DBMS

Database Management System

What is DBMS

- Data : collection of facts
- Example : I got 60 mks in maths is a fact
- My salary is Rs 90,000 is a fact
- **What is a Database?**
- A database is a collection of related data which represents some aspect of the real world.
- A database system is designed to be built and populated with data for a certain task.

What is DBMS?

- **Database Management System (DBMS)** is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data.

- DBMS allows users to create their own databases as per their requirement. The term “DBMS” includes the user of the database and other application programs. It provides an interface between the data and the software application.

Example of Databases

- University database
- Train database
- Flight database
- Aadhar database
- Passport database
- Hospital database
- Books database
- Banking
- Sales/HR/Manufacturing/Telecommunication

Popular DBMS Software

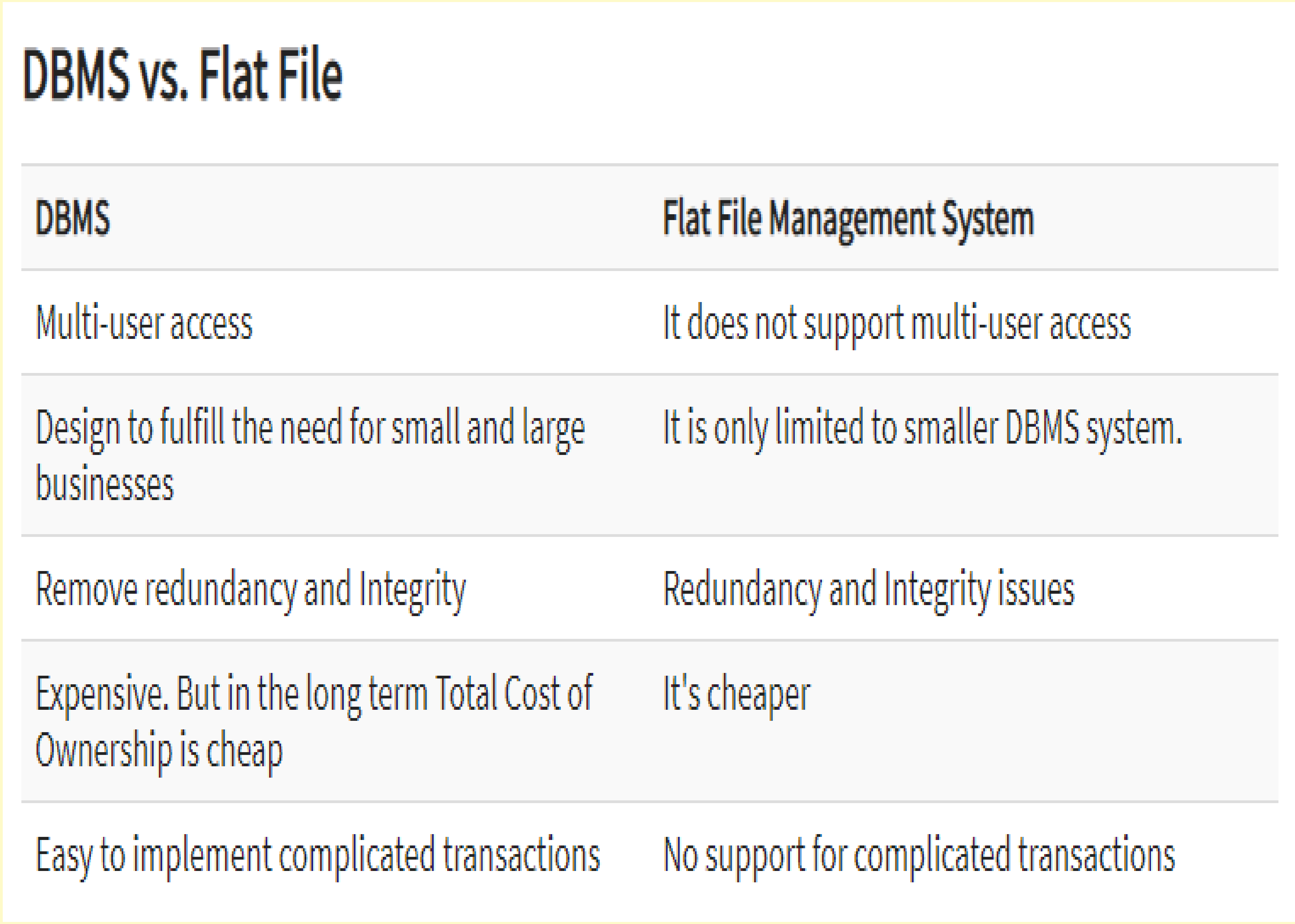
- Here, is the list of some popular DBMS system:
- MySQL
- Microsoft Access
- Oracle
- PostgreSQL
- dBASE
- FoxPro
- SQLite
- IBM DB2
- LibreOffice Base
- MariaDB
- Microsoft SQL Server etc.

