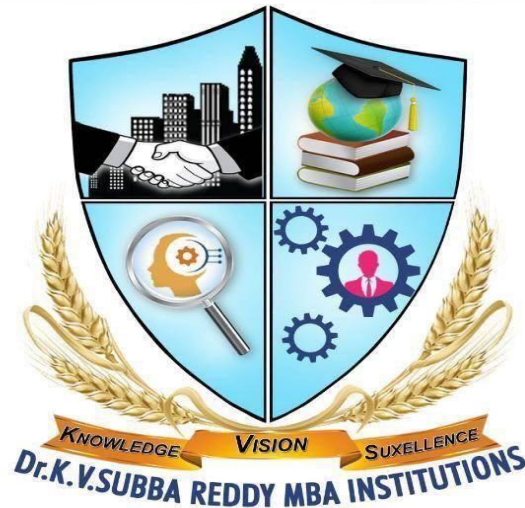


**KVSB - Dr.K.V.SUBBA REDDY**  
**SCHOOL OF BUSINESS MANAGEMENT**  
**MCA & MBA COLLEGES**



Institute Name:	<b>Dr. K. V. Subba Reddy School of Business Management</b>
College Code:	<b>JJ</b>
ICET Code:	<b>KVSB</b>
Name of Programmes:	<b>MBA &amp; MCA</b>
<b>SUBJECT: DATA BASE MANAGEMENT SYSTEMS</b>	

# UNIT-1

## **Introduction:**

In science, business, education, economy, law, culture, all areas of human development “work” with the data. Databases play a crucial role within science research: the body of scientific and technical data and information in the public domain is massive and factual data are fundamental to the progress of science.

Stock exchange data are absolutely necessary to any analyst. In everyday activity of a teacher, an educator, an academic or a lawyer has the data.

There are databases collecting all sorts of different data:

- The Evaluated Nuclear Structure Data
- Human Genome Database.
- Prisoners’ DNA data.
- Telephone Numbers,
- Legal Materials and many others.

## **Data and Information:**

Data are **raw facts** that constitute building block of information. Data are the heart of the DBMS. It is to be noted that all the data will not be a useful information. Useful information is obtained from processed data.

In other words, data has to be interpreted in order to obtain information. Data are a representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or automatic means.

Data are the most stable part of an organization’s information system. A company needs to save information about employees, departments, and salaries. These pieces of information are called data. Permanent storage of data are referred to as persistent data.

Generally, we perform operations on data or data items to supply some information about an entity. For **example** library keeps a list of members, books, due dates, and fines.

## **Database:**

A database is a well-organized collection of data that are related in a meaningful way, which can be accessed in different logical orders.

Database systems are systems in which the interpretation and storage of information are of primary importance.

The database should contain all the data needed by the organization as a result, a huge volume of data, the need for long-term storage of the data, and access of the data by a large number of users generally characterize database systems.

The simplified view of database system is shown in Fig. 1.1. From this figure, it is clear that several users can access the data in an organization.

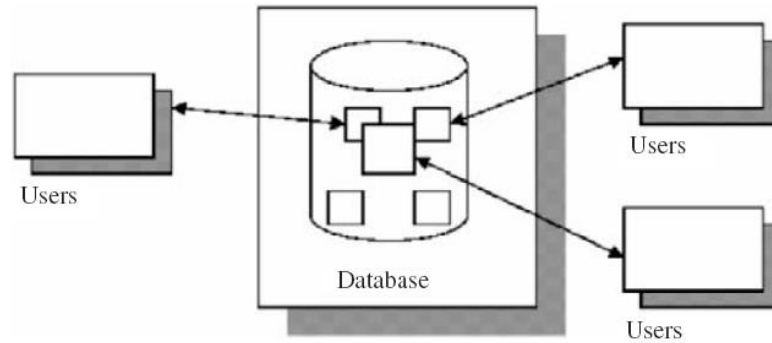


Fig. 1.1. Simplified database view

Here the integrity of the data should be maintained. A database is integrated when same information is not recorded in two places.

### **Database Management System:**

A database management system (DBMS) consists of collection of interrelated data and a set of programs to access that data. It is **software** that is helpful in maintaining and utilizing a database.

### **A DBMS Consists of:**

- A collection of interrelated and persistent data. This part of DBMS is referred to as database (DB).
- A set of application programs used to access, update, and manage data. This part constitutes data management system (MS).
- A DBMS is general-purpose software i.e., not application specific. The same DBMS (e.g., Oracle, Sybase, etc.) can be used in railway reservation system, library management, university, etc.
- A DBMS takes care of storing and accessing data, leaving only application specific tasks to application programs.

DBMS is a complex system that allows a user to do many things to data as shown in Fig. 1.2.

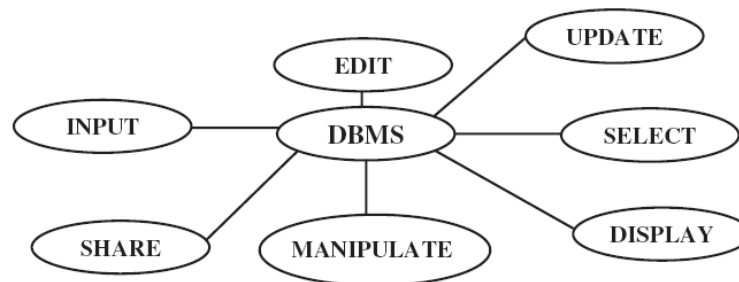


Fig. 1.2. Capabilities of database management system

From this figure, it is evident that DBMS allows user to input data, share the data, edit the data, manipulate the data, and display the data in the database. Because a DBMS allows more than one user to share the data; the complexity extends to its design and implementation.

## **Objectives of DBMS:**

The main objectives of database management system are data availability, data integrity, data security, and data independence.

### **Data Availability**

Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

### **Data Integrity**

Data integrity refers to the correctness of the data in the database. In other words, the data available in the database is a reliable data.

### **Data Security**

Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

### **Data Independence**

DBMS allows the user to store, update, and retrieve data in an efficient manner. DBMS provides an “abstract view” of how the data is stored in the database. In order to store the information efficiently, complex data structures are used to represent the data. The system hides certain details of how the data are stored and maintained.

### **Mass Storage**

DBMS can store a lot of data in it. So for all the big firms, DBMS is really ideal technology to use. It can store thousands of records in it and one can fetch all that data.

### **Removes Duplicity**

If you have lots of data then data duplicity will occur for sure at any instance. DBMS guarantee it that there will be no data duplicity among all the records. While storing new records, DBMS makes sure that same data was not inserted before.

### **Multiple Users Access**

No one handles the whole database alone. There are lots of users who are able to access database. So this situation may happen that two or more users are accessing database. They can change whatever they want, at that time DBMS makes it sure that they can work concurrently.

## **Database Applications**

### **Enterprise Information**

Sales: customers, products, purchases

Accounting: payments, receipts, assets

**Human Resources:** Information about employees, salaries, payroll taxes.

Manufacturing: management of production, inventory, orders, supply chain.

### **Banking and finance**

customer information, accounts, loans, and banking transactions.

Credit card transactions

**Finance:** sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data

**Universities:** registration, grades

## Purpose of Database Systems

- Data redundancy and inconsistency: data is stored in multiple file formats resulting in duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats
- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become “buried” in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

## University Database(Design) Example

- In this text we will be using a university database to illustrate all the concepts
- Data consists of information about:
  - Students
  - Instructors
  - Classes
- Application program examples:
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts

## ANSI / SPARC Data Model

The distinction between the logical and physical representation of data were recognized in 1978 when ANSI/SPARC committee proposed a generalized framework for database systems.

ANSI/SPARC is an acronym for the *American National Standard Institute / Standard Planning and Requirements Committee*. A standard three level approach to database design has been approved by this committee

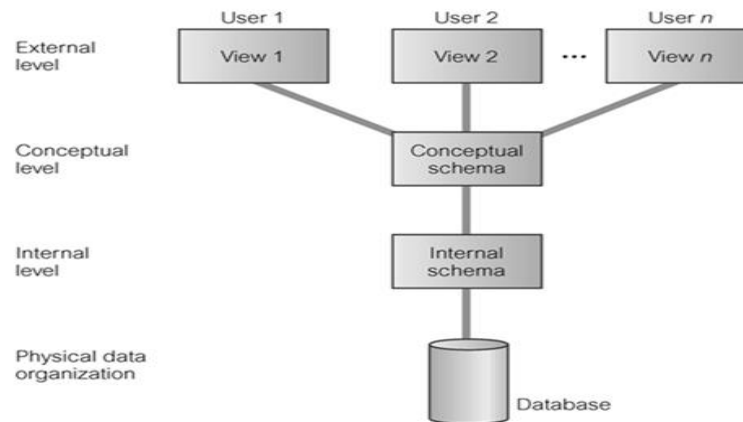
Already we know data independency, means the internal structure of database should be unaffected by changes to physical aspects of storage. Because of data independence, the Database administrator can change the database storage structures without affecting the users view.

The different levels of data abstraction are:

- 1) External level
- 2) Conceptual level
- 3) Internal level (includes physical data storage)

The 3 Level Architecture has the aim of enabling users to access the same data but with a personalized view of it. The distance of the internal level from the external level means that

users do not need to know how the data is physically stored in the database. This level of separation also allows the Database Administrator (DBA) to change the database storage structures without affecting the users' views.



**External Level (User Views):** A user's view of the database describes a part of the database that is relevant to a particular user. It excludes irrelevant data as well as data which the user is not authorized to access. It is also known as an **view level**.

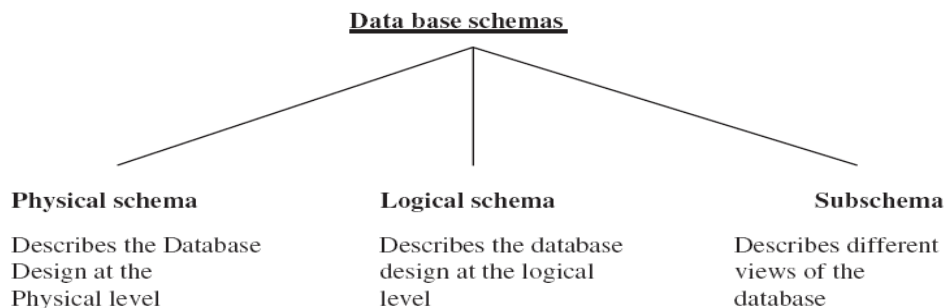
View level is the highest level of abstraction. It is the view that the individual user of the database has. There can be many view level abstractions of the same data.

**Conceptual Level:** The conceptual level describes what data is stored within the whole database and how the data is inter-related. It is also known as Logical level. The conceptual level does not specify how the data is physically stored.

**Internal Level:** The internal level involves how the database is physically represented on the computer system. It describes how the data is actually stored in the database and on the computer hardware. It is also known as **physical level**.

It provides the internal view of the actual physical storage of data. The physical level describes complex low-level data structures in detail.

**Database Schema:** The database schema provides an overall description of the database structure (not actual data). There are three types of schema which relate to the 3 Level Database Architecture.



•

## Database Design

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - Business decision – What attributes should we record in the database?
    - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

## Database Engine

A database system is partitioned into modules that deal with each of the responsibilities of the overall system.

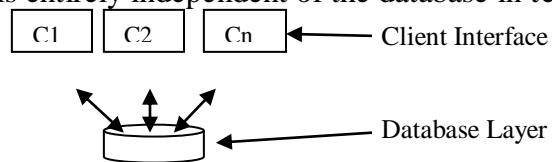
- The functional components of a database system can be divided into
- The storage manager,
- The query processor component,
- The transaction management component.
- 

## DBMS – Architecture (1-tier, 2-tier, and 3-tier Architecture:)

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.

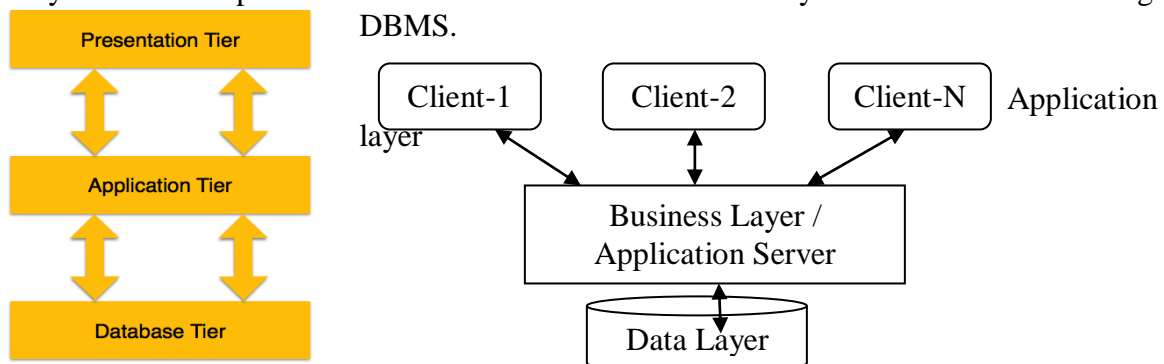
In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.



