKESH	OS CN
002	RAHUL	DBMS CO



ROLL	NAME	COURSE
001	RAKESH	OS
001	RAKESH	CN
002	RAHUL	DBMS
002	RAHUL	CO

SECOND NORMAL FORM (2NF): -

❖ A relation is in 2NF if: -

✓ It is in 1NF

R(A B C D)

✓ It includes only that attribute which is fully dependent on its primary key.

AB → D
B → C

❖ Steps to convert a relation to its 2NF is: -

✓ Find and remove the fields that are related to the only part of the primary key.

✓ Group the removed items in another relation.

✓ Assign the new relation with a primary key

Definition: A relation schema R is in 2NF if every nonprime attribute A in R is fully dependent on the primary key of R.

3RD NORMAL FORM(3NF): -

❖ A relation is said to be in 3NF if: -

- ✓ It is in 2NF and
- ✓ There exists no transitive functional dependency.

$R(A B C D)$

$AB \rightarrow C$
 $C \rightarrow D$

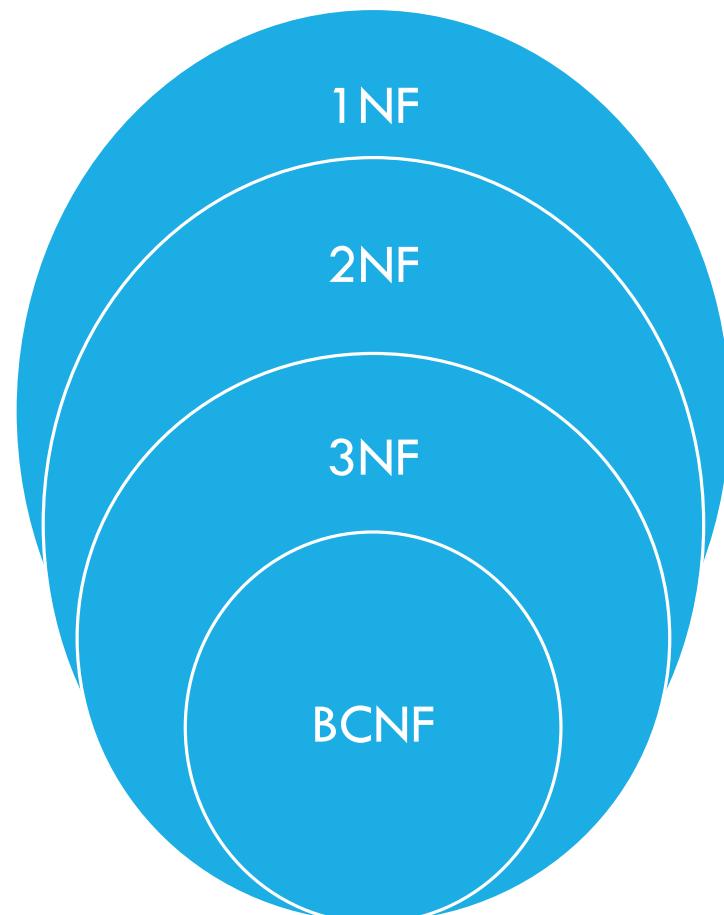
Definition: A relation schema R is in 3NF if it satisfies 2NF and no nonprime attributes of R is transitively dependent on the set of primary key.

Transitive dependency: $X \rightarrow Y$ in a relational schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R and both $X \rightarrow Z$ and $Z \rightarrow Y$ holds .

BOYCE –CODD NORMAL FORM: -(BCNF)

❖ A relation is said to be BCNF if and only if all the determinants are candidate keys.

Restrictions	
2NF	$P \rightarrow NP$
3NF	$NP \rightarrow NP$
BCNF	$P/NP \rightarrow P$



$R(A B C)$
 $AB \rightarrow C$
 $C \rightarrow B$

BOYCE – CODD NORMAL FORM: -(BCNF)

$R(A B C D)$

$AB \rightarrow C$
 $C \rightarrow B$

R		
A	B	C
a	1	x
b	2	y
c	2	z
c	3	w
d	3	w
e	3	w

BCNF



A	C
a	x
b	y
c	z
c	w
d	w
e	w

C	B
x	1
y	2
z	2
w	3

Restrictions	
2NF	$P \rightarrow NP$
3NF	$NP \rightarrow NP$
BCNF	$P / NP \rightarrow P$

IDENTIFYING NORMAL FORM

R(A B C D E F G H)

AB→C
A→DE
B→F
F→GH

R(A B C D E F G H)

AB→C
BD→EF
AD→GH
A→I

R(A B C D E)

CE→D
D→B
C→A

R(A B C D E F)

AB→C
DC→AE
E→F

R(A B C D E)

AB→CD
D→A
BC→DE

R(A B C D E)

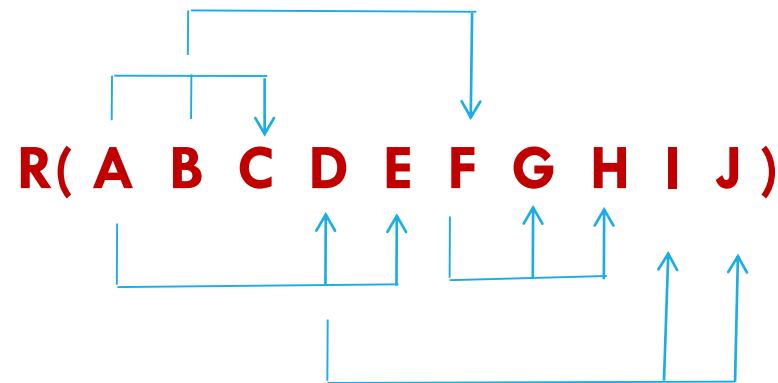
BC→ADE
D→B

AB, ABD, CE, ABD-BCD, AB-BD-BC, BC-CD

Restrictions

Restrictions	
2NF	P→NP
3NF	NP→NP
BCNF	P/NP→P

HOW TO NORMALIZE A RELATION TABLE?



$R(A, B, C, D, E, F, G, H, I, J)$

$AB \rightarrow C$

$A \rightarrow DE$

$B \rightarrow F$

$F \rightarrow GH$

$D \rightarrow IJ$

RELATIONAL DECOMPOSITION:

❖ For a good database design the database schema must satisfy the normal form by decomposing the relation schema and must satisfy certain properties: -

- Dependency preservation property
- Lossless / non additive join property

Dependency preservation property: -

- ❖ By using the functional dependencies, the universal relation schema R is divided/decomposed into a set of relation schema $D = \{R_1, R_2, \dots, R_n\}$
- ❖ This property states that each functional dependency $X \rightarrow Y$ specified in F that appeared directly in one of the relation schema R must be appeared in one of the relation in the decomposition D, So that no dependency are lost.
- ❖ It is sufficient that the union of the dependencies that hold in the individual relation in D be equivalent to f of R.

Suppose

For a relation $R = \{A_1, A_2, \dots, A_n\}$, The set of FD in F ($X \rightarrow Y$) Decomposition (D) = $\{R_1, R_2, \dots, R_n\}$

Then the property says that

$$(R_1(F) \cup R_2(F) \cup \dots \cup R_n(F))^+ = F^+$$

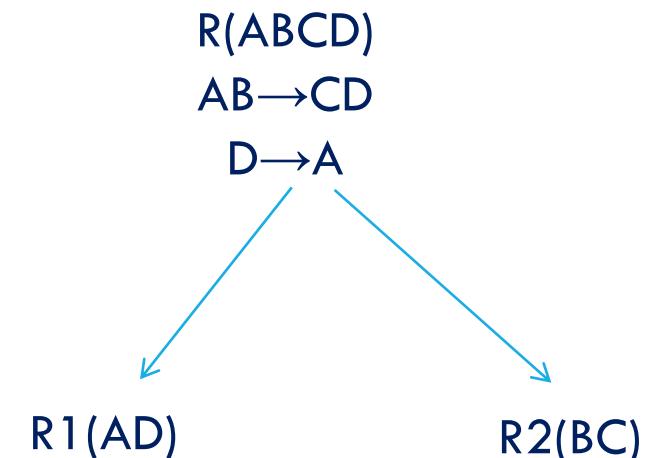
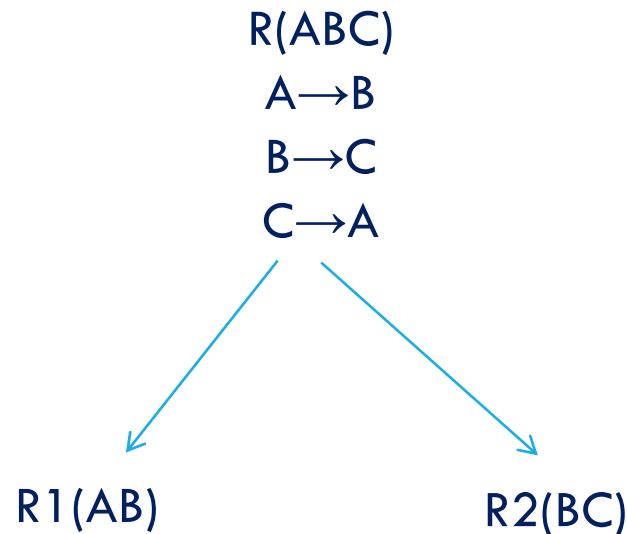
Examples for Decomposition Property

❖ If a table having FD set of F , is decomposed into two tables R_1 and R_2 having FD set F_1 and F_2 than,

$$F_1 \subseteq F^+$$

$$F_2 \subseteq F^+$$

$$(F_1 \cup F_2)^+ = F^+$$



Lossless/non additive join property:

- ❖ This property ensures that no spurious tuples are generated when a natural join operation is applied to the relations in the decomposition.
- ❖ Let R be the relation schema,

- F be the set of functional dependencies on R . R_1 and R_2 be the decomposition of R
- r be the relation instance with schema R

Then we can say that the decomposition is a loss less decomposition if

$$R_1 \bowtie R_2 = R$$

- ❖ If we project r onto R_1 and R_2 and compute the natural join of the projection results, we get back exactly r .
- ❖ The decomposition that is not a lossless decomposition is called as lossy decomposition.

Lossless JOIN Dependencies:

- ❖ This property ensures that the extra or less tuple generation problem doesn't occur after decomposition.
- ❖ If a relation R is decomposed into two relations R1 and R2, then it will be lossless iff

- $\text{Attribute}(R1) \cup \text{attribute}(R2) = \text{attribute}(R)$
- $\text{Attribute}(R1) \cap \text{attribute}(R2) \neq \emptyset$
- $\text{Attribute}(R1) \cap \text{attribute}(R2) \rightarrow \text{Attribute}(R1)$

OR

- $\text{Attribute}(R1) \cap \text{attribute}(R2) \rightarrow \text{Attribute}(R2)$

A	B	C	D
1	a	p	x
2	b	q	y

A	B
1	a
2	b

C	D
p	x
q	y

Lossless JOIN Dependencies:

- ❖ $\text{Attribute}(R1) \cup \text{attribute}(R2) = \text{attribute}(R)$
- ❖ $\text{Attribute}(R1) \cap \text{attribute}(R2) \neq \emptyset$
- ❖ $\text{Attribute}(R1) \cap \text{attribute}(R2) \rightarrow \text{Attribute}(R1)$

OR

- ❖ $\text{Attribute}(R1) \cap \text{attribute}(R2) \rightarrow \text{Attribute}(R2)$

A	B	C
1	a	p
2	b	q
3	a	r

A	B
1	a
2	b
3	a

B	C
a	p
b	q
a	r

Lossless JOIN Dependencies:

A	B	C	D	E
a	122	1	p	w
b	234	2	q	x
a	568	1	r	y
c	347	3	s	z

$R_1(AB), R_2(CD)$
 $R_1(ABC), R_2(DE)$
 $R_1(ABC), R_2(CDE)$
 $R_1(ABCD), R_2(ACDE)$

$R(VWXYZ)$
 $Z \rightarrow Y$
 $Y \rightarrow Z$
 $X \rightarrow YV$
 $VW \rightarrow X$

$R_1(VWX), R_2(XYZ)$
 $R_1(VW), R_2(YZ)$
 $R_1(VWX), R_2(YZ)$
 $R_1(VW), R_2(WXYZ)$

- ❖ $\text{Attribute}(R_1) \cup \text{attribute}(R_2) = \text{attribute}(R)$
- ❖ $\text{Attribute}(R_1) \cap \text{attribute}(R_2) \neq \emptyset$
- ❖ $\text{Attribute}(R_1) \cap \text{attribute}(R_2) \rightarrow \text{Attribute}(R_1)$
- OR
- ❖ $\text{Attribute}(R_1) \cap \text{attribute}(R_2) \rightarrow \text{Attribute}(R_2)$

4th Normal Form :

- ❖ If two or more independent relation are kept in a single relation or we can say multivalue dependency occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.
- ❖ A multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third.
- ❖ For a dependency $A \rightarrow B$, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for $A \rightarrow\!\!\!> B$ multivalued dependency.

For example,

Person->-> mobile,
Person ->-> food_likes

PERSON	MOBILE	FOOD_LIKES
Mahesh	9893/9424	Burger / pizza
Ramesh	9191	Pizza

Fourth normal form (4NF):

❖ Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties – A relation R is in 4NF if and only if the following conditions are satisfied:

- It should be in the Boyce-Codd Normal Form (BCNF).
- the table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Fourth normal form (4NF):

Example – Consider the database table of a class which has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

SID	SNAME
S1	A
S2	B

R1 X R2

CID	CNAME
C1	C
C2	D

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Fifth Normal Form / Projected Normal Form (5NF):

❖ A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

Properties – A relation R is in 5NF if and only if it satisfies following conditions:

- R should be already in 4NF.
- It cannot be further non loss decomposed (join dependency)

Fifth Normal Form / Projected Normal Form (5NF):

Example – Consider the above schema, with a case as “if a company makes a product and an agent is an agent for that company, then he always sells that product for the company”. Under these circumstances, the ACP table is shown as:

AGENT	COMPANY	PRODUCT
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decompose into 3 relations. Now, the natural Join of all the three relations will be shown as:

AGENT	COMPANY	PRODUCT
A1	PQR	Nut
A1	XYZ	
A2	PQR	

COMPANY	PRODUCT
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

Storage System in DBMS

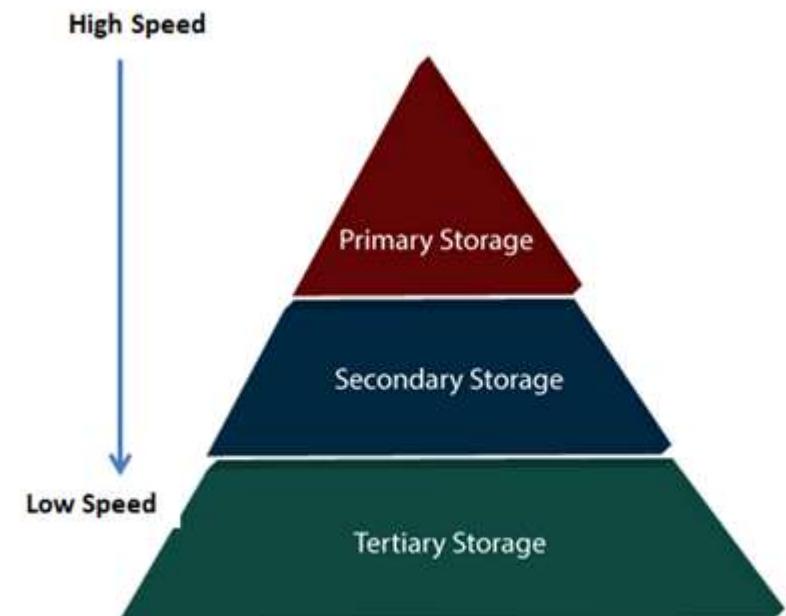
A database system provides an ultimate view of the stored data.

However, data in the form of bits, bytes get stored in different storage devices.

Types of Data Storage

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

- Primary Storage
- Secondary Storage
- Tertiary Storage



Storage System in DBMS

- **Primary Storage** – The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
- **Secondary Storage** – Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

Storage System in DBMS

- **Tertiary Storage** – Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

- **Memory Hierarchy**

A computer system has a well-defined hierarchy of memory. A CPU has direct access to its main memory as well as its inbuilt registers. The access time of the main memory is obviously less than the CPU speed. To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.