History of DBMS

- Here, are the important landmarks from the history:
- 1960 - Charles Bachman designed first DBMS system
- 1970 – E.F Codd introduced IBM'S Information Management System (IMS)
- 1976- Peter Chen coined and defined the Entity-relationship model also know as the ER model
- 1980 - Relational Model becomes a widely accepted database component
- 1985- Object-oriented DBMS develops.
- 1990s- Incorporation of object-orientation in relational DBMS.
- 1995: First Internet database applications
- 1997: XML applied to database processing. Many vendors begin to integrate XML into DBMS products.

Characteristics of Database Management System

- Provides security and removes redundancy
- Self-describing nature of a database system
- Insulation between programs and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing
- DBMS allows entities and relations among them to form tables.
- It follows the ACID concept (Atomicity, Consistency, Isolation, and Durability).
- DBMS supports multi-user environment that allows users to access and manipulate data in parallel.

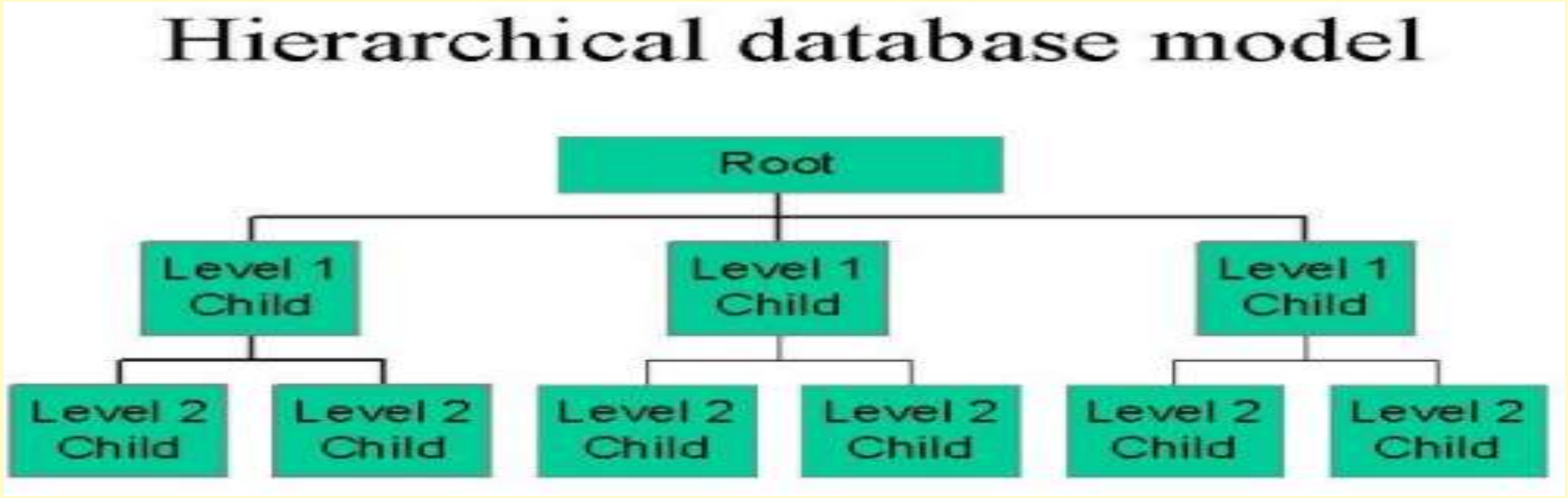


DBMS vs. Flat File

DBMS	Flat File Management System
Multi-user access	It does not support multi-user access
Design to fulfill the need for small and large businesses	It is only limited to smaller DBMS system.
Remove redundancy and Integrity	Redundancy and Integrity issues
Expensive. But in the long term Total Cost of Ownership is cheap	It's cheaper
Easy to implement complicated transactions	No support for complicated transactions