### **3-tier Architecture:**

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



(or)

**User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

**Application (Middle) Tier** – at this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

**Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Database users are categorized based up on their interaction with the data base.

These are seven types of data base users in DBMS.

1. **Database Administrator (DBA) :**

Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database.

The DBA will then create a new account id and password for the user if he/she need to access the data base.

DBA is also responsible for providing security to the data base and he allows only the authorized users to access/modify the data base.

- DBA also monitors the recovery and back up and provide technical support.
- The DBA has a DBA account in the DBMS which called a system or superuser account.
- DBA repairs damage caused due to hardware and/or software failures.

2. **Naive / Parametric End Users :**

Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results.

For examples, Railway's ticket booking users are naive users. Clerks in any bank is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.

3. **System Analyst :**

System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.



#### **4.Sophisticated Users :**

Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own data base applications according to their requirement. They don't write the program code but they interact the data base by writing SQL queries directly through the query processor.

#### **5.Data Base Designers :**

Data Base Designers are the users who design the structure of data base which includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items to be related.

#### **6.Application Program :**

Application Program are the back end programmers who writes the code for the application programs.They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.

#### **7.Casual Users / Temporary Users :**

Casual Users are the users who occasionally use/access the data base but each time when they access the data base they require the new information, for example, Middle or higher level manager.

### **Relational Data Model:**

The relational model uses a collection of tables to represent both data and the relationships among those data. Tables are logical structures maintained by the database manager. The relational model is a combination of three components, such as **1) Structural, 2) Integrity, and 3) Manipulative** parts.

#### **1) Structural Part**

The structural part defines the database as a collection of relations.

#### **2) Integrity Part**

The database integrity is maintained in the relational model using primary and foreign keys.

#### **3) Manipulative Part**

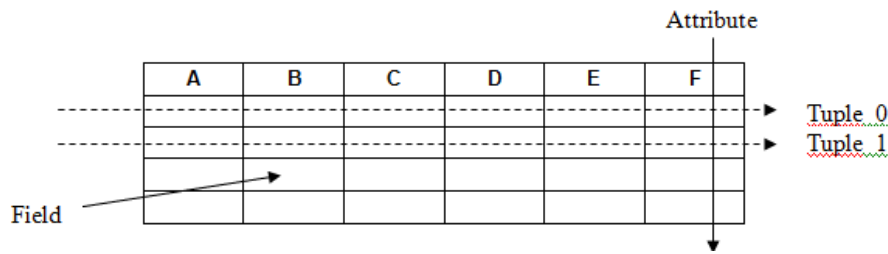
The relational algebra and relational calculus are the tools used to manipulate data in the database. Thus relational model has a strong mathematical background. The key features of relational data model are as follows:

- Each row in the table is called **tuple**.
- Each column in the table is called **attribute**.
- The intersection of row with the column will have data value.
- In relational model rows can be in any order.
- In relational model attributes can be in any order.
- By definition, all rows in a relation are distinct. No two rows can be exactly the same.
- Relations must have a key. Keys can be a set of attributes.

- For each column of a table there is a set of possible values called its domain. The domain contains all possible values that can appear under that column.
- Domain is the set of valid values for an attribute.
- Degree of the relation is the number of attributes (columns) in the relation.
- Cardinality of the relation is the number of tuples (rows) in the relation.

The **terms** commonly used by **user**, **model**, and **programmers** are given below.

<u>User</u>	<u>Model</u>	<u>Programmer</u>
Row	Tuple	Record
Column	Attribute	Field
Table	Relation	File



**Table and**

**Relation:**

The general doubt that will rise when one reads the relational model is the difference between table and relation. For a table to be relation, the following rules holds good:

- The intersection row with the column should contain single value (atomic value).
- All entries in a column are of same type.
- Each column has a unique name (column order not significant).
- No two rows are identical (row order not significant).

### Example of Relational Model

Representation of Movie data in tabular form is shown later.

MOVIE			
Movie Name	Director	Actor	Actress
Titanic	James Cameron	Leonardo DiCapiro	Kate Winslet
Autograph	Cheran	Cheran	Gopika
Roja	Maniratnam	AravindSwamy	Madubala

In the earlier relation:

The degree of the relation (i.e., is the number of column in the relation) = 4.

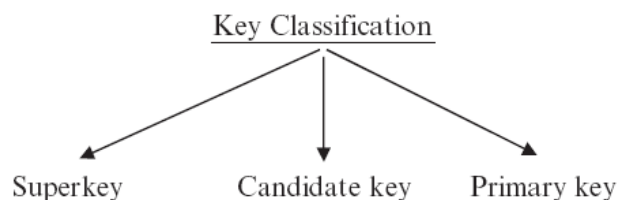
The cardinality of the relation (i.e., the number of rows in the relation) = 3.

### 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 superkey is a subset of attributes of an entity-set that uniquely identifies the entities. Superkeys represent a constraint that prevents two entities from ever having the same value for those attributes.

**Candidate Key:**

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

**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 Integrity, NULL Integrity, Domain Integrity, 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.

Consider the entity PLAYER; the attributes of the entity PLAYER are Name, Age, Nation, and Rank. In this example, let us consider PLAYER’s name as the primary key even though two players can have same name. We cannot insert any data in the relation PLAYER without entering the name of the player. This implies that primary key cannot be null.

**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:**

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size.

The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. The domain integrity constraint specifies that each attribute must have values derived from a valid range.

#### **Example 1**

The **age** of the person cannot have any letter from the alphabet. The age should be a numerical value.

#### **Example 2**

Consider the relation **APPLICANT**. Here APPLICANT refers to the person who is applying for job. The **gender** of the applicant should be either male (M) or female (F). Any entry other than M or F violates the domain constraint.

### **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.

### **Relational Algebra:**

The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation. Thus, both the operands and the results are relations; hence the output from one operation can become the input to another operation. This allows expressions to be nested in the relational algebra. This property is called closure.

Relational algebra is an abstract language, which means that the queries formulated in relational algebra are not intended to be executed on a computer. Relational algebra consists of group of relational operators that can be used to manipulate relations to obtain a desired result. Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system.

### **Role of Relational Algebra in DBMS**

Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system. The role of relational algebra in DBMS is shown in Fig. 3.1. (Diagram in next page).

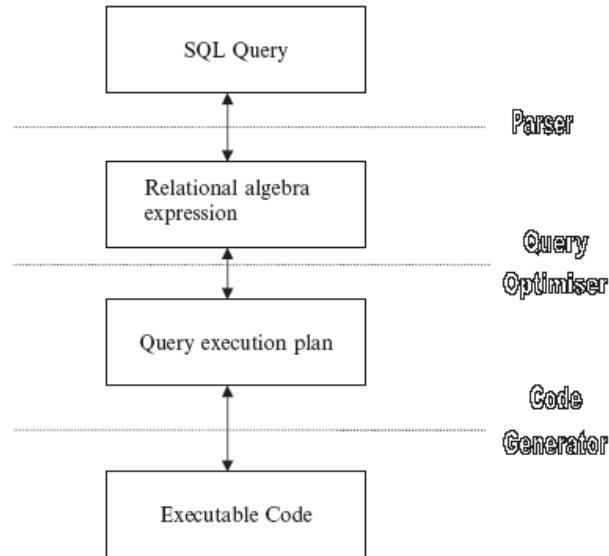


Fig. 3.1. Relational algebra in DBMS

From the figure it is evident that when a SQL query has to be converted into an executable code, first it has to be parsed to a valid relational algebraic expression, then there should be a proper query execution plan to speed up the data retrieval. The query execution plan is given by query optimizer.

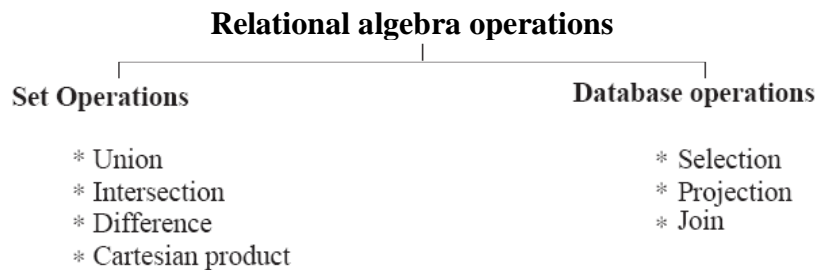
**Relational Algebra Operations:**

Operations in relational algebra can be broadly classified into set operation and database operations.

**Unary and Binary Operations:**

**Unary** operation involves one operand, whereas **binary** operation involves two operands. The selection and projection are **unary operations**. Union, difference, Cartesian product, and Join operations are **binary operations**:

- Unary operation operate on one relation.
- Binary operation operate on more than one relation.



Three main database operations are SELECTION, PROJECTION, and JOIN.

**1) Selection Operation:**

The selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (Predicate). Selection operation can be considered as row wise filtering.

**Syntax** of Selection Operation:  **$\sigma$  Predicate (R)**

Here R refers to relation and predicate refers to condition. **Example:** consider the STUDENT relation with the attributes Roll number, Name, and GPA (Grade Point Average).

STUDENT		
Student Roll. No	Name	GPA
001	Aravind	7.2
002	Anand	7.5
003	Balu	8.2
004	Chitra	8.0
005	Deepa	8.5
006	Govind	7.2
007	Hari	6.5

**Query 1:** List the Roll. No, Name, and GPA of those students who are having GPA of above 8.0  
Query expressed in relational algebra as  $\sigma_{GPA > 8}(\text{Student})$ .

The result of the earlier query is:

Student Roll. No	Name	GPA
003	Balu	8.2
005	Deepa	8.5

## 2) Projection Operation:

The projection operation works on a single relation **R** and defines a relation that contains a vertical subject of R, extracting the values of specified attributes and elimination duplicates. The projection operation can be considered as column wise filtering.

**Syntax** of Projection Operation:

The syntax of projection operation is given by:  $\Pi_{a_1, a_2, \dots, a_n}(R)$ . Where  $a_1, a_2, \dots, a_n$  are attributes and R stands for relation.

STAFF				
Staff No	Name	Gender	Date of birth	Salary
SL21	Raghavan	M	1-5-76	15,000
SL22	Raghu	M	1-5-77	12,000
SL55	Babu	M	1-6-76	12,500
SL66	Kingsly	M	1-8-78	10,000

**Example:** consider the relation STAFF, with the attributes Staff number, Name, Gender, Date of birth, and Salary.

**Query 1:** Produce the list of salaries for all staff showing only the Name and salary detail.

Relational algebra expression:  $\Pi_{\text{Name.salary}}(\text{staff})$ .

**Output for the Query 1**

Name	Salary
Raghavan	15,000
Raghu	12,000
Babu	12,500
Kingsly	10,000

## 3) Union Operation:

The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.

**Relational Algebra Expression**

The union of two relations R and S are denoted by  $R \cup S$ .  $R \cup S$  is pictorially represented in the Fig. 3.4.

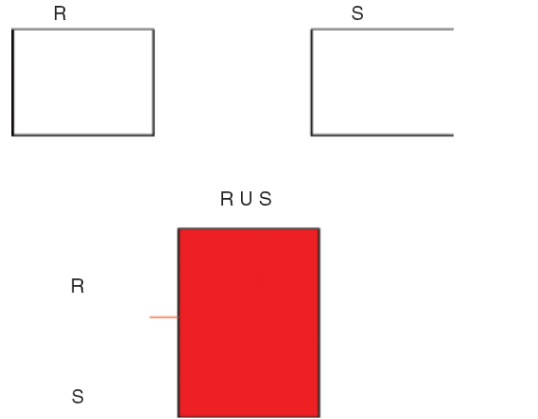


Fig. 3.4. Union of two relations R and S

Illustration of UNION Operation:

To illustrate the UNION operation consider the two relations Customer 1 and Customer 2 with the attributes Name and city.

Customer 1		Customer 2	
Name	City	Name	City
Anand	Coimbatore	Gopu	Tirunelveli
Aravind	Chennai	Balu	Kumbakonam
Gopu	Tirunelveli	Rahu	Chidambaram
Helan	Palayankottai	Helan	Palayamkottai

**Example**

*Query* Determine Customer 1  $\cup$  Customer 2

Result:

Customer 1 $\cup$ Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai
Balu	Kumbakonam
Gopu	Tirunelveli
Rahu	Chidambaram
Helan	Palayamkottai

**4) Intersection Operation:**

The intersection operation defines a relation consisting of the set of all tuples that are in both R and S. Relational Algebra Expression:

The intersection of two relations R and S is denoted by  $R \cap S$ .

Illustration of Intersection Operation:

The intersection between the two relations R and S is shown in **Example**

Find the intersection of Customer 1 with Customer 2 in the following table.

Customer 1 $\cap$ Customer 2	
Name	City
Gopu	Tirunelveli
Helan	Palayamkottai

### 5) Difference Operation:

The set difference operation defines a relation consisting of the tuples that are in relation R but not in S.