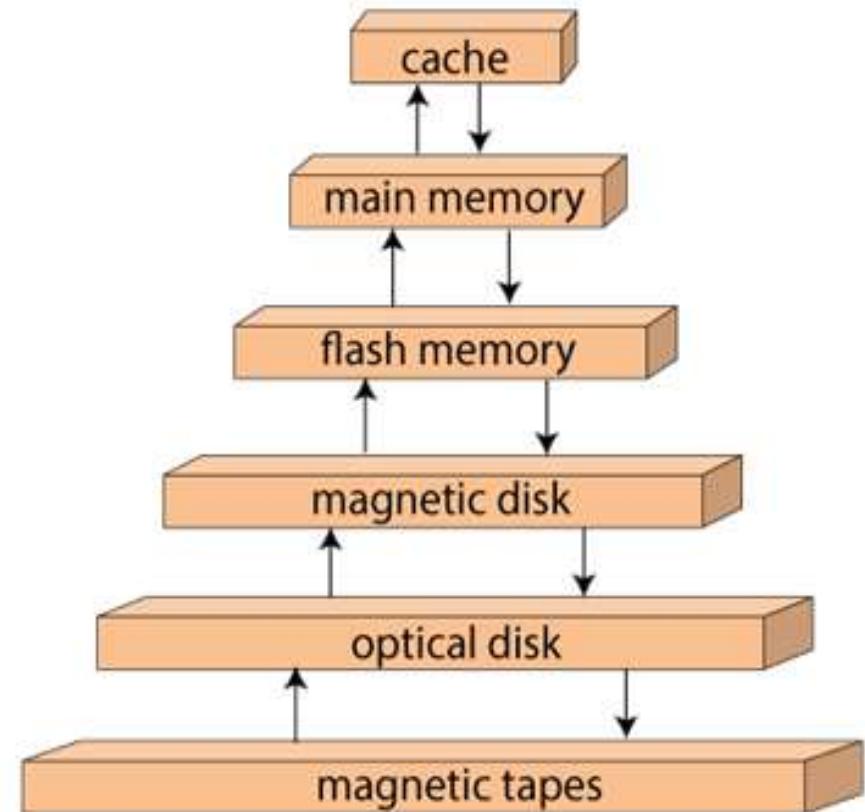
Magnetic Disks

Note: The memory with the fastest access is the costliest one. Larger storage devices offer slow speed and they are less expensive, however they can store huge volumes of data as compared to CPU registers or cache memory.

Hard disk drives are the most common secondary storage devices in present computer systems. These are called magnetic disks because they use the concept of magnetization to store information. Hard disks consist of metal disks coated with magnetizable material. These disks are placed vertically on a spindle. A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one).

Magnetic Disks

Hard disks are formatted in a well-defined order to store data efficiently. A hard disk plate has many concentric circles on it, called tracks. Every track is further divided into sectors. A sector on a hard disk typically stores 512 bytes of data.



Storage device hierarchy

File Organization

- The File is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

File Organization

Objective of file organization

- ✓ It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- ✓ To perform insert, delete or update transaction on the records should be quick and easy.
- ✓ The duplicate records cannot be induced as a result of insert, update or delete.
- ✓ For the minimal cost of storage, records should be stored efficiently.

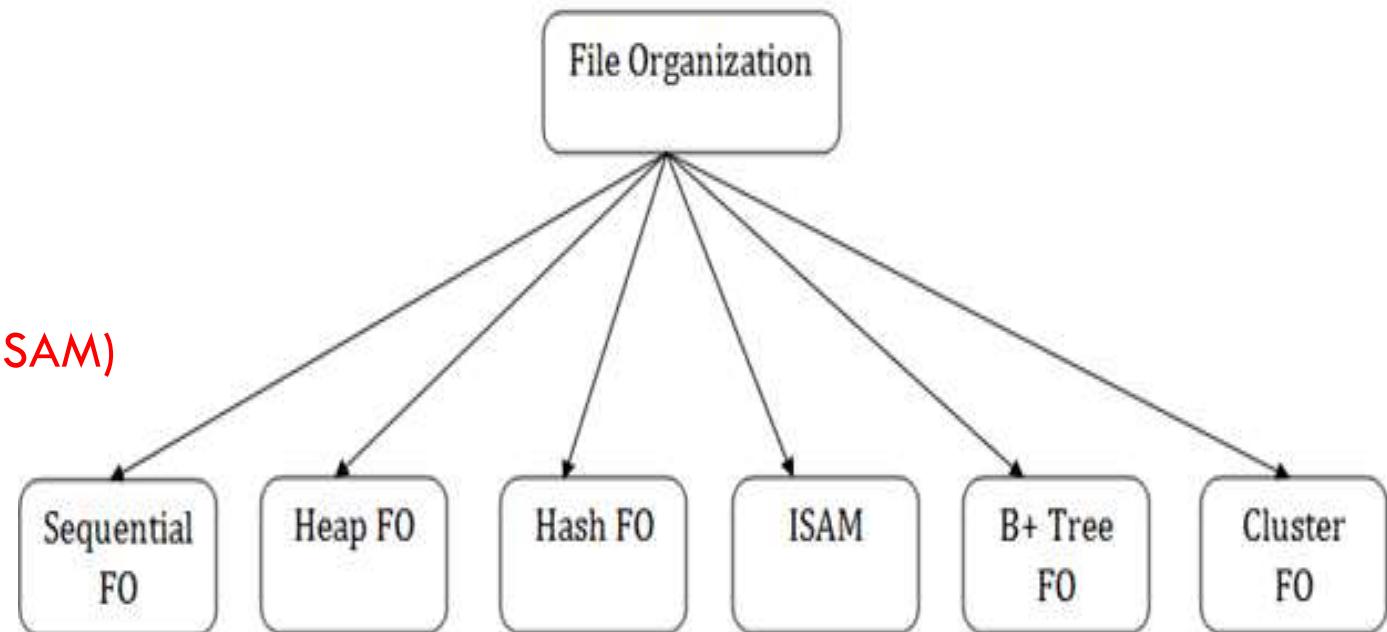
Types of file organization:

- ✓ File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

File Organization

Types of file organization are as follows:

- ✓ Sequential file organization
- ✓ Heap file organization
- ✓ Hash file organization
- ✓ B+ file organization
- ✓ Indexed sequential access method (ISAM)
- ✓ Cluster file organization



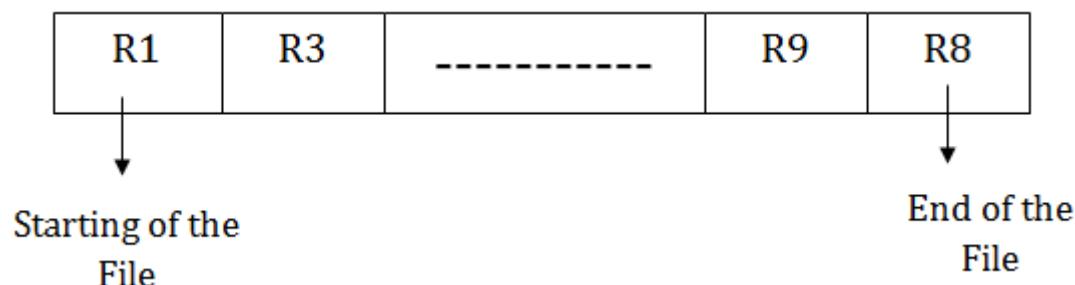
Sequential File Organization

❖ This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

1. Pile File Method:

▪ It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.

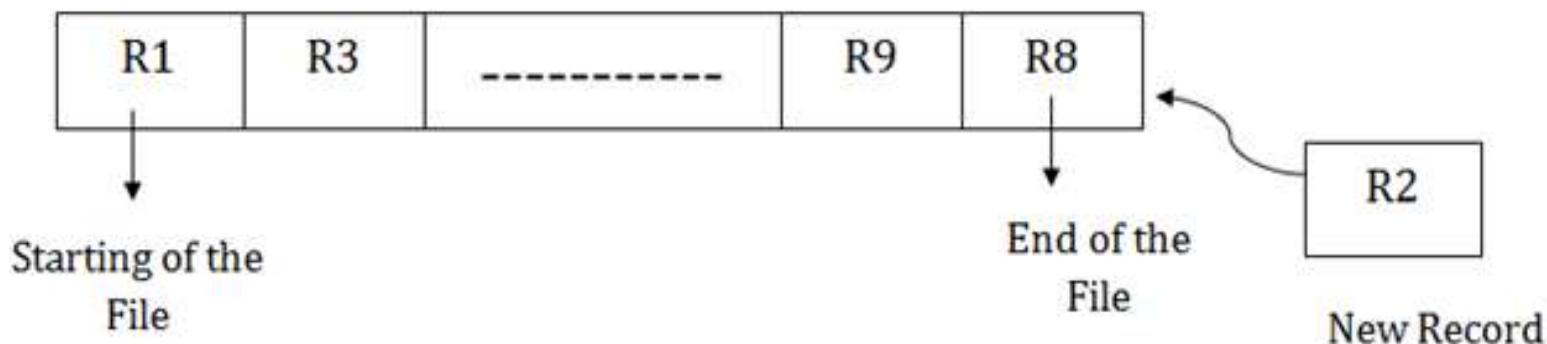
▪ In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



Sequential File Organization

Insertion of the new record:

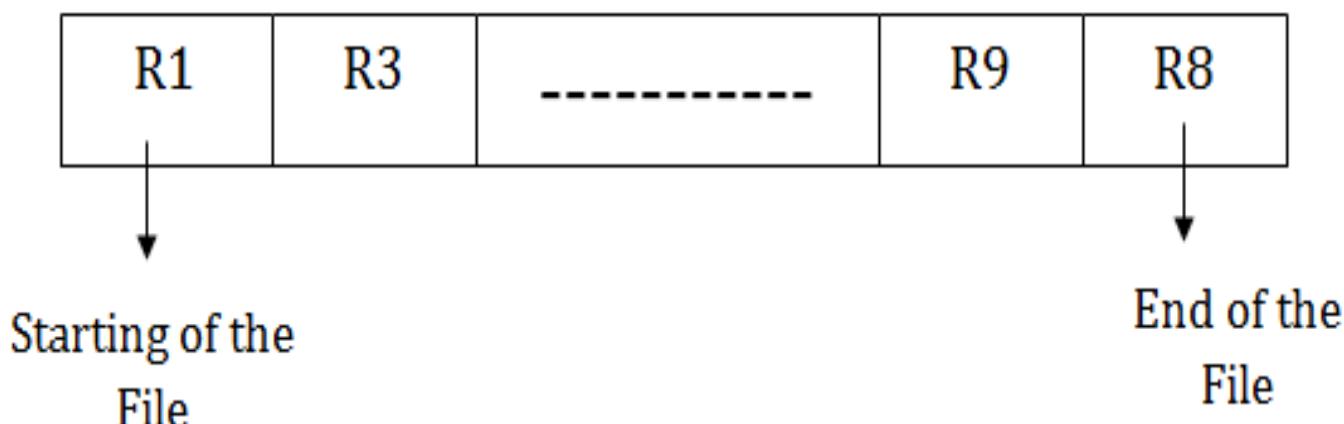
Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



Sequential File Organization

2. Sorted File Method:

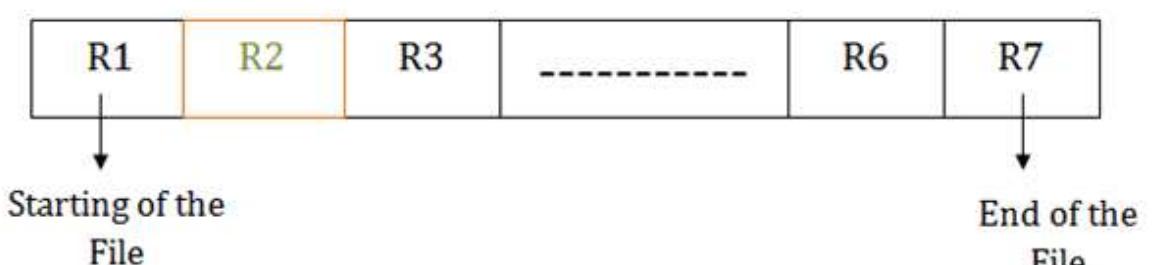
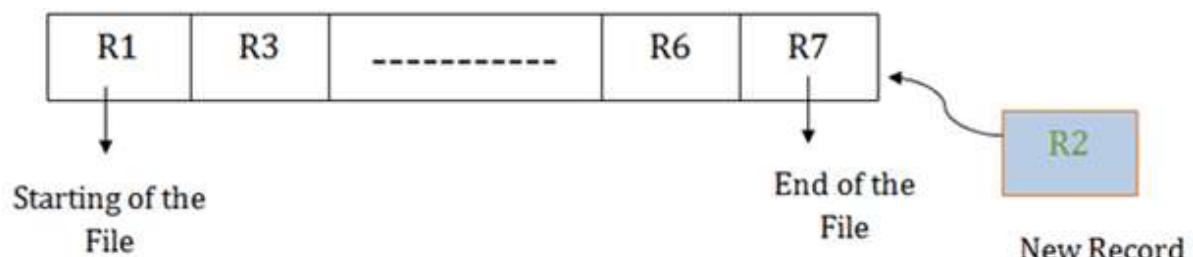
- ✓ In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- ✓ In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Sequential File Organization

Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.



Sequential File Organization

Pros of sequential file organization

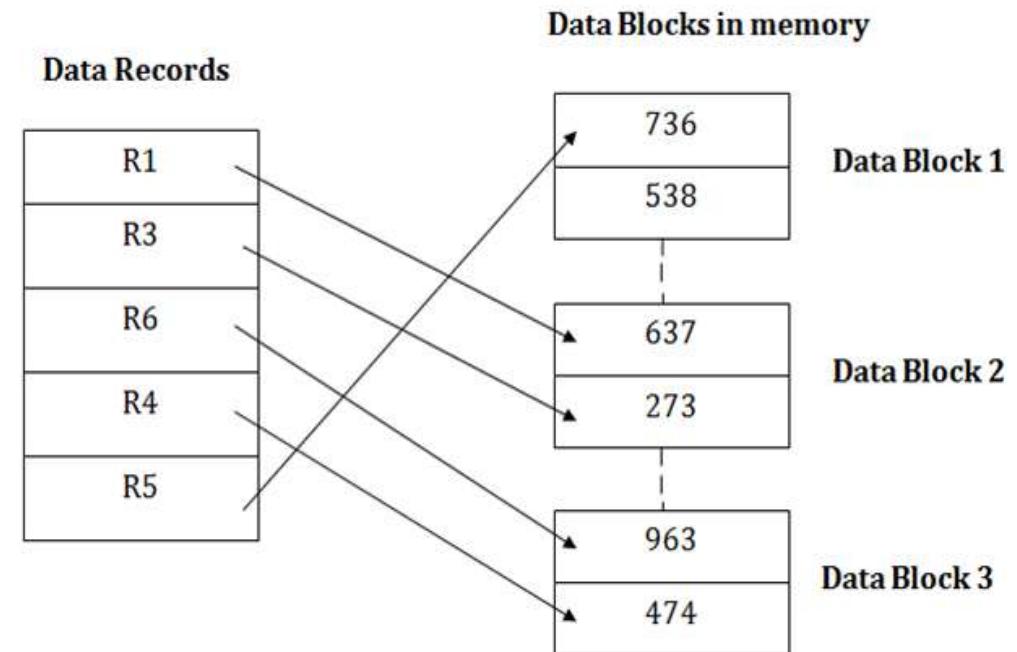
- ✓ It contains a fast and efficient method for the huge amount of data.
- ✓ In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- ✓ It is simple in design. It requires no much effort to store the data.
- ✓ This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- ✓ This method is used for report generation or statistical calculations.

Cons of sequential file organization

- ✓ It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- ✓ Sorted file method takes more time and space for sorting the records.

Heap file organization

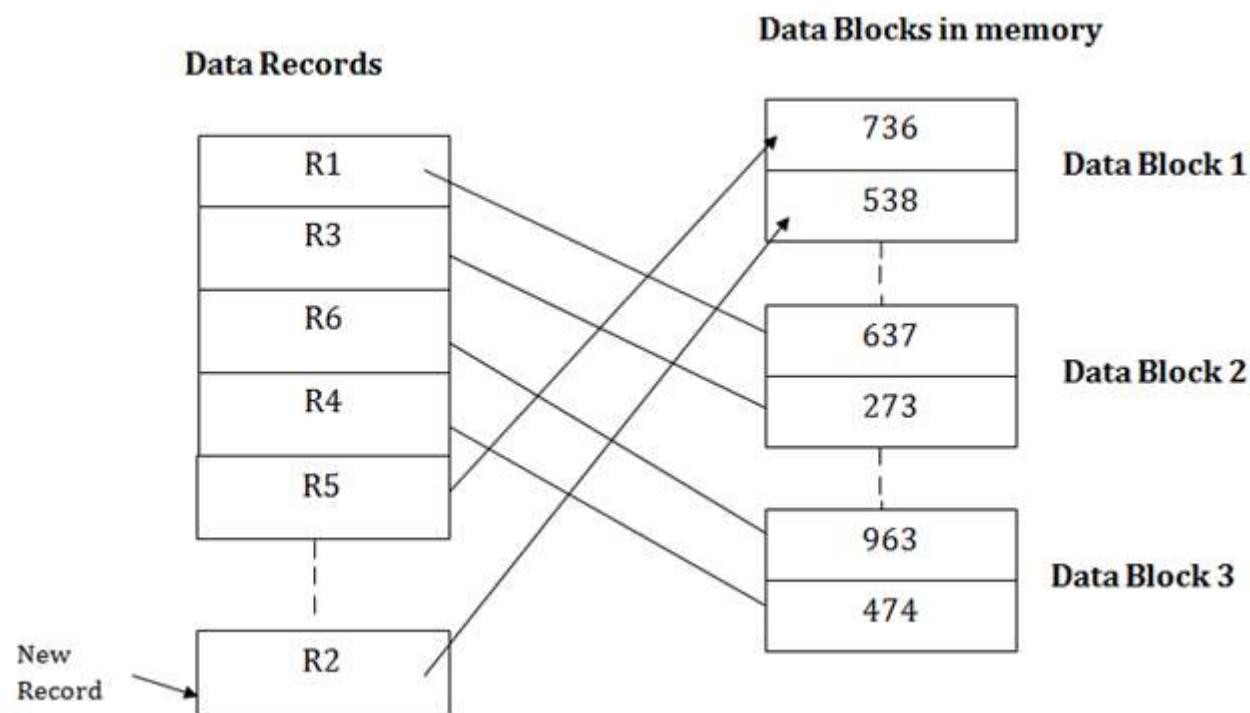
- ❖ It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- ❖ When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- ❖ In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Heap file organization

Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



- ✓ If we want to search, update or delete the data in heap file organization, then we need to traverse the data from starting of the file till we get the requested record.
- ✓ If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Heap file organization

Pros of Heap file organization

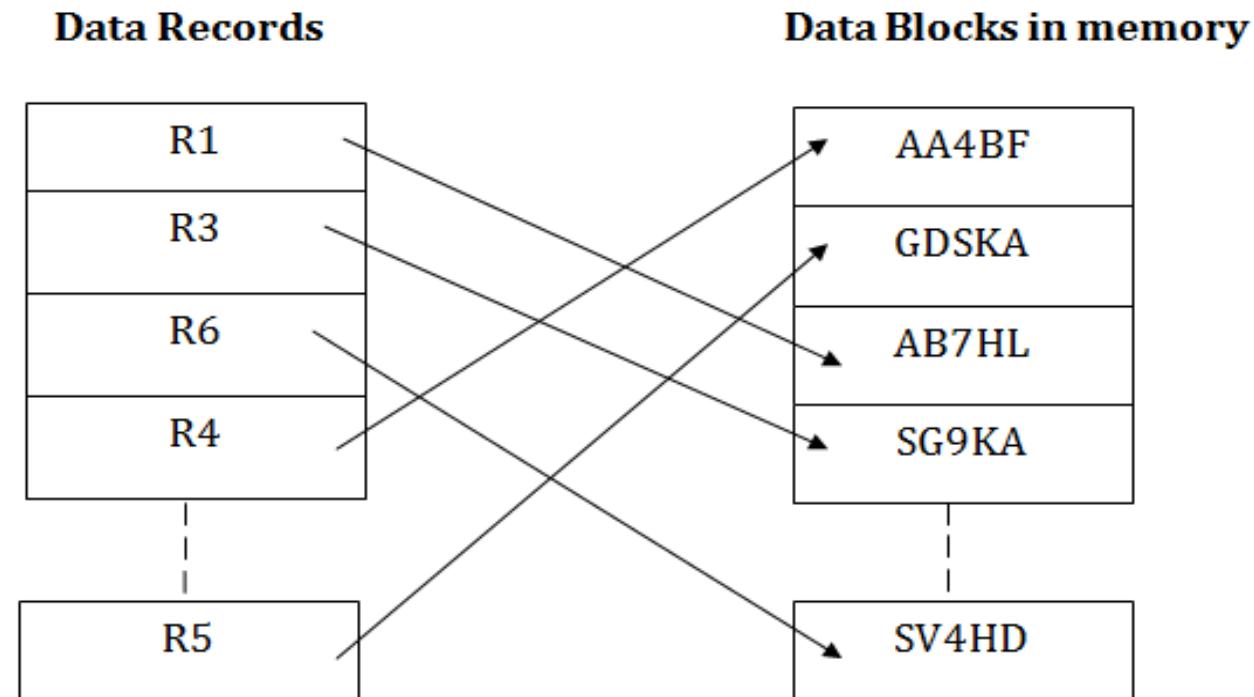
- ✓ It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- ✓ In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

- ✓ This method is inefficient for the large database because it takes time to search or modify the record.
- ✓ This method is inefficient for large databases.

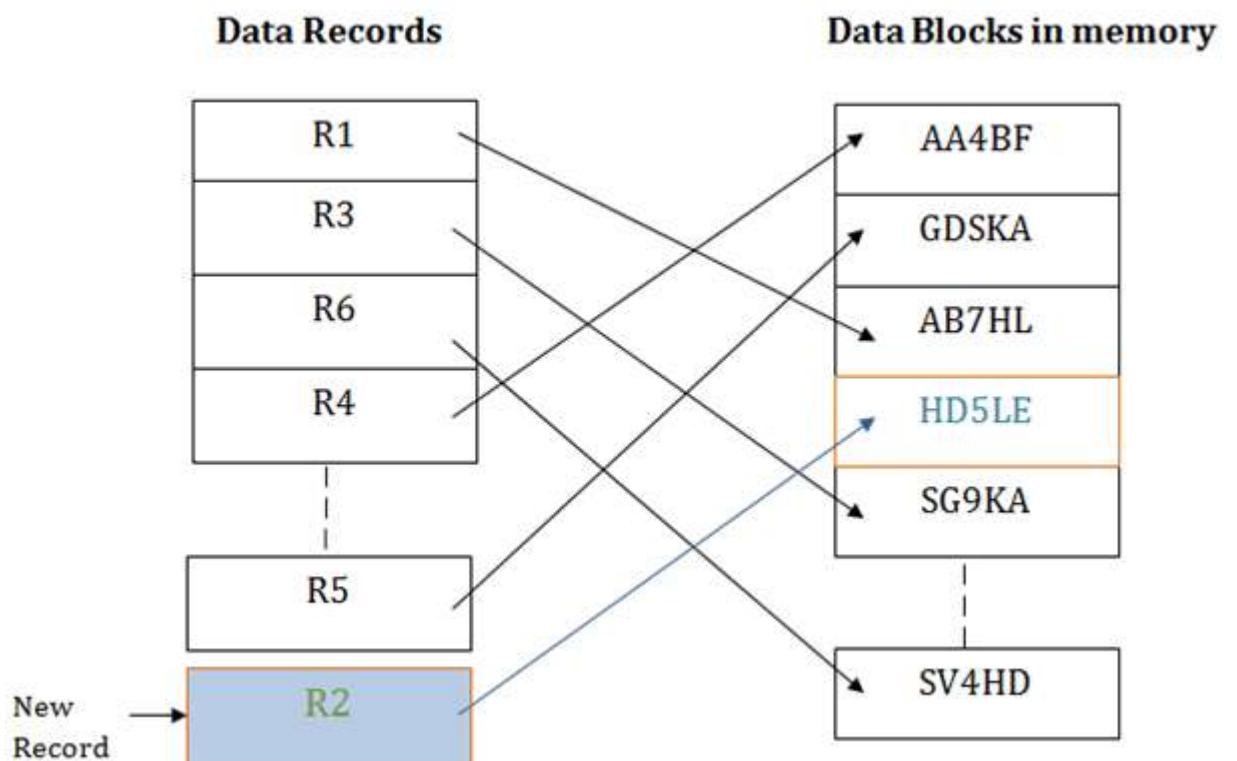
Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



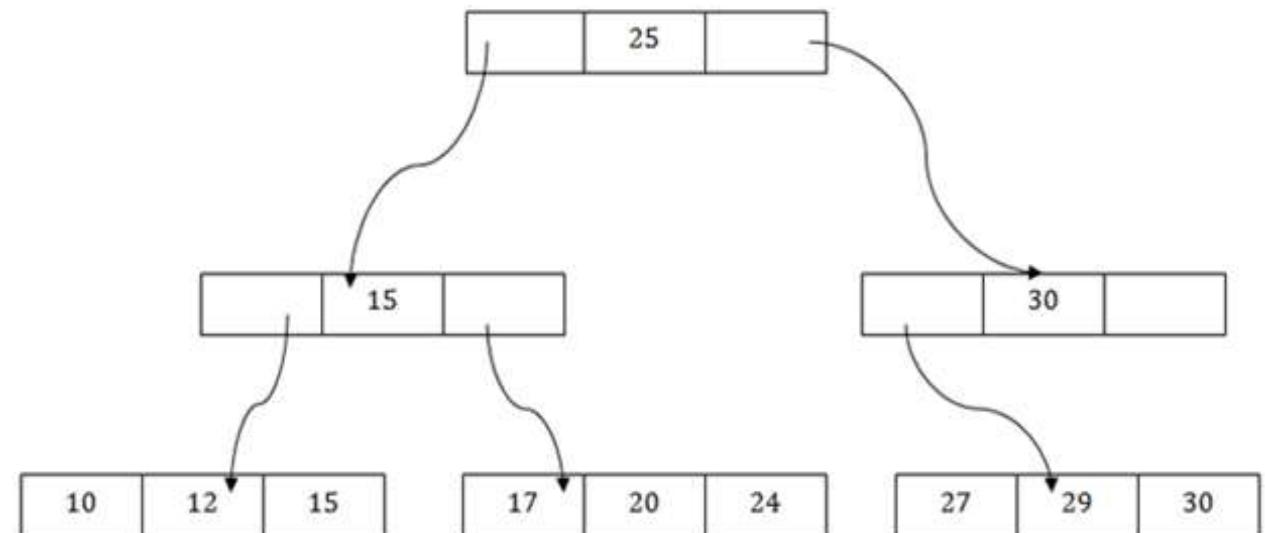
Hash File Organization

- ❖ When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.
- ❖ In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



B+ File Organization

- ❖ B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- ❖ It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- ❖ The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



B+ File Organization

The above B+ tree shows that:

- ✓ There is one root node of the tree, i.e., 25.
- ✓ There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
- ✓ The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- ✓ There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- ✓ Searching for any record is easier as all the leaf nodes are balanced.
- ✓ In this method, searching any record can be traversed through the single path and accessed easily.

B+ File Organization

Pros of B+ tree file organization

- ✓ In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- ✓ Traversing through the tree structure is easier and faster.
- ✓ The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- ✓ It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

Cons of B+ tree file organization

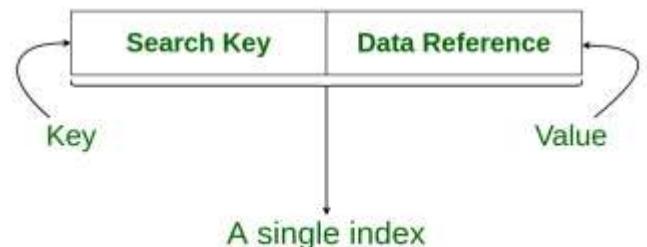
- ✓ This method is inefficient for the static method.

Indexing in Databases

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

Structure of an Index in Database



- ✓ The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
- Note: The data may or may not be stored in sorted order.
- ✓ The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.

Indexing in Databases

The indexing has various attributes:

- ✓ Access Types: This refers to the type of access such as value based search, range access, etc.
- ✓ Access Time: It refers to the time needed to find particular data element or set of elements.
- ✓ Insertion Time: It refers to the time taken to find the appropriate space and insert a new data.
- ✓ Deletion Time: Time taken to find an item and delete it as well as update the index structure.
- ✓ Space Overhead: It refers to the additional space required by the index.

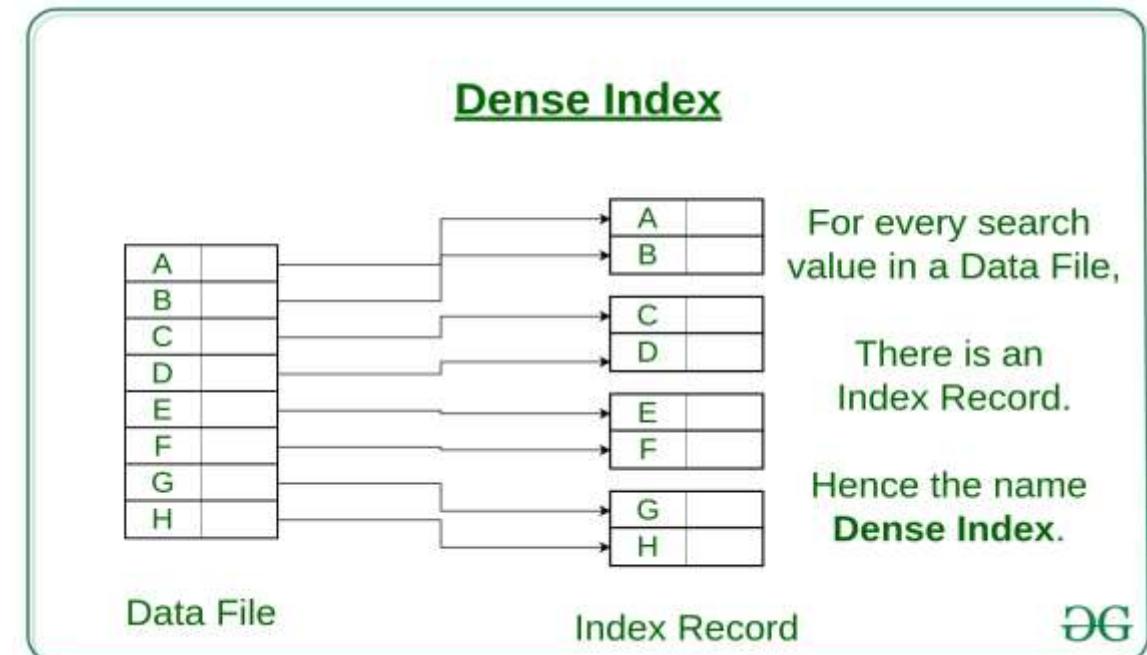
In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

Indexing in Databases

Sequential File Organization or Ordered Index File: In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

Dense Index:

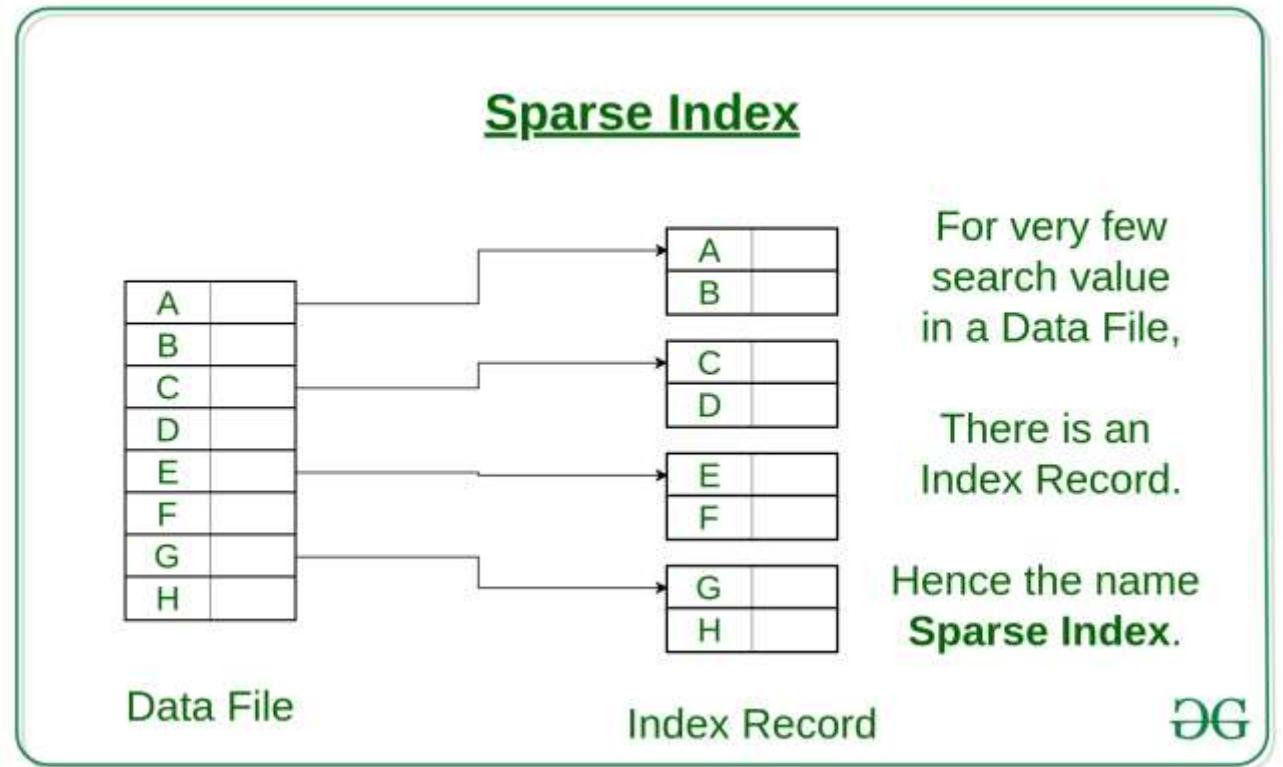
- ✓ For every search key value in the data file, there is an index record.
- ✓ This record contains the search key and also a reference to the first data record with that search key value.