Relational Algebra Expression:

The difference between two relations R and S is denoted by  $R - S$ .

Illustration of Difference Operation

The difference between two relations R and S is shown in

### Example

Compute R-S for the relation shown in the following table.

Customer 1 - Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai

### 6) Division Operation:

The division of the **relation R** by the **relation S** is denoted by  $R \div S$ , where  $R \div S$  is given by:

$$R \div S = \Pi_{R-S(r)} - \Pi_{R-S}((\Pi_{R-S(r)} \times S) - r)$$

To illustrate division operations consider two relations STUDENT and MARK. The STUDENT relation has the attributes Student Name and the mark in particular subject say mathematics. The MARK relation consists of only one column mark and only one row.

Student		Mark
Name	Mark	Mark
Arul	97	100
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

**Case (1)**



If we divide the STUDENT relation by the MARK relation, the resultant relation is shown as:

<u>Answer</u>
<u>Name</u>
Banu
<u>Dinesh</u>

**Case (2)**

Now modify the relation MARK that is change the mark to be **98**. So that the entry in the MARK relation is modified as **98**.

<u>Student</u>	<u>Mark</u>	
Name	Mark	<u>Answer</u>
Arul	97	Name
Banu	100	Christi
Christi	98	<u>Lakshmi</u>
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

**7) Cartesian Product Operation:**

The Cartesian product operation defines a relation that is the concatenation of every tuples of relation R with every tuples of relation S. The result of Cartesian product contains all attributes from both relations R and S.

**Relational Algebra Symbol for Cartesian Product:**

The Cartesian product between the two relations R and S is denoted by **R × S**.

*Note:* If there are **n1** tuples in relation R and **n2** tuples in S, then the number of tuples in **R × S** is **n1\*n2**.

**Example:** If there are 5 tuples in relation “R” and 2 tuples in relation “S” then the number of tuples in **R × S** is **5 \* 2 = 10**.

<b>R</b>	<b>S</b>	
<div style="width: 80%; margin: 0 auto;">a</div> <div style="width: 80%; margin: 0 auto;">b</div>	<div style="width: 80%; margin: 0 auto;">1</div> <div style="width: 80%; margin: 0 auto;">2</div> <div style="width: 80%; margin: 0 auto;">3</div>	R
		S
		a
		a
		a
		b
		b
		b

**Note:**  
 No. of tuples in **R × S** = **2 \* 3 = 6**  
 No. of attributes in **R × S** = **2**

## UNIT-III

### Entity-Relationship Model:

#### Introduction:

The entity-relationship (E-R) model consisting of **basic objects**, called **entities**, and **relationships** among these objects. It was developed to facilitate database design by allowing specification of an enterprise schema, which represents the overall logical structure of a database. The E-R data model is one of several semantic data models;

The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model.

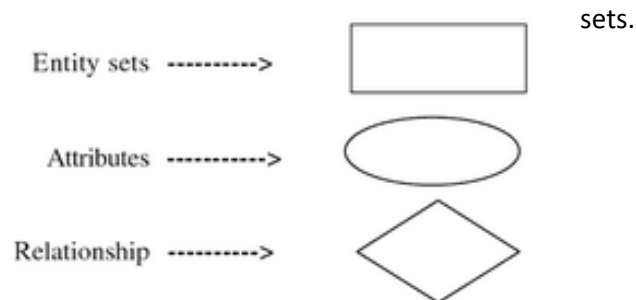
### The Building Blocks of An Entity Relationship Diagram:

ER diagram is a graphical modeling tool to standardize ER modeling. The modeling can be carried out with the help of pictorial representation of entities, attributes, and relationships. So the basic building blocks of Entity-Relationship diagram are

- 1) **Entity**
- 2) **Attribute**
- 3) **Relationship**

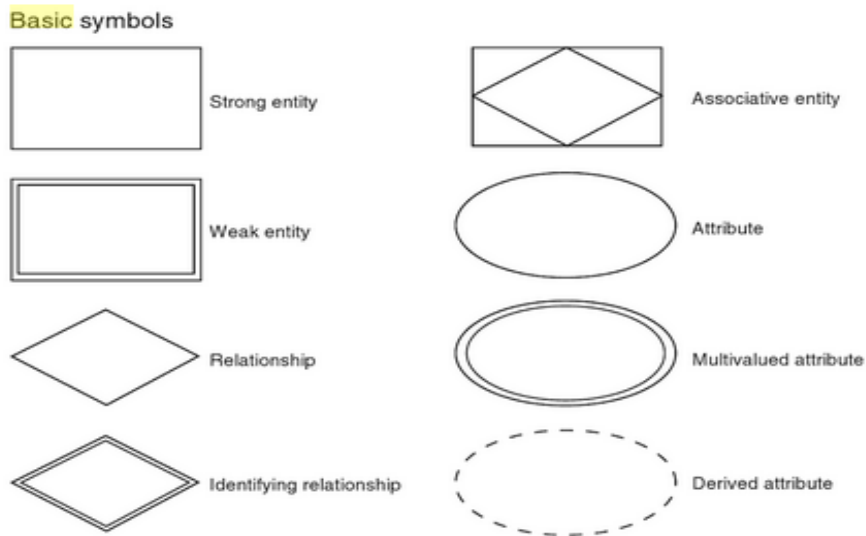
ER Diagram is used to represent database schema. In ER diagram:

- A rectangle represents an entity set.
- An ellipse represents an attribute.
- A diamond represents a relationship.
- Lines represent linking of attributes to entity sets and of entity sets to relationship sets.



### **Symbols used in ER Diagram:**

The elements in ER diagram are entity, attribute, and relationship. The different types of entities like strong, weak, and associative entity, different types of attributes like multivalued and derived attributes and identifying relationship and their corresponding symbols are shown here.



1) **Entity:** An entity is an object that exists in the world and which is distinguishable from other objects. In other words the entity can be uniquely identified.

**Examples:**

- A particular person, for ex: Dr. A P J Abdul Kalam is an entity.
- A particular department, for ex: Computer Science Department.
- A particular place, for ex: Bangalore city.

**Entity Type:** An entity type or entity set is a collection of similar entities. Some examples of entity types are:

- All students of SSKDC, say STUDENT.
- All courses in SSKDC, say COURSE.

An entity may belong to more than one entity type. For example a teaching staff working in a particular department may pursue higher education as a part time. So same person is a LECTURER at one instance and STUDENT at another instance.

2) **Attributes:** Attributes are properties of entity types. In other words, entities are described in a database by a set of attributes.

**Examples:**

- Brand, cost, and weight are the attributes of CELLPHONE.
- Roll number, name, and grade are the attributes of STUDENT.

Here CELLPHONE and STUDENT are entities.

Attributes are the following types.

- **Simple attribute:** Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute:** Composite attributes are made of more than one simple attribute, means it can be divided into subparts. For example, a student's complete *name* may have *first\_name* and *last\_name*.
- **Derived attribute:** Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.

For **example-1**, average\_salary in a department should not be saved directly in the database, instead it can be derived. For another **example-2**, age can be derived from data\_of\_birth.

- **Single-value attribute:** Single-value attributes contain single value. For example – Social\_Security\_Number, Aadhar card no.
- **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email\_address, etc.

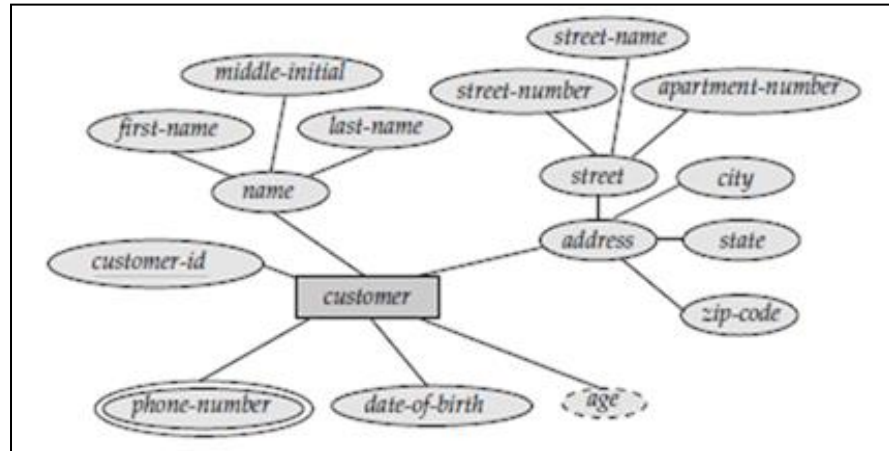


Fig: E-R-Diagram with composite, multivalued, and derived attributes.

**3) Relationship:** A relationship is an association among several entities. Where the association includes one entity from each participating entity type whereas relationship type is a meaningful association between entity types.

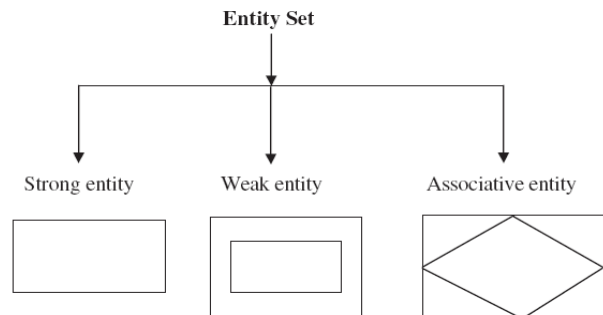
Examples:

- **Teaches** is the relationship type between LECTURER and STUDENT.
- **Buying** is the relationship between VENDOR and CUSTOMER.
- **Treatment** is the relationship between DOCTOR and PATIENT.

**Classification of Entity Sets:**

Entity sets can be broadly classified into:

- 1) Strong entity.
- 2) Weak entity.
- 3) Associative entity.



**1) Strong Entity**

Strong entity is one whose existence does not depend on other entity.

**Example**

Consider the example, student takes course. Here student is a strong entity.

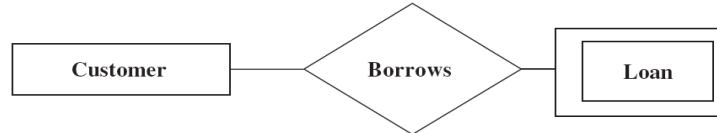


In this example, **course** is considered as weak entity because, if there are no **students** to take a particular course, then that course cannot be offered. The COURSE entity depends on the STUDENT entity. But STUDENT entity does not depend on any other entity. So **STUDENT** entity will be the strong entity.

**2) Weak Entity**

Weak entity is one whose existence depends on other entity. In many cases, weak entity does not have primary key.

**Example:** Consider the example, **customer borrows loan**. Here loan is a weak entity. For every loan, there should be at least one customer. Here the entity loan depends on the entity customer hence loan is a weak entity.

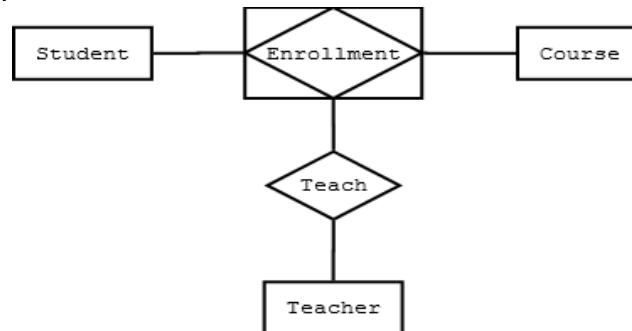


**3) Associative entity**

An associative entity is an entity type that associates the instances of more or many entity types. (or)

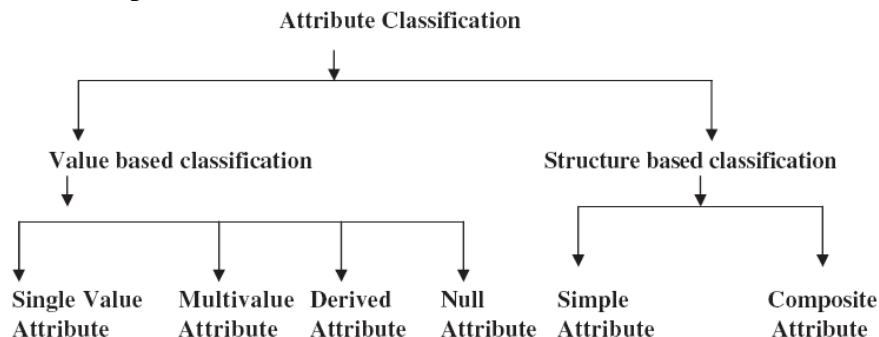
Associative where the entity describes a connection between two entities with an otherwise many-to-many relationship.

**Example:** STUDENTS enrollment to a COURSE, TEACHER teach to the STUDENTS to particular COURSE.



**Attribute Classification:**

Attribute is used to describe the properties of the entity. This attribute can be broadly classified based on **value** and **structure**. Based on value the attribute can be classified into single value, multivalued, derived, and null value attribute. Based on structure, the attribute can be classified as simple and composite attribute.