Indexing in Databases

Sparse Index:

- ✓ The index record appears only for a few items in the data file. Each item points to a block as shown.
- ✓ To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- ✓ We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.



Indexing in Databases

There are primarily three methods of indexing:

- ✓ Clustered Indexing
- ✓ Non-Clustered or Secondary Indexing
- ✓ Multilevel Indexing

Clustered Indexing

✓ When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joining of more than two tables(records).

Indexing in Databases

✓ Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.

Clustered index

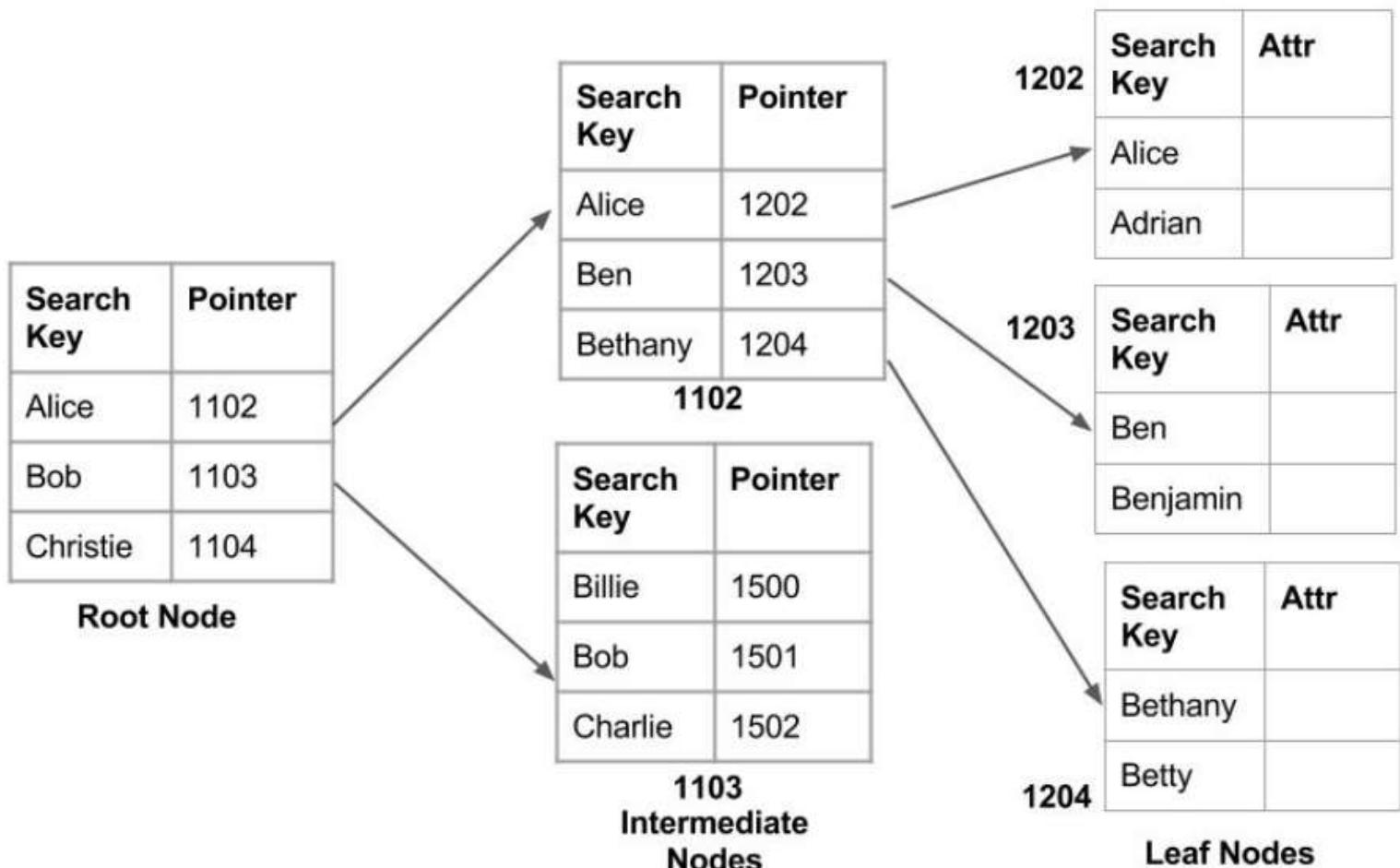
Clustered index sorted according to first name (Search key)

Primary Indexing:

✓ This is a type of Clustered Indexing where the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

Non-clustered or Secondary Indexing

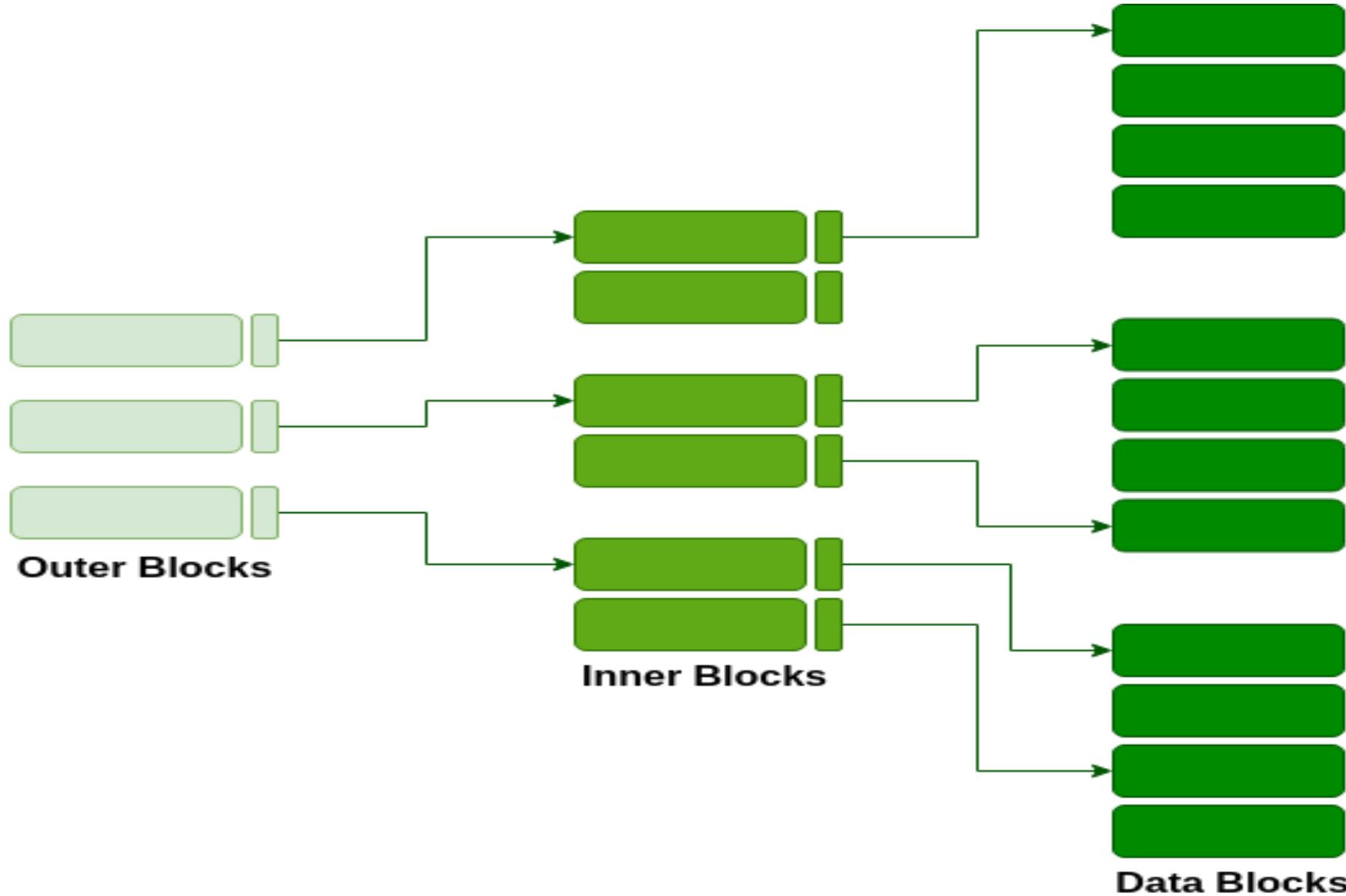
- ✓ A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of the book) is not organized but we have an ordered reference(contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly.
- ✓ It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



Non clustered index

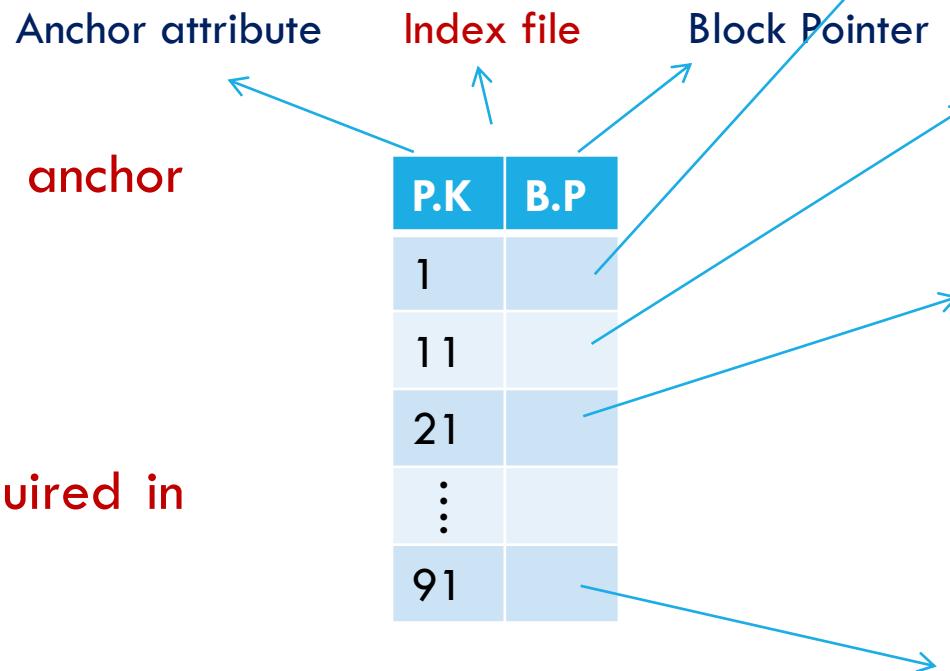
Multilevel Indexing

✓ With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can be stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.



Primary Indexing

- ✓ The main file is sorted
- ✓ Primary key is used as anchor attribute(search key)
- ✓ It's an example of sparse indexing
- ✓ No of entries = no of blocks acquired in index file by the main memory.
- ✓ No of access required = $\log 2^n + 1$



P.K	--	--	--
1			
2			
⋮			
11			
⋮			
21			
⋮			
⋮			
91			
⋮			

Clustered Indexing

- ✓ The main file is sorted(on a non-key attribute)
- ✓ There will be one entry for each unique value of the non key attribute.
- ✓ If no. of block acquired by index file is n, then block access required will be $\geq \log_2 n + 1$

Non-key attribute

Index file

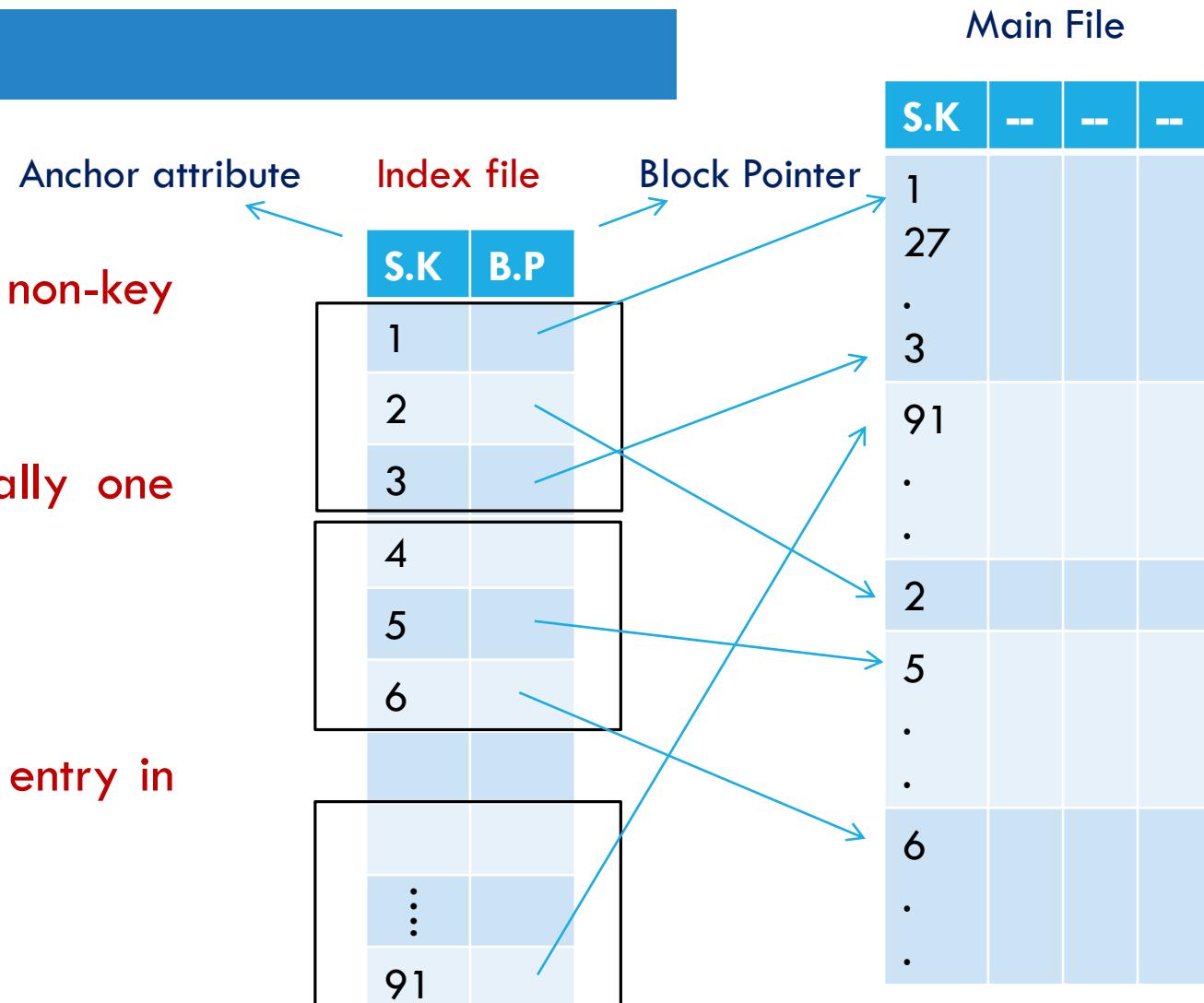
Block Pointer

N.K	B.P
1	
2	
3	
4	
5	
6	

N.K	--	--	--
1			
1			
1			
2			
3			
4			
4			
5			
5			
6			
6			
.			
.			
.			

Secondary Indexing

- ✓ The main file is un-sorted
- ✓ Can be done on key as well as non-key attribute.
- ✓ Called secondary because normally one indexing is already done.
- ✓ It's an example of dense indexing.
- ✓ No of entry in index file = no of entry in main file.
- ✓ No of Access = $[\log 2^n] + 1$



RAID (Redundant Arrays of Independent Disks)

❑ RAID, or “Redundant Arrays of Independent Disks” is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both. The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

Why data redundancy?

❑ Data redundancy, although taking up extra space, adds to disk reliability. This means, in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation. On the other hand, if the data is spread across just multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

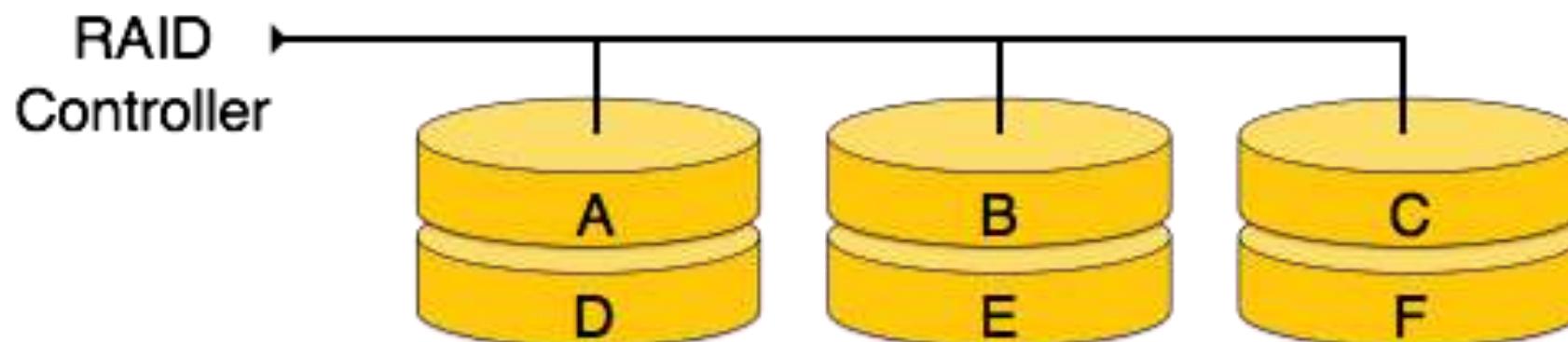
Key evaluation points for a RAID System

- Reliability: How many disk faults can the system tolerate?
- Availability: What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
- Performance: How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
- Capacity: Given a set of N disks each with B blocks, how much useful capacity is available to the user?
RAID is very transparent to the underlying system. This means, to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes in the existing code.

RAID (Redundant Arrays of Independent Disks)

RAID 0

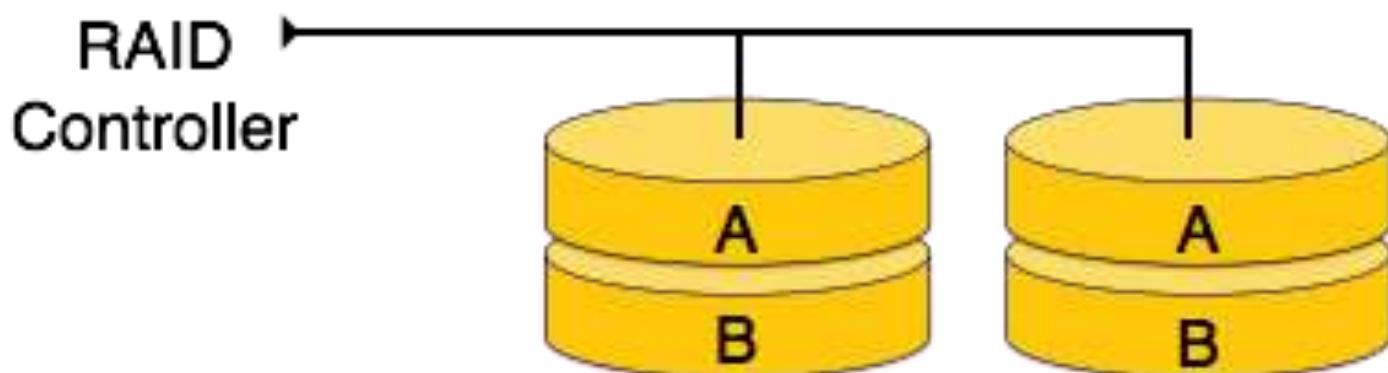
- ✓ In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.
- ✓ There is no duplication of data. Hence, a block once lost cannot be recovered.



RAID (Redundant Arrays of Independent Disks)

RAID 1

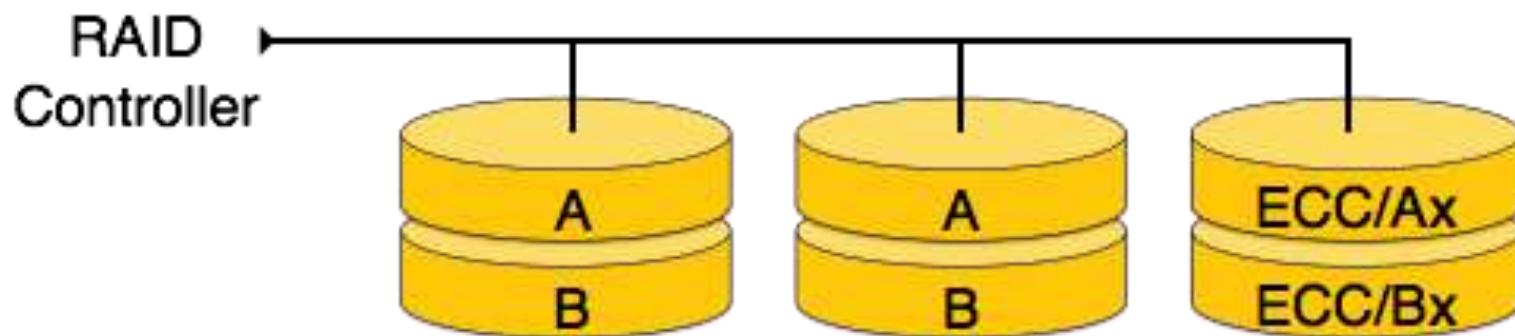
- ✓ RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called mirroring and provides 100% redundancy in case of a failure.
- ✓ 1 disk failure can be handled for certain, because blocks of that disk would have duplicates on some other disk. If we are lucky enough and disks 0 and 2 fail, then again this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, $N/2$ disk failures can be handled.



RAID (Redundant Arrays of Independent Disks)

RAID 2

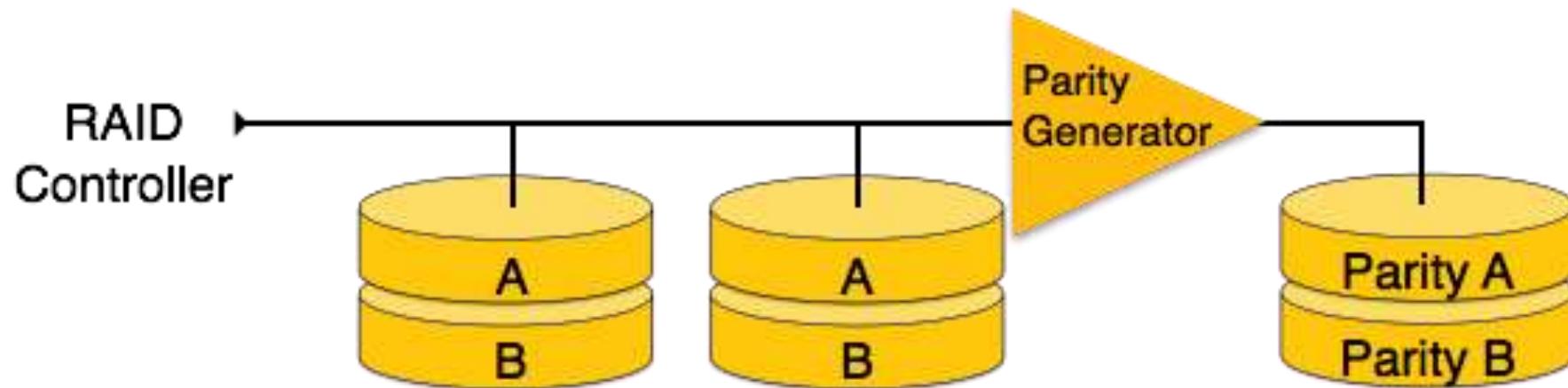
- ✓ RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.
- ✓ RAID-2 consists of bit-level striping using a Hamming Code parity.



RAID (Redundant Arrays of Independent Disks)

RAID 3

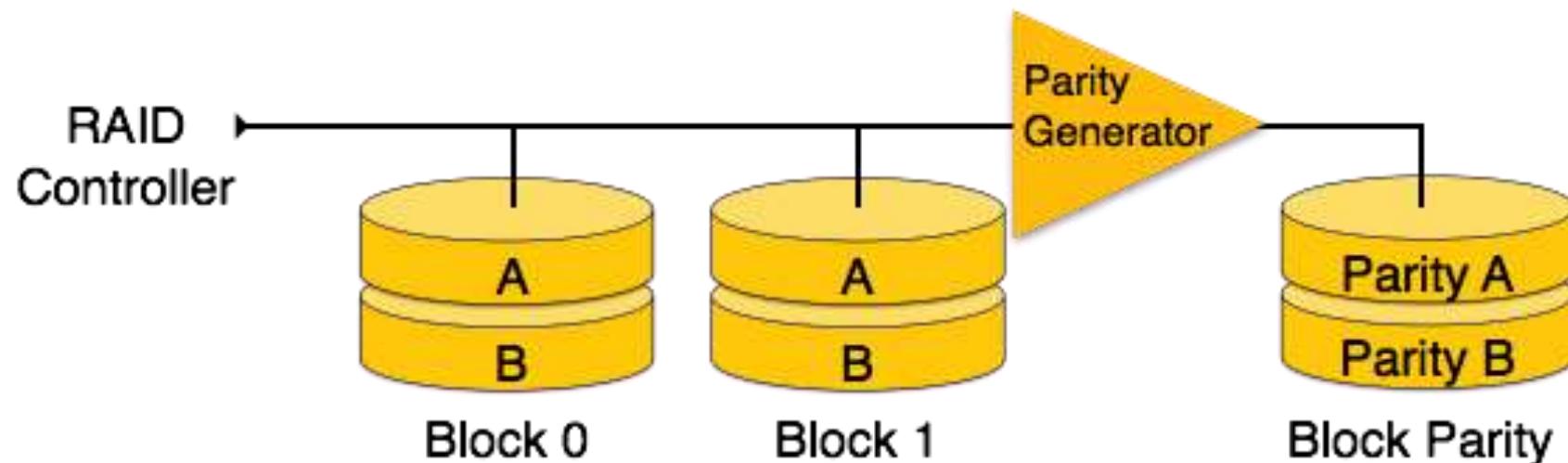
- ✓ RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.
- ✓ RAID-3 consists of byte-level striping with a dedicated parity.



RAID (Redundant Arrays of Independent Disks)

RAID 4

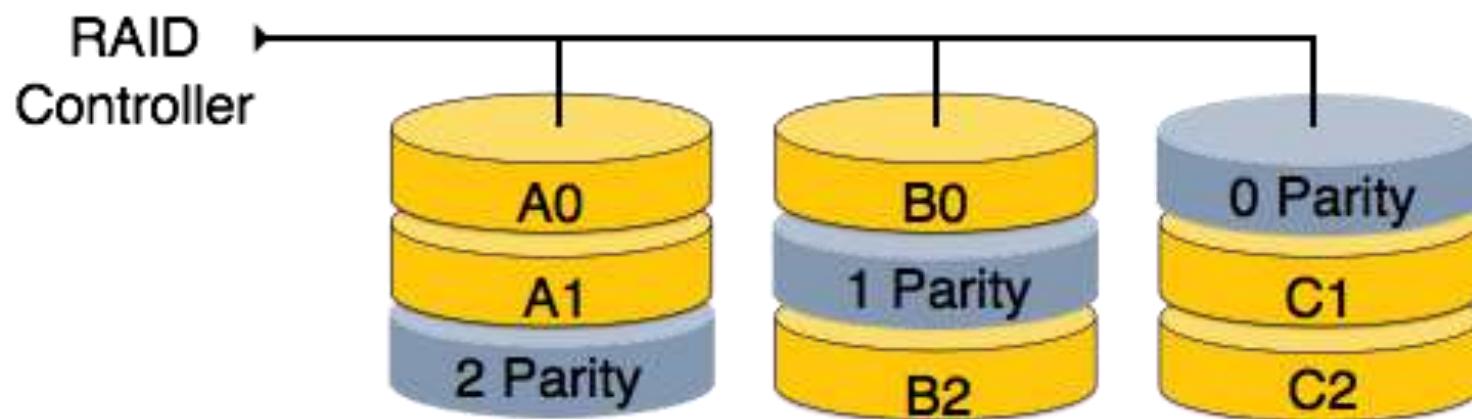
In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



RAID (Redundant Arrays of Independent Disks)

RAID 5

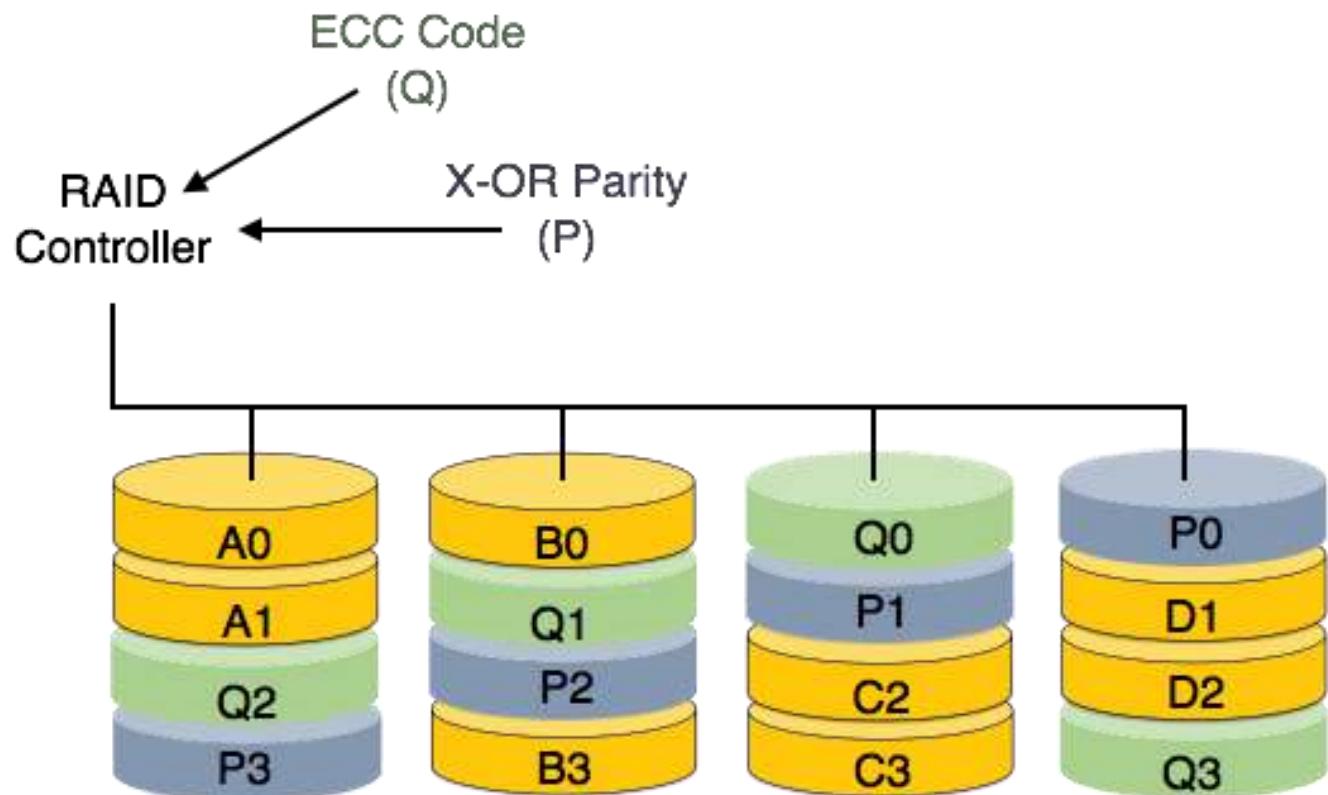
RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID (Redundant Arrays of Independent Disks)

RAID 6

RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.

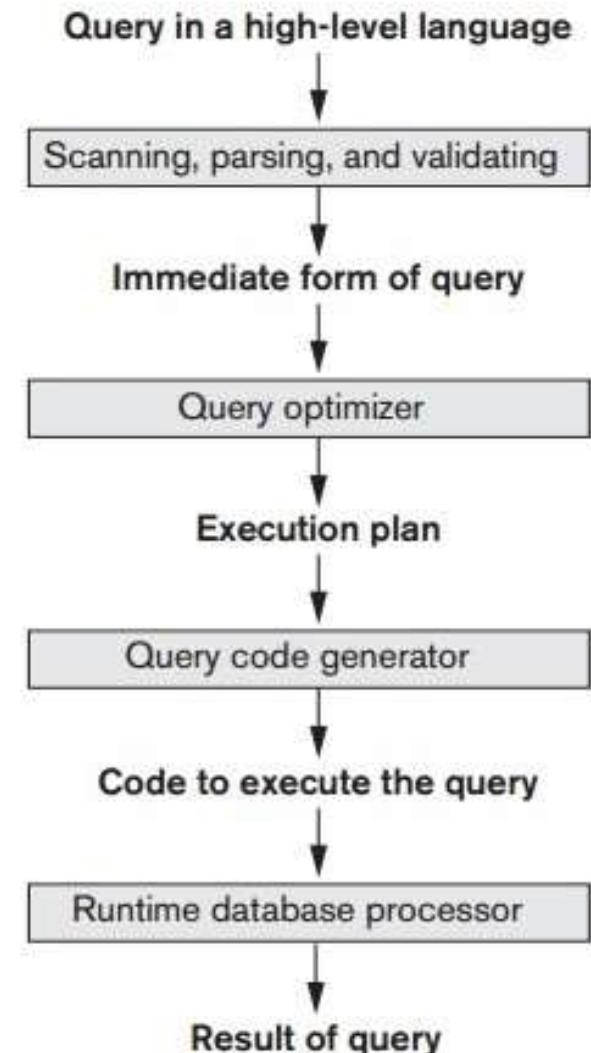


Query Processing and Optimization

- ✓ Query processing refers to the range of activities involved in extracting data from a database.
- ✓ A query expressed in a high level query language such as SQL must first be scanned, parsed and validated.