### Single Value Attribute

Single value attribute means, there is only one value associated with that attribute.

#### Example

The examples of single value attribute are age of a person, Roll number of the student, Registration number of a car, etc.

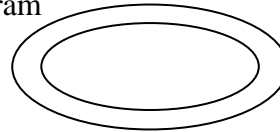
Representation of Single Value Attribute in ER Diagram:



### Multivalued Attribute

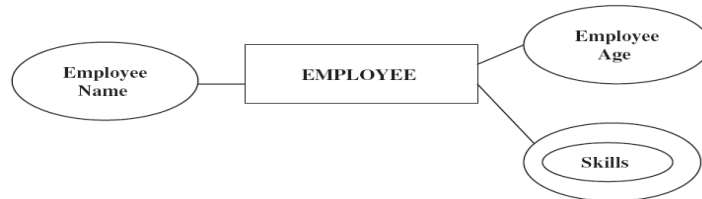
In the case of multivalued attribute, more than one value will be associated with that attribute.

Representation of Multivalued Attribute in ER Diagram

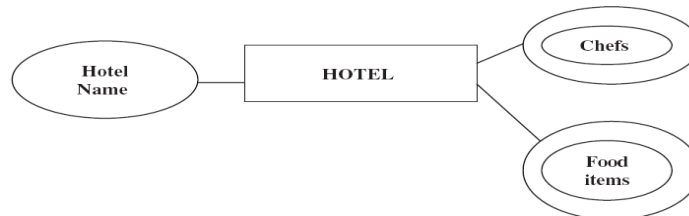


#### Examples -1 of Multivalued Attribute

Consider an entity EMPLOYEE: An Employee can have many skills; hence skills associated to an employee are a multivalued attribute.



Example-2: Number of chefs in a hotel is an example of multivalued attribute. Moreover, a hotel will have variety of food items. Hence food items associated with the entity HOTEL is an example of multivalued attribute. (Diagram in next page)



### Derived Attribute

The value of the derived attribute can be derived from the values of other related attributes or entities.

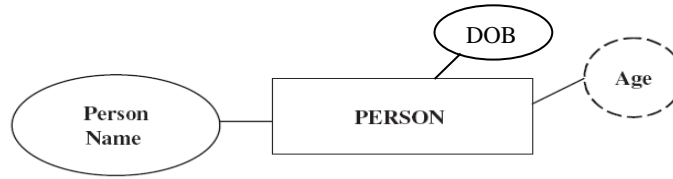
In ER diagram, the derived attribute is represented by dotted ellipse.

Representation of Derived Attribute in ER Diagram



#### Examples of Derived Attribute

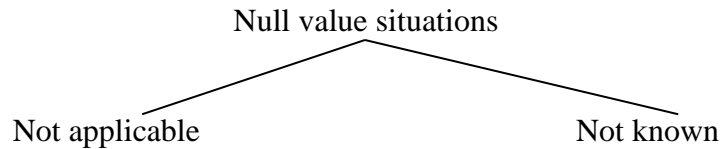
1. Age of a person can be derived from the date of birth of the person. In this example, age is the derived attribute.



2. **Experience** of an employee in an organization can be derived from date of joining of the employee.

### Null Value Attribute

In some cases, a particular entity may not have any applicable value for an attribute. For such situation, a special value called null value is created.



### Example

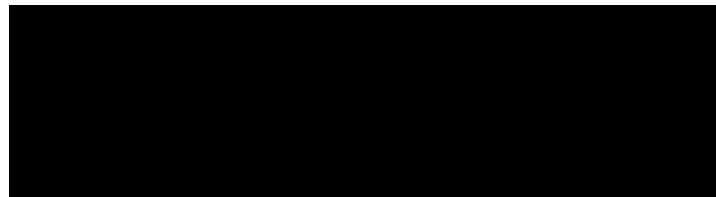
In application forms, there is one column called phone no. if a person do not have phone then a null value is entered in that column.

### Composite Attribute

Composite attribute is one which can be further subdivided into simple attributes.

#### Example

Consider the attribute “**address**” which can be further subdivided into Street name, City, State and Pincode.



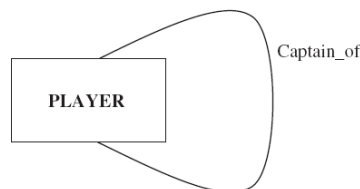
### Relationship Degree:

Relationship degree refers to the number of associated entities. The relationship degree can be broadly classified into

- 1) **Unary** Relationship,
- 2) **Binary** Relationship,
- 3) **Ternary** Relationship.

#### 1) Unary Relationship:

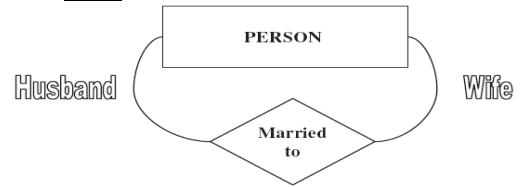
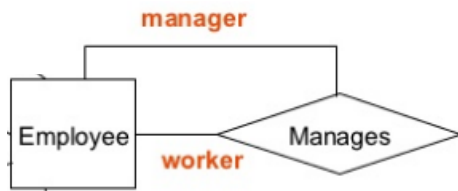
The unary relationship is otherwise known as recursive relationship. In the unary relationship the number of associated entity is one. An entity related to itself is known as recursive relationship.



### Roles and Recursive Relation

When an entity sets appear in more than one relationship, it is useful to add labels to connecting lines. These labels are called as roles.

**Example:** In this example, Husband and wife are referred as roles.



## 2) Binary Relationship

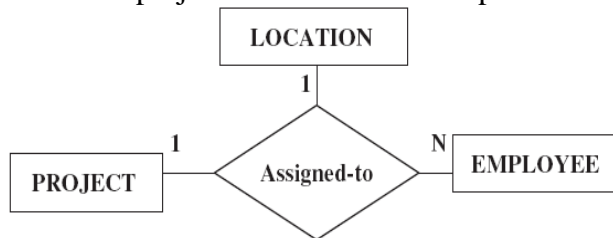
In a binary relationship, two entities are involved. Consider the **example**; each staff will be assigned to a particular department. Here the two entities are STAFF and DEPARTMENT.



## 3) Ternary Relationship

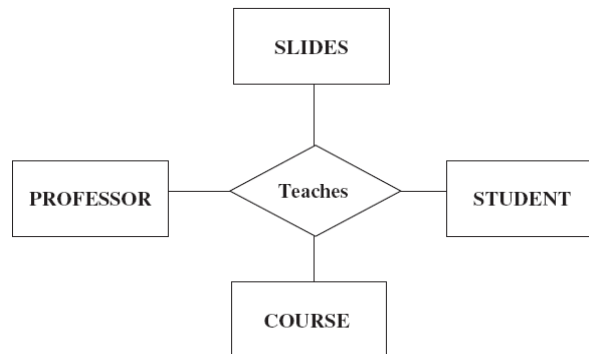
In a ternary relationship, three entities are simultaneously involved. Ternary relationships are required when binary relationships are not sufficient to accurately describe the semantics of an association among three entities.

**Example:** Consider the example of employee assigned a project. Here we are considering three entities EMPLOYEE, PROJECT, and LOCATION. The relationship is “assigned-to.” Many employees will be assigned to one project hence it is an example of one-to-many relationship.



## 4) Quaternary Relationships

Quaternary relationships involve four entities. The **example** of quaternary relationship is “A professor teaches a course to students using slides.” Here the four entities are PROFESSOR, SLIDES, COURSE, and STUDENT. The relationships between the entities are “Teaches.”





**Relationship Classification:**

Relationship is an association among one or more entities. This relationship can be broadly classified into

- 1) One-to-One Relation,
- 2) One-to-Many Relation,
- 3) Many- to - One Relation
- 4) Many- to - Many Relation.

**1) One-to-One Relationship:**

One-to-one relationship is a special case of one-to-many relationship. True one-to-one relationship is rare.

The relationship between the **President** and the **Country** is an **example** of one-to-one relationship. For a particular country there will be only one President. In general, a country will not have more than one President hence the relationship between the country and the President is an example of one-to-one relationship.

Another **example** of one-to-one relationship is **House to Location**. A house is obviously in only one location.

**2) One-to-Many Relationship:**

The relationship that associates one entity to more than one entity is called one-to-many relationship.

**Example** of one-to-many relationship is Country having multiple states. For one country there can be more than one state hence it is an example of one-to-many relationship.

**Example** parent-child relationship. For one parent there can be more than one child.

**3) Many-to-One Relationship:**

The relationship that associates more than one entity to one entity is called many-to-one relationship.




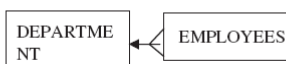




**Example** The relationship between EMPLOYEE and DEPARTMENT is an example of many-to-one relationship. There may be many EMPLOYEEES working in one DEPARTMENT.

**4) Many-to-Many Relationship:**

The relationship that associates more than one entity to more than one entity is called many-to-many relationship.

**Example** The relationship between EMPLOYEE entity and PROJECT entity is an example of many-to-many relationship. Many employees will be working in many projects. The four relationship types are summarized and shown in Table 2.1.

Table 2.1. Relationship types

Relationship type	Representation	Example
One-to-one		
One-to-many		
Many-to-many		
Many-to-one		

## **Reducing ER Diagram to Tables:**

To implement the database, it is necessary to use the Relational Model (table). There is a simple way of mapping from ER model to the relational model. There is almost one-to-one correspondence between ER constructs and the relational ones.

### **Rules:**

The basic rule for converting the ER diagrams into tables is

1) Convert all the Entities in the diagram to tables.

All the entities represented in the rectangular box in the ER diagram become independent tables in the database.

2) All single valued attributes of an entity is converted to a column of the table.

All the attributes, whose value at any instance of time is unique, are considered as columns of that table.

3) Key attribute in the ER diagram becomes the Primary key of the table.

4) Declare the foreign key column, if applicable.

5) Any multi-valued attributes are converted into new table.

6) Any composite attributes are merged into same table as different columns.

7) One can ignore derived attribute, since it can be calculated at any time.

These are the very basic rules of converting ER diagram into tables and columns, and assigning the mapping between the tables.

Here we can see the conversion of ER diagrams in to tables in different situations.

### **Regular Entity**

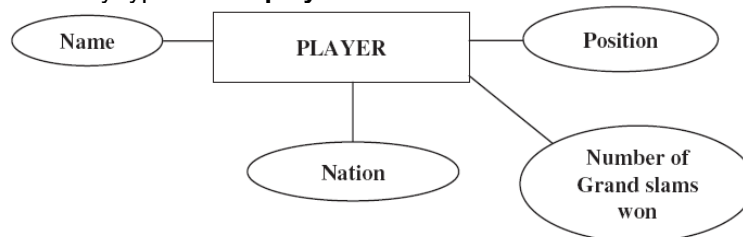
Regular entities are entities that have an independent existence and generally represent real-world objects such as persons and products. Regular entities are represented by rectangles with a single line.

## **1) Mapping Regular Entities to Tables (Converting Strong Entity into Table):**

- Each regular entity type in an ER diagram is transformed into a Relation (table). The name given to the relation is generally the same as the entity type.
- Each simple attribute of the entity type becomes an attribute/column of the relation.
- The identifier of the entity type becomes the primary key of the corresponding relation.

### **Example 1**

Mapping regular entity type **tennis player**



This ER diagram is converted into corresponding table as

**PLAYER**

Player Name	Nation	Position	Number of Grand slams won
Roger Federer	Switzerland	1	5
Roddick	USA	2	4

Here

**Entity name = Name of the relation or table**

In our above example, the entity name is PLAYER which is the name of the table.

**Attributes of ER diagram=Column name of the table.**

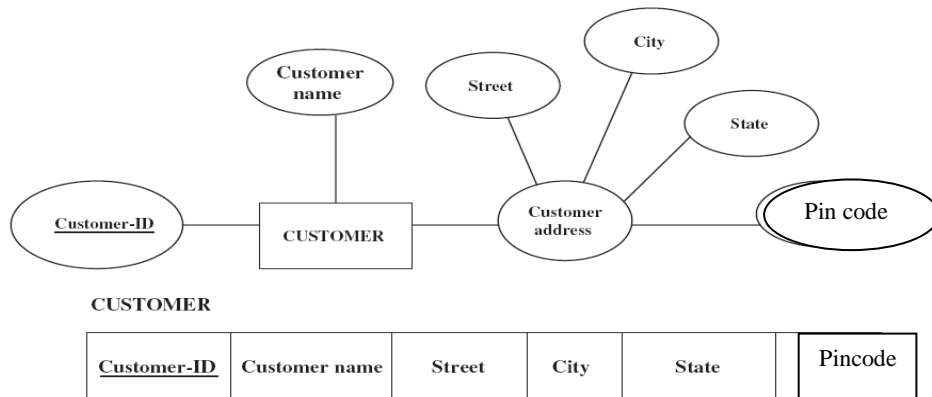
In our example the Name, Nation, Position, and Number of Grand slams won which forms the **column** of the table.