Steps while processing high level query:-

- ✓ when a query written in a high level language than first it is scanned
- ✓ The scanner identifies the language taken such as SQL keywords, attributes and relation name



Query Processing and Optimization

- ✓ The parser checks the query syntax to determine whether the query is formulated according to the syntax rules (rules of grammar) of the query language.
- ✓ Then the query must be validated, by checking that all attributes names and relation name are valid and semantically meaningful name.
- ✓ An internal representation of the query is created usually pre data structure called query tree or a graph data structure called query graphs.
- ✓ Then the DBMS must then make an execution strategy /plan. A query typically has possible execution strategy and the process of choosing a suitable one for the processing a query is called as query optimization.

Query Processing and Optimization

- ✓ Then basing upon the query execution plan the code generator generates the code to execute the plan.
- ✓ Finally the runtime database processor has the task of running the query code whether in compiled/interpreted mode.
- ✓ If the runtime error occurs an error message is generated by the runtime data base processor.

Query Optimization:

Query optimization is the process of selecting the most efficient query execution plan from many strategies available, that is usually possible for processing a given query especially if the query is complex.

There are two main techniques for implementing query optimization:-

- ✓ Heuristic rules for ordering the operation in a query execution strategy. This involves mainly query tree/ query graph.
- ✓ Systematic cost estimation of different execution plan/strategy and choosing the best with the lower cost estimation

Translating SQL queries into Relational Algebra:-

SQL is a query language which is first translated into an equivalent extended relational algebra, represented as a query tree data structure which is then optimized.

Ex:

SQL> select name from STUDENT where branch = “CSE”

Relational Algebra: $\prod_{NAME} (\sigma_{Branch = "MCA"} (STUDENT))$

Heuristic Rules in Query Optimization:

- ❖ The optimization technique that apply heuristic rules to modify the internal representation of a query ,which is usually in the form of a query tree/ query graph data structure to improve the expectation performance.
- ❖ The parser of a high-level query first generates an initial representation which is then optimized according to the Heuristic rules.

Query Tree:-

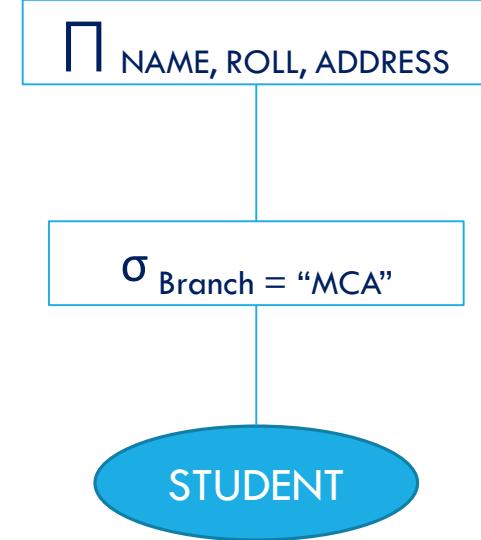
- ✓ A query tree is a tree data structure that corresponds to a relational algebra expression or extended relational algebra expression.
- ✓ A query tree represents the input relation of the query as leaf nodes of the tree ,and represents the relational algebra operations as internal nodes.
- ✓ An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.
- ✓ The execution terminates when the root node is executed and produces the result relation for the query.

Examples:

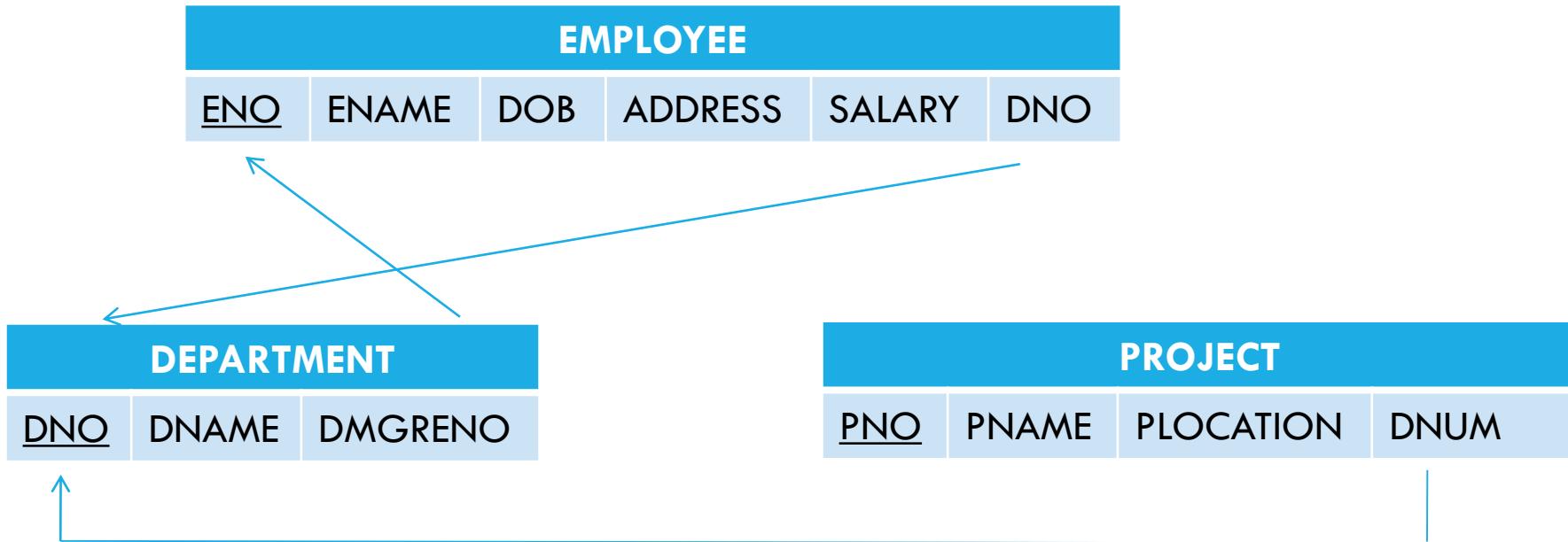
Q. Select name, roll, address from student where branch = 'MCA'

R. $\prod \text{NAME, ROLL, ADDRESS} (\sigma_{\text{Branch} = "MCA"} (\text{STUDENT}))$

The corresponding query tree will be:



Examples:



Q1. Find out the employee name and his department name whose address is 'GUNUPUR' and working under marketing department.

Q2. Find out the employees name, department name, date of birth and the project name where every project located in 'MUMBAI'

Examples: (Solution – Q1)

Q1. SELECT E.ENAME, D.DNAME FROM EMPLOYEE E, DEPARTMENT D
WHERE E.ADDRESS = 'GUNUPUR' AND E.ENO = D.DMGRENO

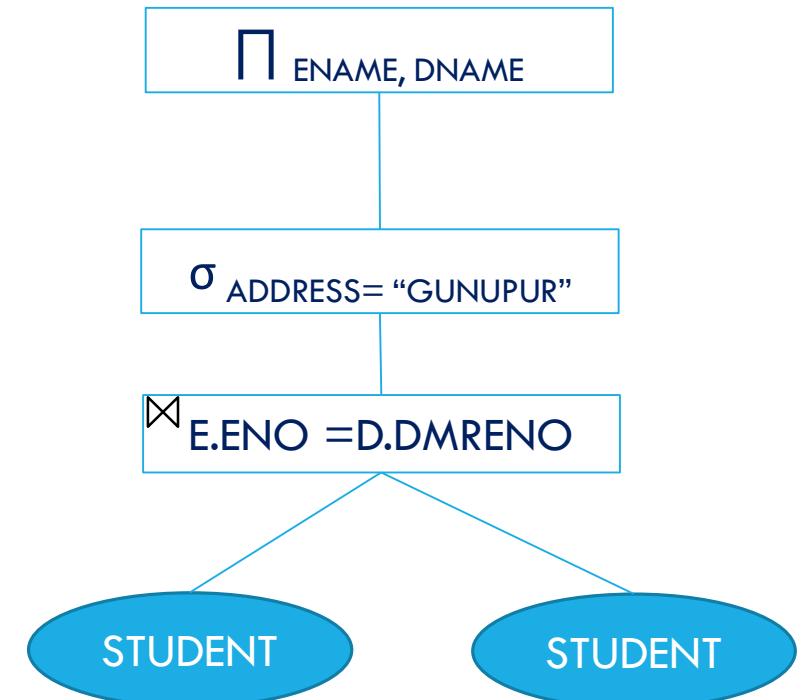
RA1. $\prod_{\text{ENAME, DNAME}} (\sigma_{\text{ADDRESS} = \text{"GUNUPUR"}} (\text{EMPLOYEE} \bowtie \text{DEPARTMENT}))$
OR

Temp1- EMPLOYEE \bowtie DEPARTMENT

Temp2- $\sigma_{\text{ADDRESS} = \text{"GUNUPUR"}} (\text{Temp1})$

Result- $\prod_{\text{ENAME, DNAME}} (\text{Temp2})$

The corresponding query tree will be:



Examples: (Solution – Q2)

Q1. SELECT E.ENAME, E.DOB, D.DNAME, P.PNAME FROM EMPLOYEE E, DEPARTMENT D, PROJECT P WHERE
P.PLOCATION = 'Mumbai'
AND E.ENO = D.DMGRNO
AND D.DNO = P.DNUM

RA1. $\prod_{\text{ENAME}, \text{DOB}, \text{DNAME}, \text{PNAME}} (\sigma_{\text{PLOCATION} = "MUMBAI"} ((\text{PROJECT} \bowtie \text{DEPARTMENT}) \bowtie \text{EMPLOYEE}))$

OR

Temp1 - $\sigma_{\text{PLOCATION} = "MUMBAI"} (\text{PROJECT})$

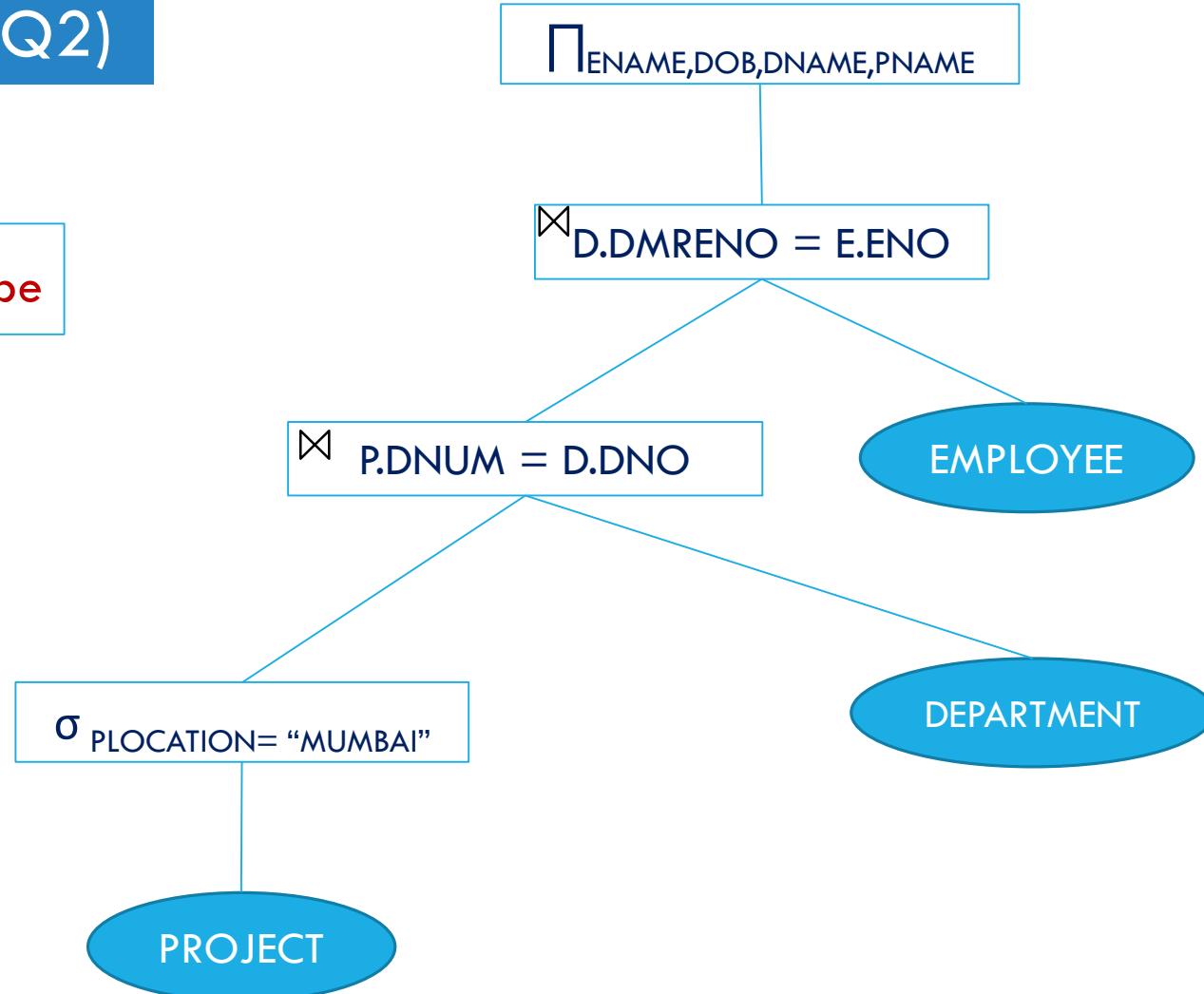
Temp2 - Temp1 \bowtie DEPARTMENT

Temp3 - Temp2 \bowtie EMPLOYEE

Result - $\prod_{\text{ENAME}, \text{DOB}, \text{DNAME}, \text{PNAME}} (\text{Temp3})$

Examples: (Solution – Q2)

The corresponding query tree will be



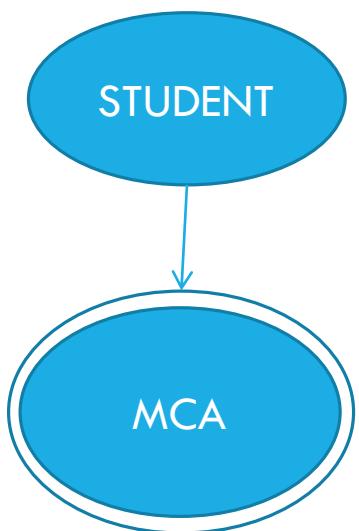
Query Graph:

- ✓ Query graph is used to represent a relational calculus expression.
- ✓ Relations in the query are represented by relation nodes which are displayed as single circles.
- ✓ Constant values, typically from the query selection conditions are represented by constant nodes ,which are displayed as double circles.
- ✓ Selection and join condition are represented by the graph edges.
- ❖ Finally, the attributes to be retrieved from each relation are displayed in square brackets above each relation.
 - ✓ The query graph representation does not indicate an order on which operations to perform first. This is a single graph corresponding to each query
 - ✓ Query tree are preferable, because the query optimizer needs to show the order of operation for query execution ,which is not possible in query graph.

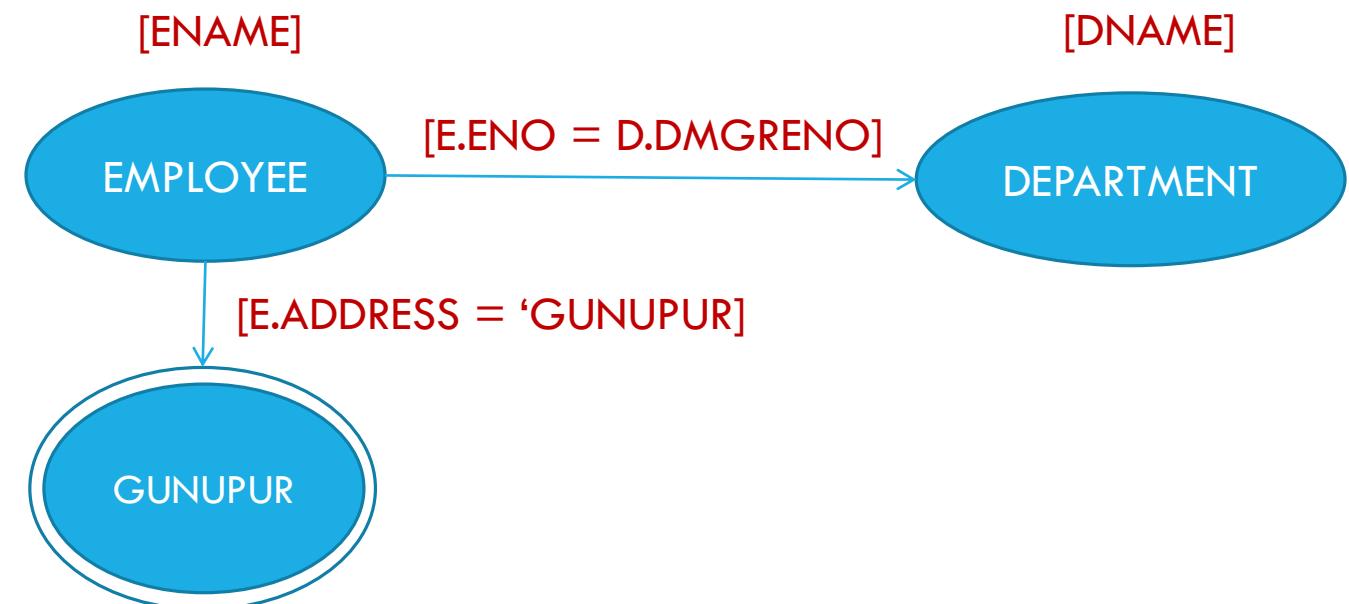
Examples: (Solution – Q1)

Example :

[NAME, ROLL, ADDRESS]

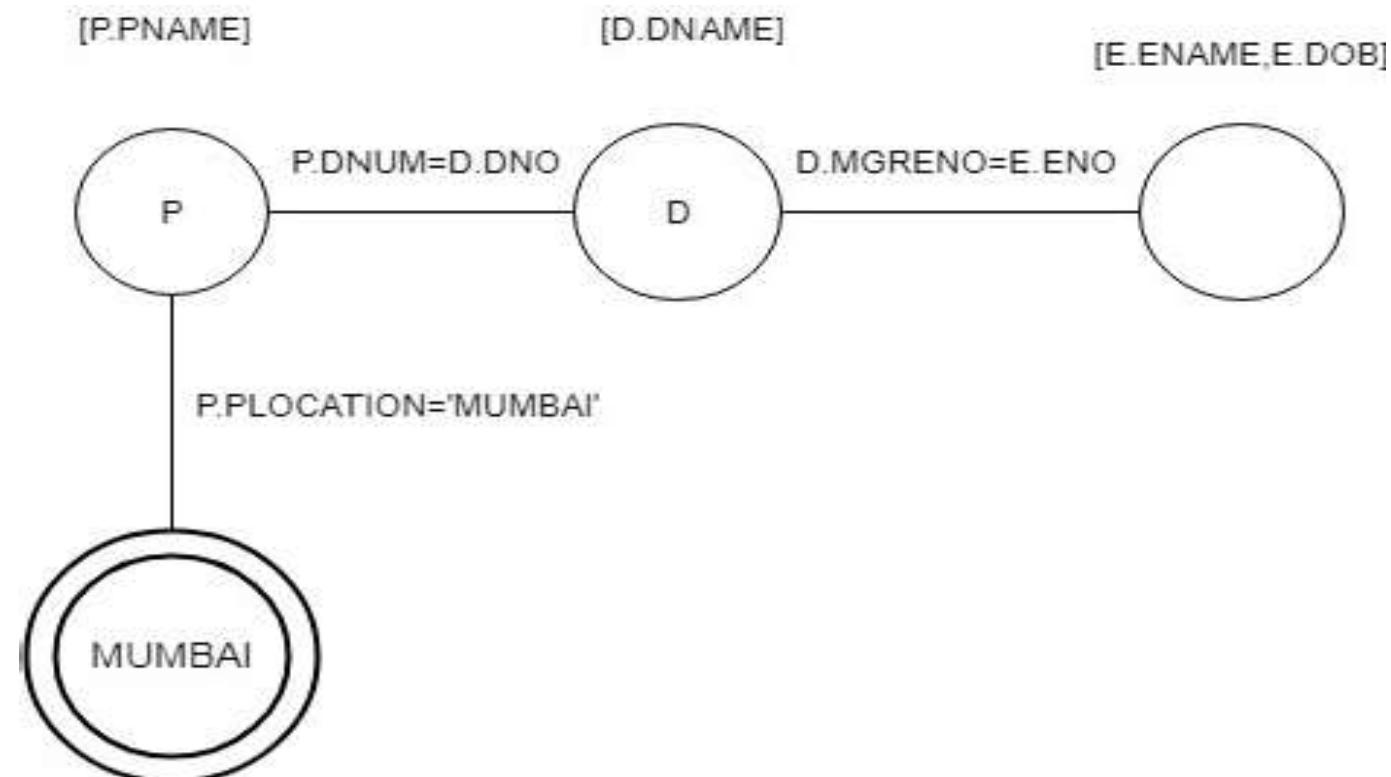


Example 1:



Examples: (Solution – Q2)

Example 2:



Heuristic Optimization of Query Tree:-

- ✓ The query parser will typically generate a standard initial query tree to correspond to an SQL query , without doing any optimization.
- ✓ This is the initial query tree or canonical query tree represent a relational algebra expression that is very inefficient if executed directly, because of the CARTESIAN PRODUCT(X) operations.
- ✓ Now, the job of the query optimizer is to transform the initial query tree into a final query tree that is efficient to execute.

TRANSACTION PROCESSING

- ✓ A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database.
- ✓ Transactions access data using read and write operations.

- **read operation(read(x))**

It transfers the data item x from the database to a local buffer belonging to the transaction that executed the read operation

- **write operation(write(x))**

It transfers the data item x from the local buffer of the transaction that executed the write back to the database.

Let T1 be a transaction that transfers Rs. 50 from account A to account B. This transaction can be defined as:-

T1: read (A);

A:=A-50;

Write (A);

Read (B);

B: =B+50;

Write (B);

ACID PROPERTIES:

✓ In order to maintain consistency in a database, before and after the transaction, certain properties are followed, these are called ACID properties.

Atomicity

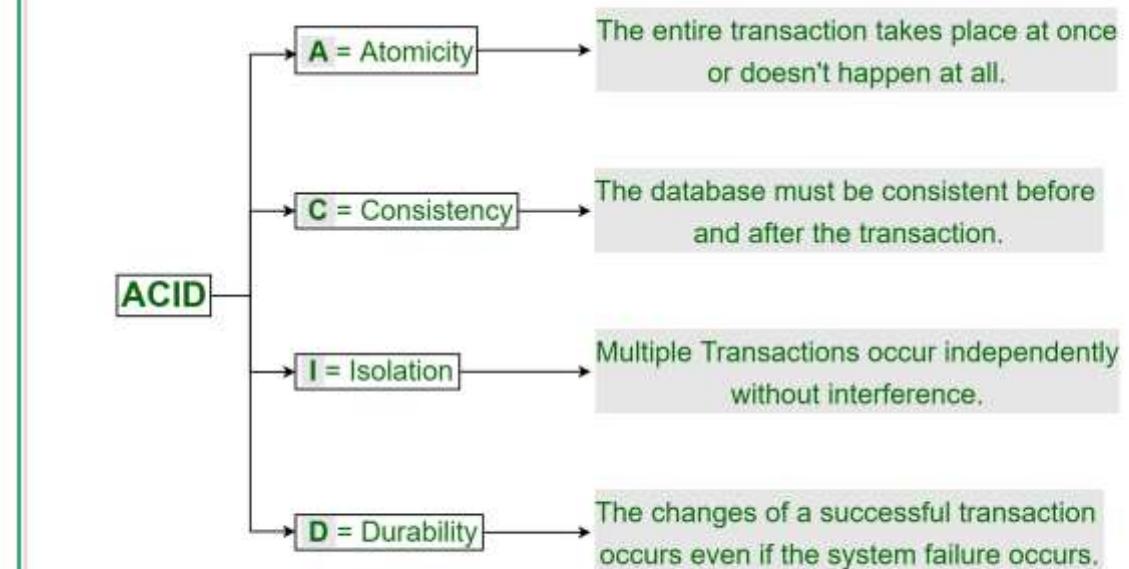
By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

— Abort: If a transaction aborts, changes made to database are not visible.

— Commit: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

ACID Properties in DBMS



ACID PROPERTIES:

✓ Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X)	Read (Y)
X: = X - 100	Y: = Y + 100
Write (X)	Write (Y)
After: X : 400	Y : 300

❖ If the transaction fails after completion of T1 but before completion of T2. (say, after write(X) but before write(Y)), then amount has been deducted from X but not added to Y. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

ACID PROPERTIES:

Consistency

- ❖ This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.
 - Referring to the example above, the total amount before and after the transaction must be maintained.

Total before T occurs = $500 + 200 = 700$.

Total after T occurs = $400 + 300 = 700$.

Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

ACID PROPERTIES:

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let $X = 500$, $Y = 500$.

Consider two transactions T and T'' .

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

T	T''
Read (X) $X := X * 100$ Write (X) Read (Y) $Y := Y - 50$ Write	Read (X) Read (Y) $Z := X + Y$ Write (Z)

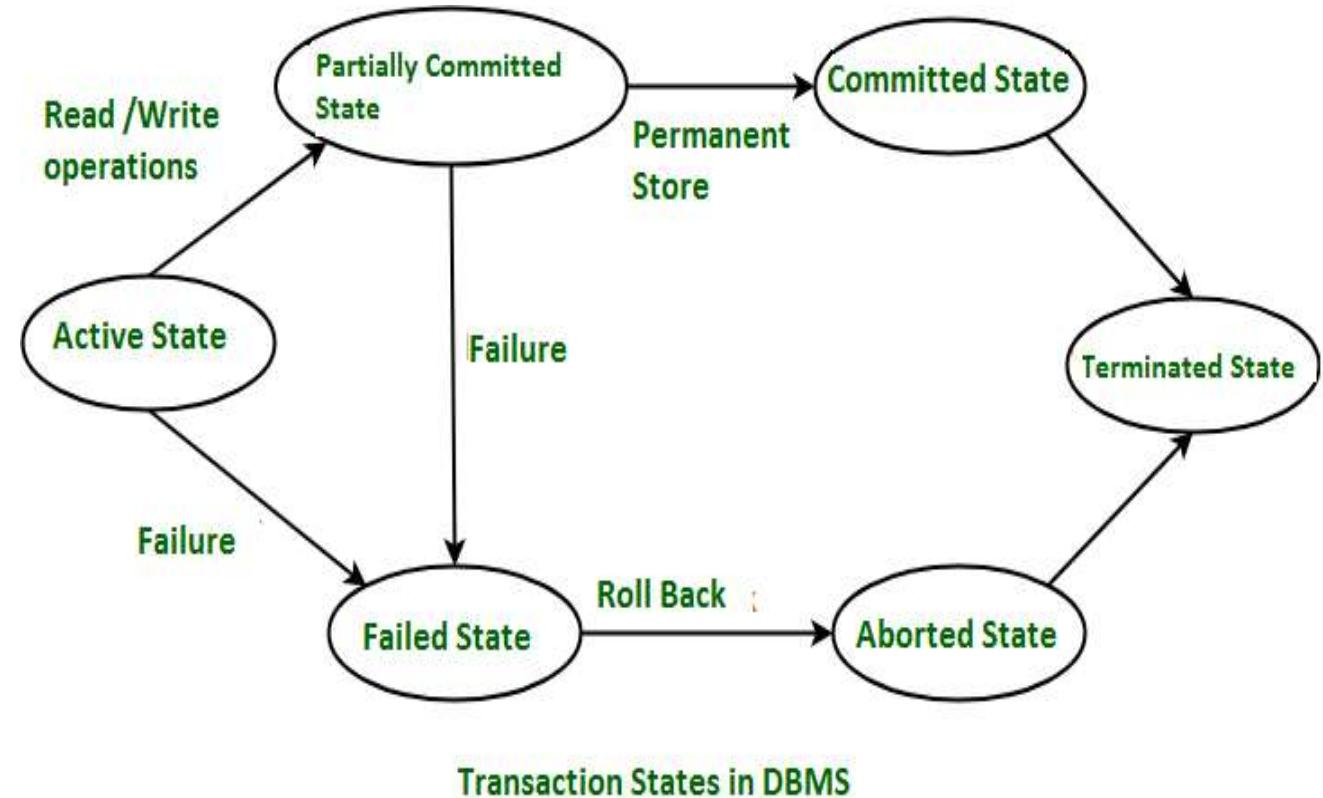
ACID PROPERTIES:

Durability:

- ✓ This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.
- ✓ The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

Transaction States in DBMS

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do processing we will do on the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.



Transaction States in DBMS

These are different types of Transaction States :

Active State –

When the instructions of the transaction is running then the transaction is in active state. If all the read and write operations are performed without any error then it goes to “partially committed state”, if any instruction fails it goes to “failed state”.

Partially Committed –

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the Data Base then state will change to “committed state” and in case of failure it will go to “failed state”.

Transaction States in DBMS

Failed State –

When any instruction of the transaction fails it goes to “failed state” or if failure occurs in making permanent change of data on Data Base.

Aborted State –

After having any type of failure the transaction goes from “failed state” to “aborted state” and in before states the changes are only made to local buffer or main memory and hence these changes are deleted or rollback.

Committed State –

It is the stage when the changes are made permanent on the Data Base and transaction is complete and therefore terminated in “terminated state”.

Terminated State –

If there is any roll back or the transaction come from “committed state” then the system is consistent and ready for new transaction and the old transaction is terminated.

CONCURRENT EXECUTIONS:-

- ✓ Transaction processing systems usually allow multiple transactions to run concurrently.
- ✓ In the serial execution- the transactions run serially that is one at a time, each transaction starting only after the previous one has completed.

ADVANTAGES OF CONCURRENT EXECUTIONS:-

The following are two good advantages of concurrent execution over the serial execution:-

Improved throughput and resource utilization:-

- ✓ While a read or write on behalf of one transaction is in progress on one disk, another transaction can be running on the CPU, while another transaction can use another disk. All of this increases the throughput of the system. (the number of transactions executed in a given amount of time.)
- ✓ Resource utilization means the processor and disk spend less time idle or not performing any useful work.

CONCURRENT EXECUTIONS:-

Reduced waiting time:-

- ✓ If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete.
- ✓ But, concurrent execution reduces the unpredictable delays in running transactions.

SCHEDULE:-

- ✓ The execution sequences are called schedules.
- ✓ When the database system executes several transactions concurrently, the corresponding schedule no longer needs to be serial.
- ✓ If two transactions are running concurrently, the operating system may execute one transaction for a little while, then perform a context switch, execute the second transaction for some time, and then switch back to the first transaction for some time and so on.

EXAMPLE:-

✓ Suppose in a schedule S, two transactions T1 and T2:-

Transaction T1 transfers Rs 50 from account A to B and,

✓ Transaction T2 transfers 10 percent of the balance from account A to account B. [Serial Schedule (S1)]

T1	T2
Read(A) A:= A-50 Write(A)	
Read(B) B:=B+50 Write(B)	
	Read(A) Temp:=A*0.1 A:=A-temp Write(A) Read(B) B:=B+temp Write(B)

[Concurrent schedule (S_2)]

T1	T2
Read(A) A:= A-50 Write(A)	
Read(B) B:=B+50 Write(B)	Read(A) Temp:=A*0.1 A:=A-temp
	Write(A) Read(B) B:=B+temp Write(B)

SERIALIZABILITY:-

- ✓ The database system must control concurrent execution of transactions, to ensure that the database state remains consistent.
- ✓ In the serializability there are mainly two concepts are involved:-
 - Conflict Serializability
 - View Serializability

Conflict Serializability:-

- ✓ Let us consider a schedule S in which there are two consecutive instructions, l_i and l_j , of transactions T_i and T_j , respectively($i \neq j$).
- ✓ If l_i and l_j refer to the same data item Q then the order of two transactions T_i and T_j may affect.

SERIALIZABILITY:-

For that there are four cases that we need to consider:-

$l_i = \text{read}(Q)$, $l_j = \text{read}(Q)$ [Then the order of l_i and l_j does not matter, since the same values Q is read by T_i and T_j .]

$l_i = \text{read}(Q)$, $l_j = \text{write}(Q)$ [If it comes before the l_j then T_i does not read.]