## 2) Converting Composite Attribute in an ER Diagram to Tables:

When a regular entity type has a composite attribute, only the simple component attributes of the composite attribute are included in the relation.

### Example

In this example the composite attribute is the Customer address, which consists of Street, City, State, and Pincode.



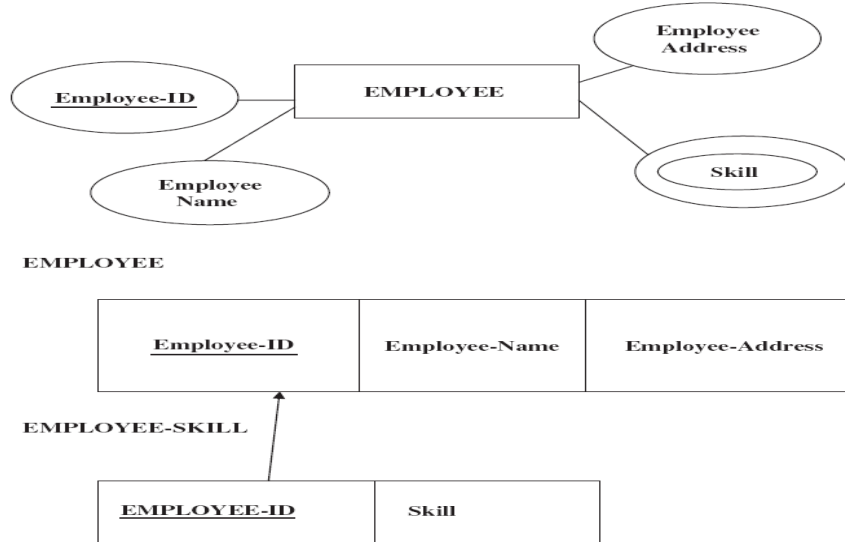
In this above ER Diagram customer-ID, Customer name are simple attributes so they converted as columns as in the table. But where as in composite attribute i.e., customer address subdivided into four parts, only subparts will be represented as columns in the table. Customer address will not be shown in the table.

## 3) Mapping Multivalued Attributes in ER Diagram to Tables:

A multivalued attribute is having more than one value. One way to map a multivalued attribute is to create two tables.

**Example:**

In this example, the **skill** associated with the EMPLOYEE is a multivalued attribute, since an EMPLOYEE can have more than one skill as fitter, electrician, turner, etc.



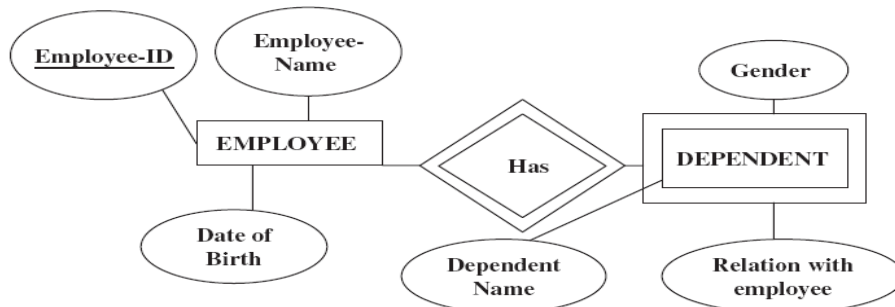
When the regular entity type contains a multivalued attribute, **two new relations or tables** are created.

The **first relation / table** contains all of the attributes of the entity type except the multivalued attribute.

The **second relation / table** contains two attributes that form the **primary key** of the second relation. The first of these attributes is the **primary key from the first relation**, which becomes a foreign key in the second relation. The second is the multivalued attribute.

**4) Converting “Weak Entities” in ER Diagram to Tables:**

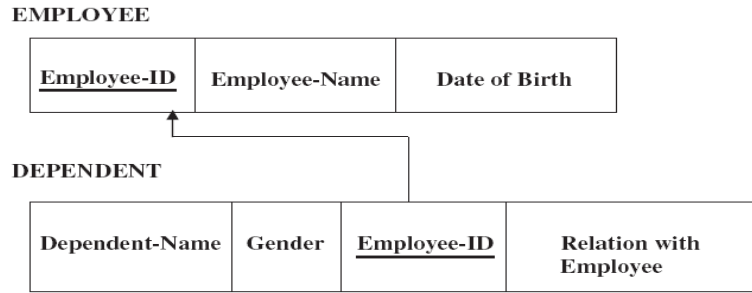
Weak entity type does not have an independent existence and it exists only through an identifying relationship with another entity type called the owner strong entity.



For each weak entity type, create a new relation and include all of the simple attributes as attributes of the relation. Then include the primary key of the identifying relation as a foreign key attribute to this new relation.

The primary key of the new relation is the combination of the primary key of the identifying and the partial identifier of the weak entity type. In this example DEPENDENT is weak entity.

The above ER Diagram converted into table as shown below:



### 5) Converting Binary Relationship to Table:

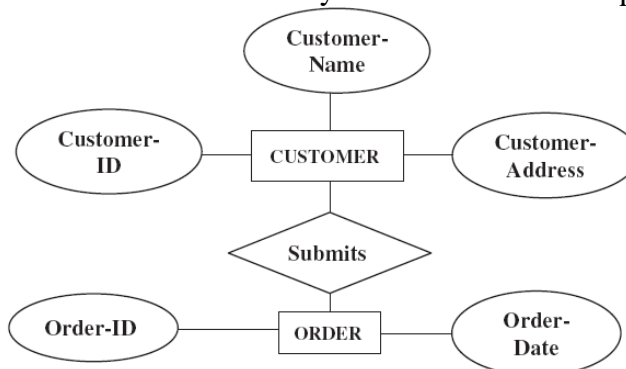
A relationship which involves two entities can be termed as binary relationship. This binary relationship can be one-to-one, one-to-many, many-to-one, and many-to-many.

#### a) Mapping one-to-Many Relationship

For each 1–M relationship, first create a relation for each of the two entity type’s participation in the relationship.

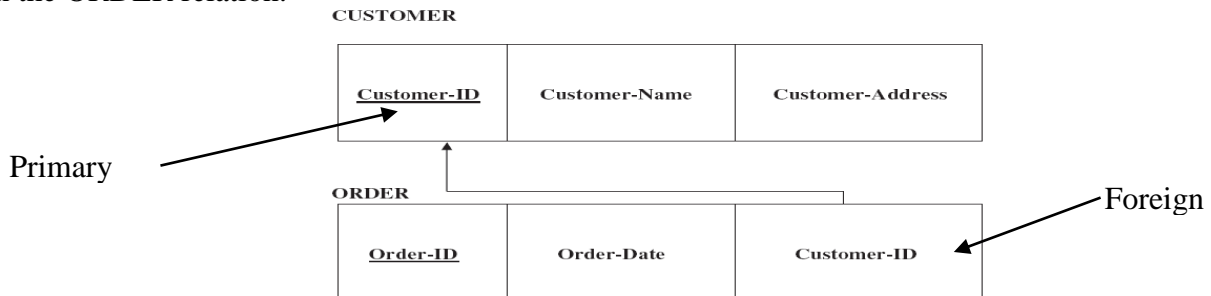
Example:

One customer can give many orders. Hence the relationship between the two entities CUSTOMER and ORDER is one-to-many relationship. In one-to many relationship, include the primary key attribute of the entity on the one-side of the relationship as a foreign key in the relation that is on the many side of the relationship.



Here we have two entities **CUSTOMER** and **ORDER**. The relationship between CUSTOMER and ORDER is one-to-many. For two entities CUSTOMER and ORDER, two tables namely CUSTOMER and ORDER are created as below.

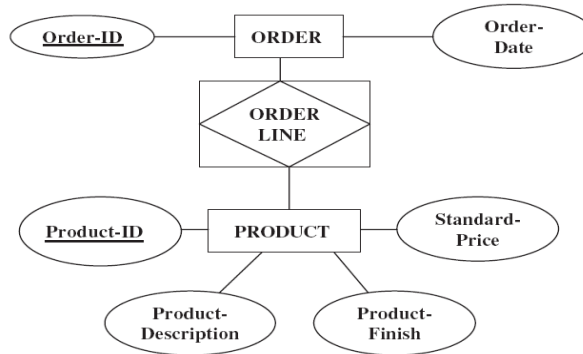
The primary key CUSTOMER ID in the CUSTOMER relation becomes the foreign key in the ORDER relation.



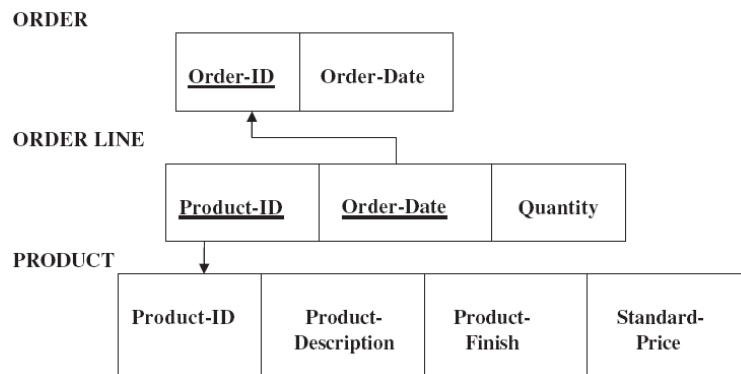
**b) Mapping Associative Entity to Tables**

Many-to-many relationship can be modeled as an associative entity in the ER diagram. Example 1. (Without Identifier)

Here the associative entity is ORDERLINE, which is without an identifier. That is the associative entity ORDERLINE is without any key attribute.



The first step is to **create three relations**, one for each of the two participating entity types and the third for the associative entity. The relation formed from the associative entity is associative relation.

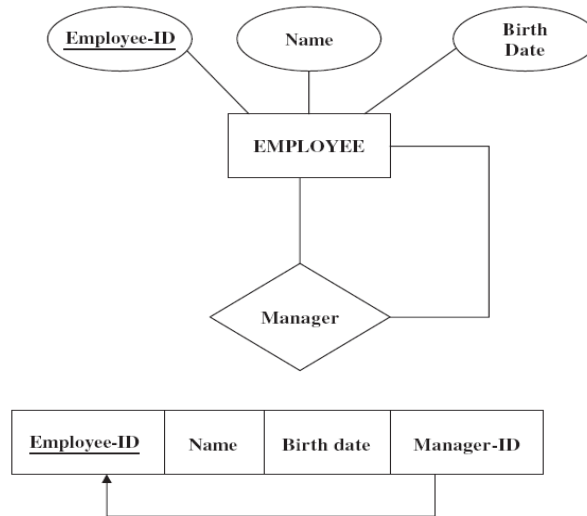


**6) Converting Unary Relationship to Tables:**

Unary relationships are also called recursive relationships. The two most important cases of unary relationship are one-to-many and many-to-many.

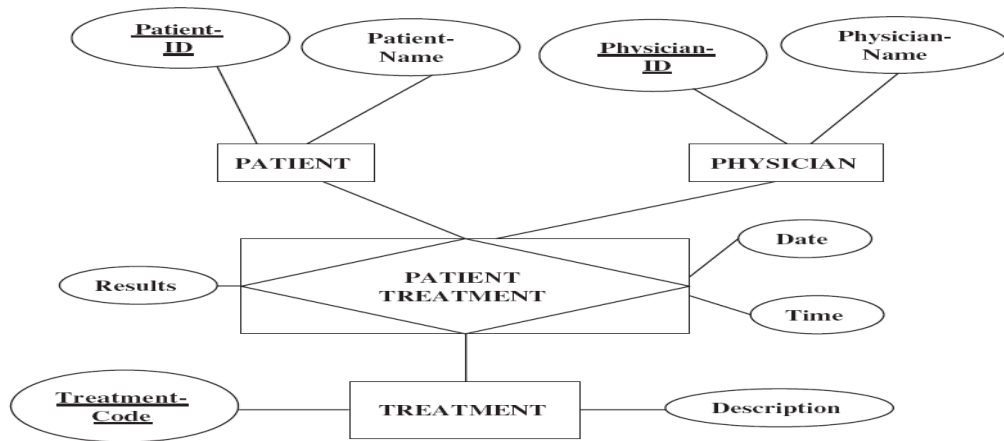
**a) One-to-many Unary Relationship**

Each employee has exactly one manager. A given employee may manage zero to many employees. The foreign key in the relation is named Manager-ID. This attribute has the same domain as the primary key Employee-ID.



**7) Converting Ternary Relationship to Tables:**

A ternary relationship is a relationship among three entity types. The three entities given in this example are **PATIENT**, **PHYSICIAN**, and **TREATMENT**. The **PATIENT-TREATMENT** is an associative entity.



The primary key attributes – Patient ID, Physician ID, and Treatment Code – become foreign keys in **PATIENT TREATMENT**. These attributes are components of the primary key of **PATIENT TREATMENT**.

PATIENT TREATMENT

<u>Patient-ID</u>	Patient-Name
-------------------	--------------

PHYSICIAN

<u>Physician-ID</u>	Physician-Name
---------------------	----------------

PATIENT TREATMENT

<u>Patient-ID</u>	<u>Physician-ID</u>	<u>Treatment-Code</u>	<u>Date</u>	<u>Time</u>	Results
-------------------	---------------------	-----------------------	-------------	-------------	---------

TREATMENT

<u>Treatment-Code</u>	Description
-----------------------	-------------

**Enhanced Entity–Relationship Model (EER Model):**

The basic concepts of ER modeling are not powerful enough for some complex applications. Hence some additional semantic modeling concepts are required, which are being provided by Enhanced ER model.

The **Enhanced ER model** is the extension of the original ER model with new modeling constructs. The new modeling constructs introduced in the **EER** model are supertype (superclass)/ subtype (subclass) relationships.

The supertype allows us to model general entity type whereas the subtype allows us to model specialized entity types.

$$\text{Enhanced ER model} = \text{ER model} + \text{hierarchical relationships.}$$

EER modeling is especially useful when the domain being modeled is object-oriented in nature and the use of inheritance reduces the complexity of the design. The extended ER model **extends** the ER model to allow various types of abstraction to be included and to express constraints more clearly.

**Supertype or Superclass**

Supertype **or** superclass is a generic entity type that has a relationship with one or more subtypes.

For example PLAYER is a generic entity type which has a relationship with one or more subtypes like CRICKET PLAYER, FOOTBALL PLAYER, HOCKEY PLAYER, TENNIS PLAYER, etc.

**Subtype or Subclass**

A subtype **or** subclass is a subgrouping of the entities in an entity type that is meaningful to the organization.

A subclass entity type represents a subset or subgrouping of superclass entity type's instances. Subtypes **inherit** the attributes and relationships associated with their supertype.

Consider the entity type ENGINE, which has two subtypes PETROL ENGINE and DIESEL ENGINE.

Consider the entity type STUDENT, which has two subtypes UNDERGRADUATE and POSTGRADUATE.



## **Generalization and Specialization:**

Generalization and specialization are two words for the same concept, viewed from two opposite directions.

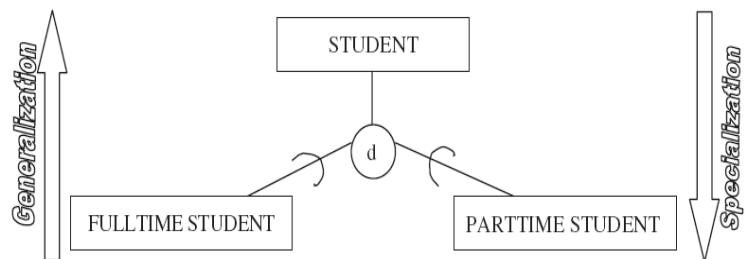
**Generalization** is the bottom-up process of defining a generalized entity type from a set of more specialized entity types.

**Specialization** is the top-down process of defining one or more subtypes of a supertype.

Generalization is the process of minimizing the differences between entities by identifying common features. It can also be defined as the process of defining a generalized entity type from a set of entity types.

Specialization is a process of identifying subsets of an entity set (the superset) that share some distinguishing characteristics. In specialization the superclass is defined first and the subclasses are defined next. Specialization is the process of viewing an object as a more refined, specialized object. Specialization emphasizes the differences between objects.

**Example** consider the entity type STUDENT, which can be further classified into FULLTIME STUDENT and PARTTIME STUDENT. The classification of STUDENT into FULLTIME STUDENT and PARTTIME STUDENT is called Specialization.

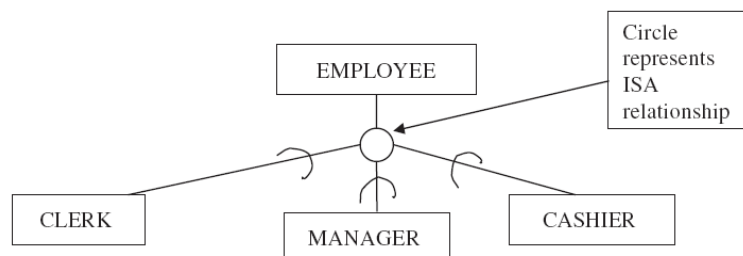


## **ISA Relationship and Attribute Inheritance:**

**IS A relationship** supports attribute inheritance and relationship participation. In the EER diagram, the subclass relationship is represented by **ISA relationship**.

**Attribute inheritance** is the property by which subclass entities inherit values for all attributes of the superclass.

Consider the **example** of EMPLOYEE entity set in a bank. The EMPLOYEE in a bank can be CLERK, MANAGER, CASHIER, ACCOUNTANT, etc. It is to be observed that the CLERK, MANAGER, CASHIER, ACCOUNTANT inherit some of the attributes of the EMPLOYEE.



In this example the superclass is EMPLOYEE and the subclasses are CLERK, MANAGER, and CASHIER. The subclasses inherit the attributes of the superclass. Since each member of the subclass is an IS A member of the superclass, the circle below the EMPLOYEE entity set represents IS A relationship.

## **Multiple Inheritance:**

A subclass with more than one superclass is called a shared subclass. A subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclass, that is it has multiple inheritance from its superclasses. In multiple inheritance a subclass can be subclass of more than one superclass.

**Example of Multiple Inheritance**

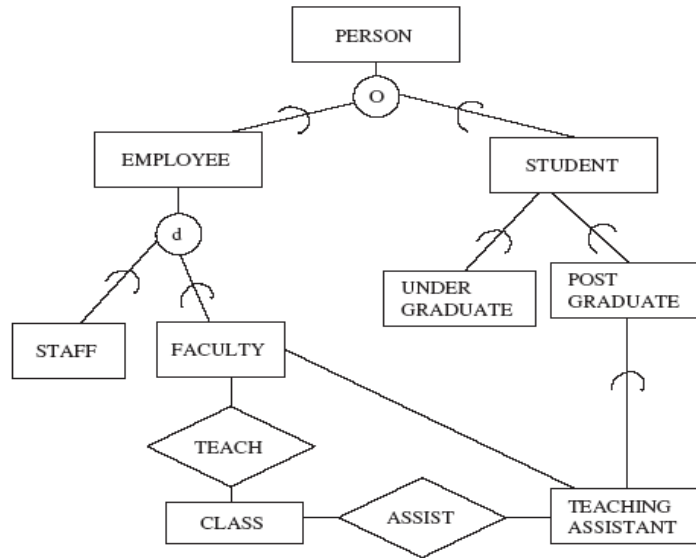


Fig. 2.2. Multiple inheritance

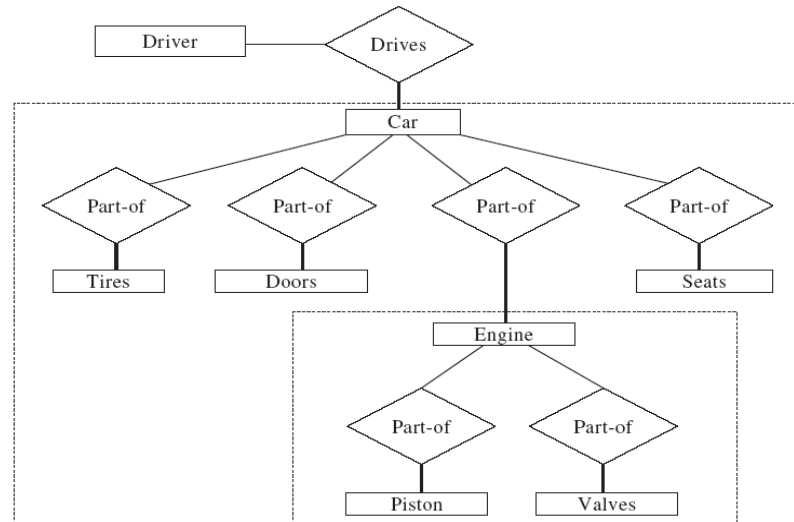
Consider a **person** in an educational institution. The **person** can be employee, alumnus, and student. The **employee** entity can be staff or faculty. The student can be a graduate student or a postgraduate student. The **postgraduate** student can be a teaching assistant. If the postgraduate student is a teaching assistant, then he/she inherits the characteristics of the faculty as well as student class. That is the teaching assistant subclass is a subclass of more than one superclass (faculty, student). This phenomenon is called multiple inheritance and is shown in the Fig. 2.2.

**Aggregation and Composition:**

Relationships among relationships are not supported by the ER model. Groups of entities and relationships can be abstracted into higher level entities using aggregation.

Aggregation represents a “**HAS-A**” or “**IS-PART-OF**” relationship between entity types. One entity type is the whole, the other is the part. Aggregation allows us to indicate that a relationship set participates in another relationship set.

Example of a driver driving a car. The car has various components like tires, doors, engine, seat, etc., which varies from one car to another. **Relationship drives is insufficient to model** the complexity of this system. *Partof* relationships allow abstraction into higher level entities. In this example engine, tires, doors, and seats are aggregated into car.



Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Consider the **example** of DEPARTMENT has PROJECT. Each project is associated with a particular DEPARTMENT. There cannot be a PROJECT without DEPARTMENT. Hence DEPARTMENT has PROJECT is an example of composition.

### **Advantages of ER Modeling:**

An ER model is derived from business specifications. ER models separate the information required by a business from the activities performed within a business. Although business can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant. The advantages of ER modeling are summarized later:

1. The ER modeling provides an easily understood pictorial map for the database design.
2. It is possible to represent the real world problems in a better manner in ER modeling.
3. The conversion of ER model to relational model is straightforward.
4. The enhanced ER model provides more flexibility in modeling real world problems.
5. The symbols used to represent entity and relationships between entities are simple and easy to follow.

## **SCHEMA REFINEMENT AND NORMALIZATION:**

### **Introduction to Schema Refinement:**

A schema can be defined as a complete description of the Database.

Any relation can be represented with an entity relation(ER) diagram. Every Relation has a ER Diagram with attributes representing the columns of the tables and entity representing the name of the tables.

Schema refinement is intended to address and a refinement approach based on the decompositions.

Redundant is a storage of information is the root cause of these problems. Although decomposition can eliminate redundancy, it can lead to problem of its own and should be used with caution.

**Problems Caused by Redundancy:**

Storing the same information is called Redundantly, i.e. in more than one place with a database.

**Redundant Storage:** Some information is stored repeatedly.

**Update Anomalies:** If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

**Insertion Anomalies:** It may not be possible to store certain information unless some other, unrelated, information is stored as well.

**Deletion Anomalies:** It may not be possible to delete certain information without losing some other, unrelated, information as well.

**Example:** Problems due to R W:

- Update anomaly: Can we change W in just the 1st tuple of SNLRWH?
- Insertion anomaly: What if we want to insert an employee and don't know the hourly wage for his rating?
- Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smily	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

**Table : S N L R W H**

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smily	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	5	40

**Table : S N L R H**

R	W
8	10
5	7

**Table: R W (Hourly\_Emps2 Wages)**

## Decomposition of a Relation Scheme:

Suppose that relation R contains attributes A1 ... An. A decomposition of R consists of replacing R by two or more relations such that:

- Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
- Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

## Reasoning About FDs:

Given some FDs, we can usually infer additional FDs:

$ssn \rightarrow did, did \rightarrow lot$  implies  $ssn \rightarrow lot$

An FD f is implied by a set of FDs F if f holds whenever all FDs in F hold.

$F^+$  = closure of F is the set of all FDs that are implied by F.

Armstrong's Axioms (X, Y, and Z are sets of attributes):

- **Reflexivity:** If  $X \supseteq Y$ , then  $X \rightarrow Y$
- **Augmentation:** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any Z
- **Transitivity:** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$

Couple of additional rules while reasoning about  $F^+$

- **Union:** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$
- **Decomposition:** If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

## Problems with Decompositions:

There are three potential problems to consider:

1) Some queries become more expensive.

e.g., How much did sailor Joe earn? (Salary = W\*H)

2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!

Fortunately, not in the SNLRWH example.

3) Checking some dependencies may require joining the instances of the decomposed relations.

Fortunately, not in the SNLRWH example.

### Lossless Join Decompositions:

1) Decomposition of R into X and Y is lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:

$$\Pi^X(r) \bowtie (\text{join}) \Pi^Y(r) = r$$

2) It is always true that  $r \mathbf{C} = \Pi^X(r) \bowtie (\text{join}) \Pi^Y(r)$

In general, the other direction does not hold! If it does, the decomposition is lossless-join.

3) Definition extended to decomposition into 3 or more relations in a straightforward way.

### NORMAL FORMS:

Normalization is a process of converting Relation form into a standard form. We need decompose a relation into smaller efficient tables/relations that depicts a good DB design.

With the help of primary key, different relations can be analyzed. This technique is called *Normalization*. Normalization technique involves a sequence of rules that are employed to test individual relations.

The process of Normalization is based on the concept of Normal Forms Each and every normal form has its own set of properties and set of constraints.