The value of Q , which is written by T_j in instruction l_j . If l_j comes before l_i , then T_i reads the value of Q that is written by T_j . Thus the order l_i and l_j matters.

$l_i = \text{write}(Q)$, $l_j = \text{read}(Q)$ [The order of l_i and l_j matters for reasons similar to those of the previous case.]

$l_i = \text{write}(Q)$, $l_j = \text{write}(Q)$ [Since both the instructions are write operations, the order of the instructions affect because the value of next $\text{read}(Q)$ instruction of S affected.]

So we can say that l_i and l_j are conflict serializability if they are operations by different transactions T_i and T_j on the same data item Q and at least one of those instructions is a write operation.

View Serializability:-

Consider two schedules S and S^l , where the same set of transactions participates in both schedules. The schedule S and S^l are said to be view equivalent if three conditions are met:-

- ✓ For each data item Q , if transaction T_i reads the initial value of Q in schedule S , then transaction T_i must, in schedule S^l , also read the initial value of Q .
- ✓ For each data item Q , if transaction T_i executes $\text{read}(Q)$ in schedule S , and if that value was produced by a $\text{write}(Q)$ operation executed by transaction T_j , then the $\text{read}(Q)$ operation of transaction T_i must in schedule S^l , also read the value of Q that was produced by the same $\text{write}(Q)$ operation of transaction T_j .
- ✓ For each data item Q , the transaction that performs the final write (Q) operation in schedule S must perform the final write (Q) operation in schedule S^l

RECOVERABILITY:-

- ✓ If a transaction T_i fails, we need to undo the effect of this transaction to ensure that atomicity property of the transaction. In a system that allows concurrent execution, it is necessary also to ensure that any transaction T_j that is dependent on T_i (that is T_j has read data item written by T_i) is also aborted.
- ✓ A recoverable schedule is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i appears before the commit operation of T_j .
- ✓ If a schedule is recoverable, to recover correctly from the failure of transaction T_i , we may have to roll back several transactions.
- ✓ If a single transaction failure leads to a series of transaction rollback, it is called as cascading rollback.
- ✓ A cascade less rollback is one where, for each pair of transaction T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i appears before the read operation of T_j .

PRECEDENCE GRAPH:-

- ✓ This is the most simple and efficient method for determining conflict serializability of a schedule.
- ✓ Precedence graph is a directed graph of a schedule.
- ✓ Suppose the graph consists of a pair $G=(V,E)$, where V is the set of vertices and E is set of edges.
- ✓ The set of vertices consists of all the transactions participating in the schedule.

The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:-

- T_i executes write (Q) before T_j executes read (Q).
- T_i executes read(Q) before T_j executes write(Q)
- T_i executes write (Q) before T_j executes write (Q).

- ✓ If an edge $T_i \rightarrow T_j$ exists in the precedence graph then in any serial schedule $S' |$ equivalent to S , T_i must appear before T_j .

PRECEDENCE GRAPH:-



All the instructions of T_i executed before the first instruction of T_j executed.



All the instructions of T_j executed before the first instruction of T_i executed.



It contains the edge $T_i \rightarrow T_j$, because T_i executes before T_j executes. It also contains the edge $T_j \rightarrow T_i$ because T_j executes before T_i executes.

CONCURRENCY CONTROL:-

When several transactions execute concurrently in the database, the system must control the interaction among the concurrent transactions, this control is achieved through one of a variety of mechanism called concurrency control schema.

The following are the concurrency control schema:-

LOCK-BASED PROTOCOLS:-

- ✓ In the concurrent execution of transactions, the data item must be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item.
- ✓ The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

CONCURRENCY CONTROL:-

- ❑ There are two modes in which a data item may be locked:-

Shared-mode lock:-

If a transaction T_i has obtained a shared-mode lock (denoted by S) on data item Q , then T_i can read, but cannot write the data item Q .

Exclusive-mode lock:-

If a transaction T_i has obtained an exclusive-mode lock (denoted by X) on data item Q , then T_i can both read and write Q .

- ❑ The transaction requests for a shared or exclusive lock to the concurrency control manager. The transaction can proceed with the operation only after the concurrency control manager grants the lock to the transaction.

CONCURRENCY CONTROL:-

COMPATIBILITY FUNCTION:-

- ✓ Let A and B represent shared and exclusive lock modes respectively.
- ✓ Suppose that a transaction T_i requests a lock of mode A on item Q on which transaction T_j currently holds a lock of mode B.
 - ✓ If transaction T_i can be granted a lock on Q immediately in spite of the presence of the mode B lock, then we say mode A is compatible with mode B.
 - ✓ Shared mode is compatible with shared mode, but not with exclusive mode.
 - ✓ If transaction T_i is wanted to lock the item Q, which is locked by another transaction T_j is an incompatible mode, then T_i have to wait until all incompatible mode have been released.
 - ✓ A transaction requests a shared lock on data item Q by executing the Lock-S(Q) instruction.
 - ✓ A transaction requests an exclusive lock on data item Q by executing the Lock-X(Q) instruction.
 - ✓ A transaction can unlock a data item by executing the unlock (Q) instruction.

CONCURRENCY CONTROL:-

Example:- suppose transaction T1 transfers Rs 100 from account A to B and transaction T2 displays the total amount of moneys in account A and B that is , the sum A+B.

T1	T2	Concurrency control Manager
Lock-X(A) Read(A) A:=A-100 Unlock(A)		Grant-X(A,T ₁)
	Lock-S(A) Read(A) Unlock(A) Lock-S(B) Read(B) Unlock(B) Display(A+B)	Grant-S(A,T ₂)
Lock-X(A) Write(A) Unlock(A)		Grant-X(A,T ₁)
	Lock-S(B) Read(B) B:=B+100 Unlock(B)	Grant-S(B,T ₁)

CONCURRENCY CONTROL:-

TWO-PHASE LOCKING PROTOCOL:-

- ❑ One protocol that ensures serializability is the two phase locking protocol.
- ❑ This protocol requires that each transaction issue lock, and unlock requests in two phases:-

Growing phase:-

A transaction may obtain locks, but may not release any locks.

Shrinking phase:- A transaction may release locks, but may not obtain any new locks.

- ✓ Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.
- ✓ The point in the schedule where the transaction has obtained its final lock(the end of its growing phase) is called the lock point of the transaction.

CONCURRENCY CONTROL:-

STRICT TWO-PHASE LOCKING PROTOCOL:-

- Cascading rollbacks can be avoided by a modification of two phase locking called the strict two phase locking protocol.
- This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits.

RIGOROUS TWO PHASE LOCKING PROTOCOL:-

- It requires that all locks be held until the transaction commits.

CONCURRENCY CONTROL:-

LOCK CONVERSION:-

- ❑ The lock conversion is a mechanism for upgrading a shared lock to an exclusive lock **and** downgrading an exclusive lock to a shared lock.
- ❑ The conversion from shared to exclusive mode is denoted by upgrade **and** from exclusive to shared by downgrade.
- ❑ The upgrading can be taken place in the growing phase, whereas downgrading can take place in only the shrinking phase.

CONCURRENCY CONTROL:-

TIMESTAMP BASED PROTOCOLS:-

- ❑ It is another method for determining the serializability.
- ❑ In this method with each transaction T_i in the system, we associate a unique fixed timestamp, denoted by $TS(T_i)$.
- ❑ This timestamp is assigned by the database system before the transaction T_i starts execution.
- ❑ If a transaction T_i has been assigned by $TS(T_i)$, and a new transaction T_j enters the system, then there must be $TS(T_i) < TS(T_j)$.

CONCURRENCY CONTROL:-

There are two simple methods for implementing this scheme:-

- ❑ Use the value of the system clock as the timestamp, a transaction's timestamp is equal to the value of the clock when the transaction enters the system.
- ❑ Use a logical counter that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter when the transaction enters the system.

To implement the method we associate with each data item Q two timestamp values:-

- ❑ W-timestamp (Q):- denotes the largest timestamp of any transaction that executed write (Q) successfully.
- ❑ R -timestamp (Q):- denotes the largest timestamp of any transaction that executed read (Q) successfully.

Database Recovery

- ❖ Basically, whenever a transaction is submitted to a DBMS for execution, the operating system is responsible for making sure or to be confirmed that all the operation which need to be performed in the transaction have completed successfully and their effect is either recorded in the database or the transaction doesn't affect the database or any other transactions.
- ❖ The DBMS must not permit some operation of the transaction T to be applied to the database while other operations of T is not. This basically may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of failures : There are basically following types of failures that may occur and leads to failure of the transaction such as:

- Transaction failure
- System failure
- Media failure and so on.

Types Of Failure

❖ System crash –

A hardware, software or network error occurs comes under this category this types of failures basically occurs during the execution of the transaction. Hardware failures are basically considered as Hardware failure.

❖ System error –

Some operation that is performed during the transaction is the reason for this type of error to occur, such as integer or division by zero. This type of failures is also known as the transaction which may also occur because of erroneous parameter values or because of a logical programming error. In addition to this user may also interrupt the execution during execution which may lead to failure in the transaction.

Types Of Failure

- ❖ Local error : This basically happens when we are doing the transaction but certain conditions may occur that may lead to cancellation of the transaction. This type of error is basically coming under Local error. The simple example of this is that, when we want to debit money from an insufficient balance account which leads to the cancellation of our request or transaction.
- ❖ Concurrency control enforcement : The concurrency control method may decide to abort the transaction, to start again because it basically violates serializability or we can say that several processes are in a deadlock.
- ❖ Disk failure : This type of failure basically occurs when some disk loses their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read /write operation of the transaction.

Database Recovery Techniques in DBMS

- ❑ There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.
- ❑ Recovery techniques are heavily dependent upon the existence of a special file known as a system log. It contains information about the start and end of each transaction and any updates which occur in the transaction.
- ❑ The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

Database Recovery Techniques in DBMS

- ❑ The log is kept on disk `start_transaction(T)`: This log entry records that transaction T starts the execution.
- ❑ `read_item(T, X)`: This log entry records that transaction T reads the value of database item X.
- ❑ `write_item(T, X, old_value, new_value)`: This log entry records that transaction T changes the value of the database item X from `old_value` to `new_value`. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- ❑ `commit(T)`: This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- ❑ `abort(T)`: This records that transaction T has been aborted.
- ❑ **checkpoint**: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Database Recovery Techniques in DBMS

- ❑ There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
- ❑ **Deferred update** – This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it does not have to change, hence UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database.
- ❑ **Immediate update** – In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo.

Shadow Paging/ Shadow Copy Technique:-

- ❑ This technique is based on making copies of the database, called shadow copies.
- ❑ In this technique, a pointer called db-pointer is maintained on disk, which points to the current copy of the database.
- ❑ In the shadow-copy technique, a transaction that wants to update the database first creates a complete copy of the database. All updates are done on the new database copy or the shadow copy by leaving the original copy untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy as the old copy of the database has not been affected.
- ❑ Once the transaction has been committed, all the updates that is performed are in the database pointed to by db-pointer.
- ❑ Thus, either all updates of the transaction are reflected or none of the effects are reflected, regardless of transaction failure.
- ❑ The atomicity and durability property is implemented in shadow copy technique.

DEADLOCK

- ❑ A system in a situation where there exists a set of transactions such that every transactions in the set is waiting for another transaction in the set .Thus ,we have arrived at a state where neither of these transactions can ever proceed with its normal execution .This situation is called as deadlock.
- ❑ For example ,there exists a set of waiting transactions { T0,T1,T2,.....TN } such that T0 is waiting for a data item that T1 holds and T1 is waiting for a data item that T2 holds, and so on ...Tn-1 is waiting for a data item that Tn holds ,and Tn is waiting for a data that T0 holds. None of the transactions can make progress in such a situation .
- ❑ There are three principles methods for dealing with the deadlock problem .These are:-
 - Deadlock prevention
 - Deadlock Detection
 - Deadlock Recovery

Deadlock prevention:-

- ❑ Deadlock prevention protocol is used to ensure that the system will never enter a deadlock state.
- ❑ There are two approaches to deadlock prevention :-
 - The first approach ensures that no cyclic wait can occur by ordering the request for locks that means each transaction locks all its data items before it begins execution .Either all are locked in one step or none are locked.
 - The second approach for preventing deadlock is to use a preemption and transaction rollbacks in preemption ,when a transaction T2 requests a lock that transaction T1 holds the lock granted to T1 may be preempted by rolling back of T1, and granting the lock to T2. To control the preemption , we assign a unique timestamp to each transaction should wait or roll back.

Deadlock prevention:-

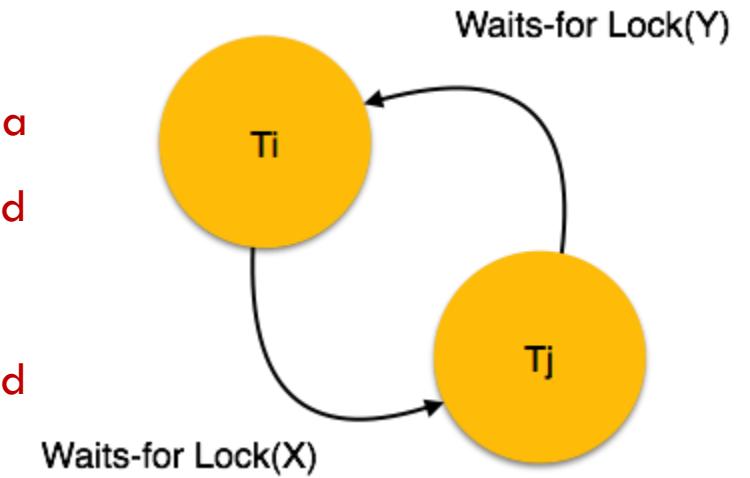
- ❑ Two different deadlock prevention schemes using timestamps has been proposed:-
 - ✓ The wait-die scheme is a non-preemptive technique .When a transaction T_i requests a data item currently held by T_j , T_i allowed to wait only if it has a timestamp smaller than that of T_j (that is T_i older than T_j)otherwise , T_i is rolled back(dies).
 - ✓ The wound-wait is a non-preemptive technique. It is counterpart of the wait-die scheme .when a transaction T_i requests a data item currently held by T_j , T_i is allowed to wait only if it has a timestamp larger than that of T_j (that is , T_i is younger than T_j),otherwise T_j is rolled back(dies).

Deadlock detection:-

- We can allow the system to enter a deadlock state, and then try to recover by using a deadlock detection and deadlock recovery scheme
- An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred .if one has ,then the system must attempt to recover from the deadlock
- Deadlocks can be described in terms of a directed graph called as wait-for-graph.
- The wait-for-graph consists of a pair $G=(V,E)$,where V is a set of vertices and E is a set of edges. The set of vertices consists of all the transaction in the system.
- When transaction T_i requests a data item currently being held by transaction T_j , then the edge $T_i \rightarrow T_j$ is inserted in the wait-for -graph.

Deadlock detection:-

- ❑ A deadlock exists in the system if and only if the wait-for-graph contains a cycle. Each transaction involved deadlocked in the cycle. Each transaction involved in the cycle is said to be deadlocked
- ❑ To detect deadlocks, the system needs to maintain the wait-for-graph, and periodically to invoke an algorithm that searches for a cycle in the graph.
- ❑ A deadlock exists in the system if and only if the wait-for-graph contains a cycle. Each transaction involved deadlocked in the cycle. Each transaction involved in the cycle is said to be deadlocked
- ❑ To detect deadlocks, the system needs to maintain the wait-for-graph, and periodically to invoke an algorithm that searches for a cycle in the graph.



Deadlock detection:-

In the example ,transaction T2,T3,nd T4 are all deadlocked ,because it contains the cycle

- Transaction T2 is waiting for transaction T4
- Transaction T4 is waiting for transaction T3
- And ,transaction T3 is waiting for transaction T2,therefore the graph contains the cycle i.e;

$T2 \rightarrow T4 \rightarrow T3 \rightarrow T2$

Deadlock recovery :-

- ❑ When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock.
- ❑ The most common solution is to rollback one or more transactions to break the deadlock.
- ❑ Three actions need to be taken :-
 - ❖ Selection of a victim :-
 - ✓ Given a set of deadlocked, we must determine which transaction transactions to roll back to break the deadlock.
 - ✓ We should rollback those transactions that will give the minimum cost.

Deadlock recovery :-

Rollback:-

- ✓ once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back.
- ✓ The simplest solution is a total rollback; Abort the transaction and then restart it.
- ✓ Partial rollback requires the system to maintain additional information about the state of all running transactions.

Starvation:-

- ✓ In the case of a selection of victim, it may happen that the same transaction is always picked as a victim.
- ✓ As a result, this transaction never completes its designated task, thus this is a starvation.
- ✓ We must ensure that a transaction can be picked as a victim only a small (finite) number of times. The most common solution is to include the number of rollback in the cost factor.