These properties/conditions are usually applied on the attributes of the tables and also on the relationships among them.

Different levels of normal forms are used to satisfy different factors and helps in reducing data maintains anomalies.

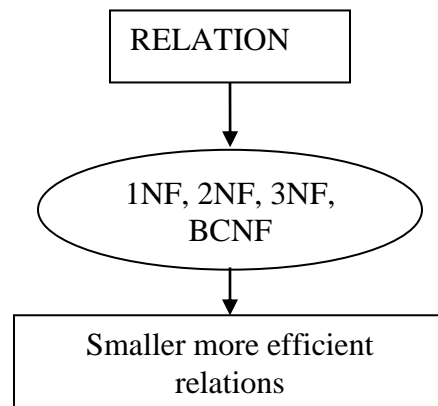
### Different Normal Forms are:

1) 1NF

2) 2NF

3) 3NF

4) BCNF



**1 NF:** A Relation schema is said to be in 1NF, if the values in the relation are atomic. In simple words we can say there should be no repeating groups in a particular column. A value can be defined as an atomic value if it doesn't contain a set of values.

A table is said to contain atomic values. If there is a unique value of data item for any given row and column intersection.

*1NF* sets the very basic rules for an organized Database.

- Eliminating the Duplicate columns from the same table (or) given table.
- Create separate tables for each group of related data and identifying each row with a unique column of set of columns.(the Primary key)

**2 NF:** *2 NF* further address the concept of removing the duplicate data

- Meet all the requirements of the 1NF
- Remove subsets of data that apply to multiple rows of table and place them in separate tables.
- Create relationships between these new tables and their predecessors through the use of the Foreign key.

**3 NF:** This *3NF* is based on the concept of the Transitive Dependency. A functional Dependency  $x \rightarrow y$  in a relation schema R is a Transitive Dependency. A relation is said to be in 3NF. If every department is a key, i.e for each and every functional dependency

$$3NF \rightarrow 2NF \rightarrow 1NF$$

This 3NF goes to one large step further.

- Meet all the requirements of the 2NF
- Remove columns that are not dependent upon the primary key.

**4NF:** Finally *4NF* has one additional requirement

- Meet all the requirements of the 3NF
- A relation is in 4NF if it has no multivalued dependencies.

### **BCNF: Boyce - Codd Normal Form**

Every Relation that is in BCNF, is also in 3NF and therefore in 2NF and hence is in 1NF

Let 'R' be a relation schema 'F' be the set of functional dependencies given to hold on 'R', and 'X' be a subset of the attributes of 'R' and 'A' be an attribute of 'R'. 'R' is said to be BCNF(Boyce-Codd Normal Form) , if every Functional Dependency  $X \twoheadrightarrow A$  in F, one of the following is true.

- $A \in X$  [i.e it is a trivial FD] or
- X is a super key

In a BCNF relation, the only nontrivial dependencies are those in which a key determines few attributes. Thus each tuple can be thought of as an entity or relationship, identified by a key and described by

remaining attributes. If suppose we use ovals for representing attributes or sets of attributes and draw arcs for indicating FDs.

**Q) Normalize the Relation R (A,B,C,D,E,F,G,H) into the 3NF using the following set of FD's**

**$AB \rightarrow C$**

**$BC \rightarrow D$**

**$CDE \rightarrow ABH$**

**$BH \rightarrow A$**

**$D \rightarrow EF$**

( *EXTERNAL EXAM* )

**Ans:** Given Relation  $AB \rightarrow C$   $BC \rightarrow D$   $CDE \rightarrow ABH$   $BH \rightarrow A$   $D \rightarrow EF$

Since  $BC \rightarrow D$  and  $D \rightarrow EF$  we can write

$BC \rightarrow EF$  }  $\rightarrow \rightarrow \rightarrow \rightarrow$  (Transitivity Rule)

As  $CDE \rightarrow ABH$  we can write

$CDE \rightarrow A$  }  $\rightarrow \rightarrow \rightarrow \rightarrow$  (Decomposition Rule)

$CDE \rightarrow B$  } OR  $CDE \rightarrow A$

$CDE \rightarrow H$  }  $CDE \rightarrow BH$

Since  $CDE \rightarrow BH$  and  $BH \rightarrow A$  we can write

$CDE \rightarrow A$  } (Transitivity Rule)

The decomposition is dependency preserving.

**Multivalued Dependency: (MVD)**

To understand the concept of multivalued dependencies, consider a relation.

Course	Student	TextBook
Physics	Raju	Principles of Science
Physics	Raju	ABC of Physics
Physics	Ramu	Principles of Science
Physics	Ramu	ABC of Physics
Chemistry	Raju	Principles of Science
Chemistry	Raju	Optics
Chemistry	Raju	Optical chemistry

Each of the tuple means that the course C is taken be the student S and the text book 'T'. The attributes student and textbooks independent. Any no. of students can refer any textbook and can take any course.

The composite key for this relation consist of *CST*.



This redundancy again gives rise to update anomalies. By decomposing this relation into two schemas with attributes CS and CT.

CS Course	Student	CT Course	Teacher
Physics	Raju	Physics	ABC of Physics
Physics	Ramu	Physics	Principles of Science
Chemistry	raju	Chemistry	Principles of Science
		Chemistry	Optics
		Chemistry	Optical Chemistry

The MVD are generalization of a FD. They can be represented as,

Course  $\twoheadrightarrow$  Student

Course  $\twoheadrightarrow$  TextBook

Example:  $x \twoheadrightarrow y$       Every FD is MVD

i.e.,  $X \twoheadrightarrow Y$  This also means  $X \twoheadrightarrow Y$

5 rules are used to compute additional Functional Dependency and MVD. These are

**MVD Complementation:** If  $A \twoheadrightarrow B$  then  $A \twoheadrightarrow R - AB$

**MVD Augmentation:** If  $A \twoheadrightarrow B$  and  $C \twoheadrightarrow D$  then  $AD \twoheadrightarrow BC$

**MVD Transitivity:** If  $A \twoheadrightarrow B$  and  $B \twoheadrightarrow C$  then  $A \twoheadrightarrow (C - B)$

**Replication:** If  $A \twoheadrightarrow B$  THEN  $A \twoheadrightarrow B$

**Coalescence:** If  $A \twoheadrightarrow B$  and there is a C such that  $B \wedge C$  is empty  $C \twoheadrightarrow D$  and  $D \twoheadrightarrow B$ , then  $A \twoheadrightarrow D$ .

## INDEX

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

**For example**, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data. Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

### **The CREATE INDEX Command:**

The basic syntax of CREATE INDEX is as follows:

```
CREATE INDEX index_name ON table_name;
```

#### Single-Column Indexes:

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

```
CREATE INDEX index_name  
ON table_name (column_name);
```

#### Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

```
CREATE UNIQUE INDEX index_name  
on table_name (column_name);
```

#### Composite Indexes:

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

```
CREATE INDEX index_name  
on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the columns that you may use very frequently in a query's WHERE clause as filter conditions. Should there be only one column used, a single-column index should be the choice. Should there

be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

#### Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

### **The DROP INDEX Command:**

An index can be dropped using SQL DROP command. Care should be taken when dropping an

index because performance may be slowed or improved.

The basic syntax is as follows:

**DROP INDEX index\_name;**

## **Unit 4**

### **Query Processing in DBMS**

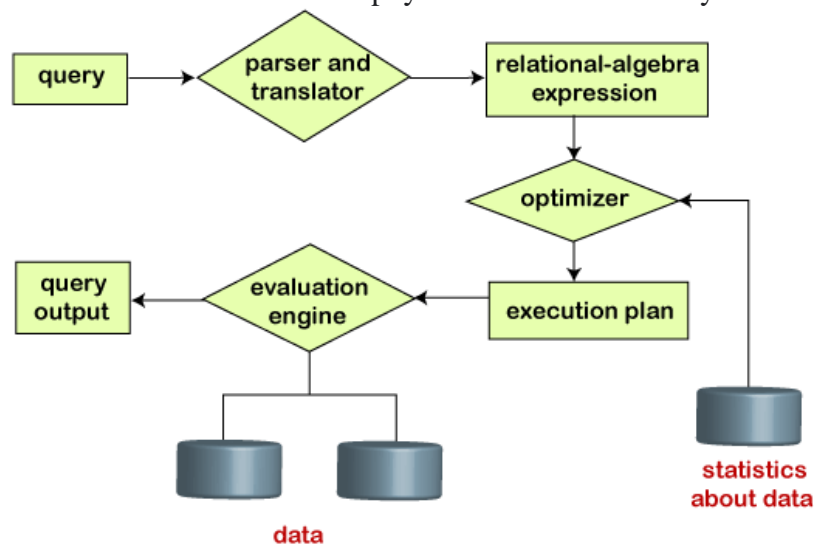
Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

1. Parsing and translation
2. Optimization
3. Evaluation

The query processing works in the following way:

#### **Parsing and Translation**

As query processing **includes certain activities for data retrieval**. Initially, the given user queries get translated in high-level database languages such as SQL. It gets translated into expressions that can be further used at the physical level of the file system.



**Steps in query processing**

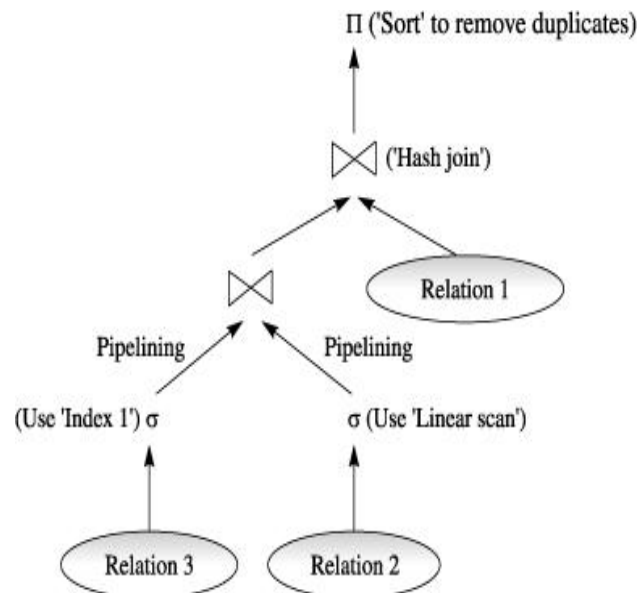
#### **Optimization**

Query Optimization in DBMS is **the process of selecting the most efficient way to execute a SQL statement**. Because SQL is a nonprocedural language, the optimizer can merge, restructure, and process data in any sequence. Query Optimization in DBMS has a significant application in database designing.

#### **Evaluation**

An evaluation plan is used to define exactly what algorithm should be used for each operation and how the execution of the operations should be coordinated. So far we have

discussed mainly two basic approaches to choosing an execution (action) plan namely, (a) heuristic optimization and (b) cost-based optimization.



### Selection Operation with Indexes

The index-based search algorithms are known as **Index scans**. Such index structures are known as **access paths**. These paths allow locating and accessing the data in the file. There are following algorithms that use the index in query processing:

- **Primary index, equality on a key:** We use the index to retrieve a single record that satisfies the equality condition for making the selection. The equality comparison is performed on the key attribute carrying a primary key.
- **Primary index, equality on nonkey:** The difference between equality on key and nonkey is that in this, we can fetch multiple records. We can fetch multiple records through a primary key when the selection criteria specify the equality comparison on a nonkey.
- **Secondary index, equality on key or nonkey:** The selection that specifies an equality condition can use the secondary index. Using secondary index strategy, we can either retrieve a single record when equality is on key or multiple records when the equality condition is on nonkey. When retrieving a single record, the time cost is equal to the primary index. In the case of multiple records, they may reside on different blocks. This results in one I/O operation per fetched record, and each I/O operation requires a seek and a block transfer.

## Selection Operations with Comparisons

For making any selection on the basis of a comparison in a relation, we can proceed it either by using the linear search or via indices in the following ways:

**Primary index, comparison:** When the selection condition given by the user is a comparison, then we use a primary ordered index, such as the primary B<sup>+</sup>-tree index. **For example**, when A attribute of a relation R compared with a given value v as A>v, then we use a primary index on A to directly retrieve the tuples. The file scan starts its search from the beginning till the end and outputs all those tuples that satisfy the given selection condition.

**Secondary index, comparison:** The secondary ordered index is used for satisfying the selection operation that involves <, >, ≤, or ≥ .

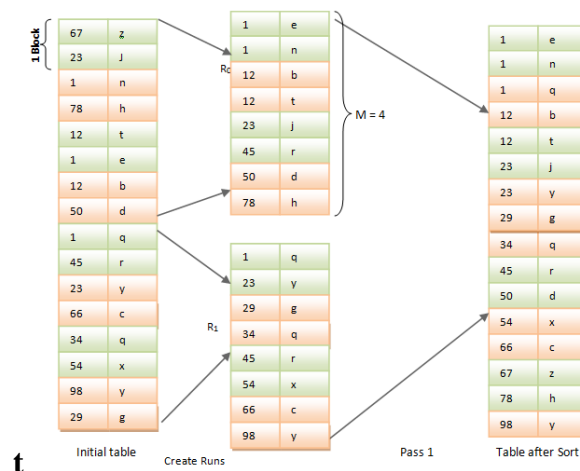
### Sorting Method in DBMS

It is the technique of storing the records in ascending or descending order of one or more columns. It is useful because, some of the queries will ask us to return sorted records, or in operations like joins will be more efficient in sorted records. All the records are by default sorted based on the primary key column. In addition, we can specify to sort the records based on other columns, as required. Two types of sorting methods are mainly used.

#### 1.Quick Sort

If a table size is small and can be accommodated into current memory, then quick sort can be used. As the name suggests it is simple and easy method of sorting. In this method a pivot element is identified among the values of the column and values less the pivot element is moved to the left of pivot and greater than pivot elements are moved to the right of the pivot. It takes very less additional space (n log (n)) to sort. It takes only n log (n) time to sort at best case and only n<sup>2</sup> time at worst case. But this method is less stable as it can alter the position of two similar records while sorting.

#### 2.Merge sort



## Evaluation of Expressions

In our previous sections, we understood various concepts in query processing. We learned about the query processing steps, selection operations, and also several types of algorithms used for performing the join operation with their cost estimations.

We are already aware of computing and representing the individual relational operations for the given user query or expression. Here, we will get to know how to compute and evaluate an expression with multiple operations.

For evaluating an expression that carries multiple operations in it, we can perform the computation of each operation one by one. However, in the query processing system, we use two methods for evaluating an expression carrying multiple operations. These methods are:

1. Materialization
2. Pipelining

Let's take a brief discussion of these methods.

### Materialization

In this method, the given expression evaluates one relational operation at a time. Also, each operation is evaluated in an appropriate sequence or order. After evaluating all the operations, the outputs are materialized in a temporary relation for their subsequent uses. It leads the materialization method to a disadvantage. The disadvantage is that it needs to construct those temporary relations for materializing the results of the evaluated operations, respectively. These temporary relations are written on the disks unless they are small in size.

### Pipelining

Pipelining is an alternate method or approach to the materialization method. In pipelining, it enables us to evaluate each relational operation of the expression simultaneously in a pipeline. In this approach, after evaluating one operation, its output is passed on to the next operation, and the chain continues till all the relational operations are evaluated thoroughly. Thus, there is no requirement of storing a temporary relation in pipelining. Such an advantage of pipelining makes it a better approach as compared to the approach used in the materialization method. Even the costs of both approaches can have subsequent differences in-between. But, both approaches perform the best role in different cases. Thus, both ways are feasible at their place.

## Measures of Query Cost in DBMS

Query Cost is a cost in which the enhancer considers what amount of time your query will require (comparative with absolute clump time). Then the analyzer attempts to pick the most ideal query plan by taking a glance at your inquiry and insights of your information, attempting a few execution designs, and choosing the most inexpensive of them.

The measures of query cost in DBMS can be done by creating a framework that can make numerous designs for an inquiry. It tends to be finished by the means of contrasting every conceivable arrangement as far as their assessed cost. For working out the net assessed cost of any arrangement, the expense of every activity inside an arrangement ought to be set in a deterministic and consolidated cost to get the net assessed cost of the query assessment plan.

**Example:** We utilize the number of square exchanges that is basically the block from the disk and the quantity of the disk seeks to appraise the expense of a query assessment plan. Assuming that the disk subsystem takes a normal of  $tT$  seconds to move a square of information and has a normal block access time (disk lookup time in addition to rotational idleness) of  $tS$  seconds, then, at that point, an activity that moves  $b$  obstructs and performs  $S$  looks for would take  $b * tT + S * tS$  seconds. The upsides of  $tT$  and  $tS$  should be aligned for the disk framework utilization, however, normal qualities for top-end disk today would be  $tS = 4$  milliseconds and  $tT = 0.1$  milliseconds, expecting a 4-kilobyte block size and an exchange pace of 40 megabytes each second.

## Transformation of Relational Expressions in DBMS

**Introduction:** When a query is submitted to the database, it is responsibility of database to determine algorithm to evaluate the query. The DBMS has to decide a low cost algorithm to evaluate the query and then optimal path to evaluate the query. The query may be simple or complex. Depending on the cost it has to pick better execution path. This is called query optimization. There are various factors affecting the performance of the query like number of records in the table, number of blocks allocated to each table, number of records in each block, size of the record, duplicate records, height of B+ tree, constraints and indexes etc. Let us see how to select a better performing query.

## Transformation of Relational Expressions

When a SQL query is submitted to DB, it can be evaluated in number of ways. For example, consider the below case:

```
SELECT EMP_ID, DEPT_NAME
FROM EMP, DEPT
WHERE EMP.DEPT_ID = DEPT.DEPT_ID
AND EMP.DEPT_ID = 10;
```

Above query selects the EMP\_ID and DEPT\_NAME from EMP and DEPT table for DEPT\_ID = 10. But when it is given to the DBMS, it divides the query into tokens and sees how it can be put together so that performance will be better. This is the duty of query optimizer. But altering the order of tokens in the query should not change the result. In either way it should give same result. Order of records can change and are least important. This is called equivalent query. There is set of rules to put tokens in the query. This is called equivalence rule.

Above query can be broken down by the DBMS in either ways below :

- Select the records of EMP with DEPT\_ID = 10 first then join them with DEPT table to get all matching records of EMP and DEPT. Then select only the columns EMP\_ID and DEPT\_NAME to display the result.
- Select all matching records from EMP and DEPT, from which filter on DEPT\_ID = 10 and select only EMP\_ID and DEPT\_NAME to display.

Both the steps above are same irrespective of how it is performed. Hence both are called equivalent query. These are not written in SQL, but using relational algebra, graph or tree.

$\Pi$  EMP\_ID, DEPT\_NAME ( $\sigma$  DEPT\_ID = 10 (EMP  $\bowtie$  DEPT))  
 or  $\sigma$  DEPT\_ID = 10 ( $\Pi$  EMP\_ID, DEPT\_NAME, DEPT\_ID (EMP  $\bowtie$  DEPT))

Above relational algebra and tree shows how DBMS depicts the query inside it. But the cost of both of them may vary. This is because the number of records in each step changes depending on the join and filters we use, and the algorithms used to evaluate them. For example we may have huge number of records in second case tree above to filter. But in the first case we are filtering the record first; hence number of records to join would have been reduced. This makes lots of difference and query optimizer calculates this difference and selects the optimal tree for query evaluation

### **Estimating Statistics of Expression results in DBMS**

In order to determine ideal plan for evaluating the query, it checks various details about the tables that are stored in the data dictionary. These informations about tables are collected when a table is created and when various DDL / DML operations are performed on it. The optimizer checks data dictionary for :

- Total number of records in a table, nr. This will help to determine which table needs to be accessed first. Usually smaller tables are executed first to reduce the size of the intermediary tables. Hence it is one of the important factors to be checked.
- Total number of records in each block, fr. This will be useful in determining blocking factor and is required to determine if the table fits in the memory or not.
- Total number of blocks assigned to a table, br. This is also an important factor to calculate number of records that can be assigned to each block. Suppose we have 100 records in a table and total number of blocks are 20, then fr can be calculated as  $nr/b = 100/20 = 5$ .
- Total length of the records in the table, lr. This is an important factor when the size of the records varies significantly between any two tables in the query. If the record



length is fixed, there is no significant affect. But when a variable length records are involved in the query, average length or actual length needs to be used depending upon the type of operations.

- Number of unique values for a column,  $d_A$ . This is useful when a query uses aggregation operation or projection. It will provide an estimate on distinct number of columns selected while projection. Number groups of records can be determined using this when Aggregation operation is used in the query. E.g.; SUM, MAX, MIN, COUNT etc.
- Selection cardinality of a column,  $s_A$ . This is the number of records present with same column value as A. This is calculated as  $n_r/d_A$ . i.e.; total number of records with distinct value of A. For example, suppose EMP table has 500 records and DEPT\_ID has 5 distinct values. Then the selection cardinality of DEPT\_ID in EMP table is  $500/5 = 100$ . That means, on an average 100 employees are distributed among each department. This is helpful in determining average number of records that would satisfy selection criteria.
- There many other factors too like index type, data file type, sorting order, type of sorting etc.

### **Choice of Evaluation Plans in DBMS**

So far we saw how a query is parsed and traversed, how they are evaluated using different methods and what the different costs are when different methods are used. Now the important phase while evaluating a query is deciding which evaluation plan has to be selected so that it can be traversed efficiently. It collects all the statistics, costs, access/evaluation paths, relational trees etc. It then analyses them and chooses the best evaluation path.

Like we saw in the beginning of this article, same query is written in different forms of relational algebra. Corresponding trees for them too is drawn by DBMS. Statistics for them based on cost based evaluation and heuristic methods are collected. It checks the costs based on the different techniques that we have seen so far. It checks for the operator, joining type, indexes, number of records, selectivity of records, distinct values etc from the data dictionary. Once all these informations are collected, it picks the best evaluation plan.

Have look at below relational algebra and tree for EMP and DEPT.

```
[[ EMP_ID, DEPT_NAME ( $\sigma$  DEPT_ID = 10 AND EMP_LAST_NAME = 'Joseph'  
(EMP)  $\bowtie$  DEPT)
```

Or

```
[[ EMP_ID, DEPT_NAME ( $\sigma$  DEPT_ID = 10 AND EMP_LAST_NAME = 'Joseph'  
(EMP  $\bowtie$  DEPT))
```

Or

```
σ DEPT_ID = 10 AND EMP_LAST_NAME = 'Joseph' (⋈ EMP_ID, DEPT_NAME, DEPT_ID (EMP ∞ DEPT))
```

What can be observed here? First tree reduces the number of records for joining and seems to be efficient. But what happens if we have index on DEPT\_ID? Then the join between EMP and EMP can also be more efficient. But we see the filter condition on EMP table, we have DEPT\_ID = 10, which is index column. Hence first applying selection condition and then join will reduce the number of records as well as make the join more efficient than without index. Next are the projected columns – EMP\_ID and DEPT\_NAME. they are all distinct values. There cannot be duplicate values for them. But we are selecting those values for DEPT\_ID = 10, hence DEPT\_NAME has only one value. Hence their selectivity is same as number of employees working for DEPT\_ID = 10. But we are selecting only those employees whose last name is 'Joseph'. Hence the selectivity is min (distinct (employee (DEPT\_10)), distinct (employee (DEPT\_10, JOSEPH))). Obviously distinct (employee (DEPT\_10, JOSEPH)) would have lesser value. The optimizer decides all these factors for above 3 trees and then decides first tree would be more efficient. Hence it evaluates the query using first tree.

This is how when any query is submitted to DB is traversed and evaluated.

### Materialized view in DBMS

A materialized view is a view whose contents are computed and stored. Materialized view is also a logical virtual table, but in this case the result of the query is stored in the table or the disk. The performance of the materialized view is better than normal view since the data is stored in the disk.

It's also called indexed views since the table created after the query is indexed and can be accessed faster and efficiently.

### Example

Consider the view given below –

**Create view branchloan(branch-name, total-loan) as select branch-name , sum(amount) from loan groupby branch-name;**

Materializing the above view would be especially useful if the total loan amount is required frequently.

It saves the effort of finding multiple tuples and adding up their amounts.

The task of keeping a materialized view up-to-date with the underlying data is known as materialised view maintenance. It can be maintained by recompilation on every update.

A better option is to use incremental view maintenance. It changes to database relations are used to compute changes to materialized view, which is then updated.

**View maintenance can be done by following :**

Manually defining triggers on insert, delete, and update of each relation in the view definition.

Manually written code to update the view whenever database relations are updated.  
Supported directly by the database.

**Regular Expressions**

A Regular Expression is popularly known as RegEx, is a generalized expression that is used to match patterns with various sequences of characters. A RegEx can be a combination of different data types such as integer, special characters, Strings, images, etc.

In SQL if you were looking for email addresses from the same company Regex lets you define a pattern using comparators and [Metacharacters](#), in this case using ~\* and % to help define the pattern:

```
SELECT * FROM Email Addresses
WHERE Email Address ~* '%@chartio.com'
```

**Metacharacters**

Here is a quick cheat sheet for metacharacters to help define the pattern:

METACHARACTER	DESCRIPTION	EXAMPLE	EXAMPLES MATCHES
^	Start the match at the beginning of a string	^c%	cat, car, chain
	Alternation (either of two alternatives)	c(a o)%	can, corn, cop
()	Group items in a single logical item	c(a o)%	can, corn, cop
_	Any single character (using LIKE and SIMILAR TO)	c_	co, fico, pico
%	Any string (using LIKE and SIMILAR TO)	c%	chart, articulation, cra
.	Any single character (using POSIX)	c.	co, fico, pico
.*	Any string (using POSIX)	c.*	chart, articulation, crate
+	Repetition of the previous item one or more times	co+	coo, cool