Types of DBMS

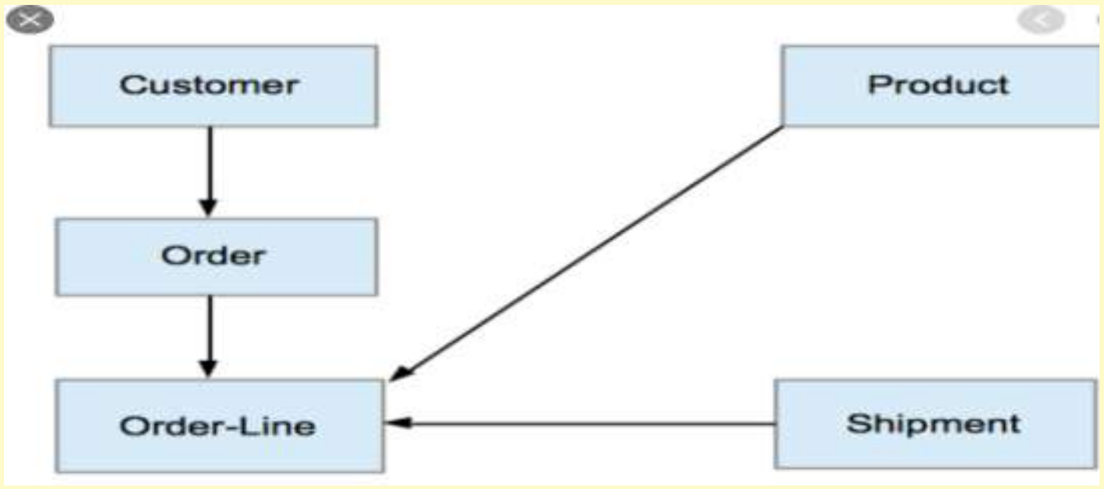
- Types of DBMS
- Four Types of DBMS systems are:
- Hierarchical database
- Network database
- Relational database
- Object-Oriented database



- In a Hierarchical database, model data is organized in a tree-like structure. Data is Stored Hierarchically (top down or bottom up) format. Data is represented using a parent-child relationship. In Hierarchical DBMS parent may have many children, but children have only one parent.

- **Advantages**
- The design of the hierarchical model is simple.
- Provides Data Integrity since it is based on parent/ child relationship
- Data sharing is feasible since the data is stored in a single database.
- Even for large volumes of data, this model works perfectly.
- **Disadvantages**
- Implementation is complex.
- This model has to deal with anomalies like Insert, Update and Delete.
- Maintenance is difficult since changes done in the database may want you to do changes in the entire database structure.

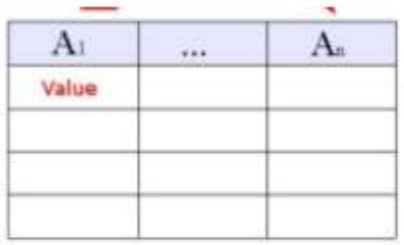
Network Model



- The network database model allows each child to have multiple parents. It helps you to address the need to model more complex relationships like as the orders/parts many-to-many relationship. In this model, entities are organized in a graph which can be accessed through several paths.

- **Advantages**
- Easy to design the Network Model
- The model can handle one-one, one-to-many, many-to-many relationships.
- It isolates the program from other details.
- Based on standards and conventions.
- **Disadvantages**
- Pointers bring complexity since the records are based on pointers and graphs.
- Changes in the database isn't easy that makes it hard to achieve structural independence.

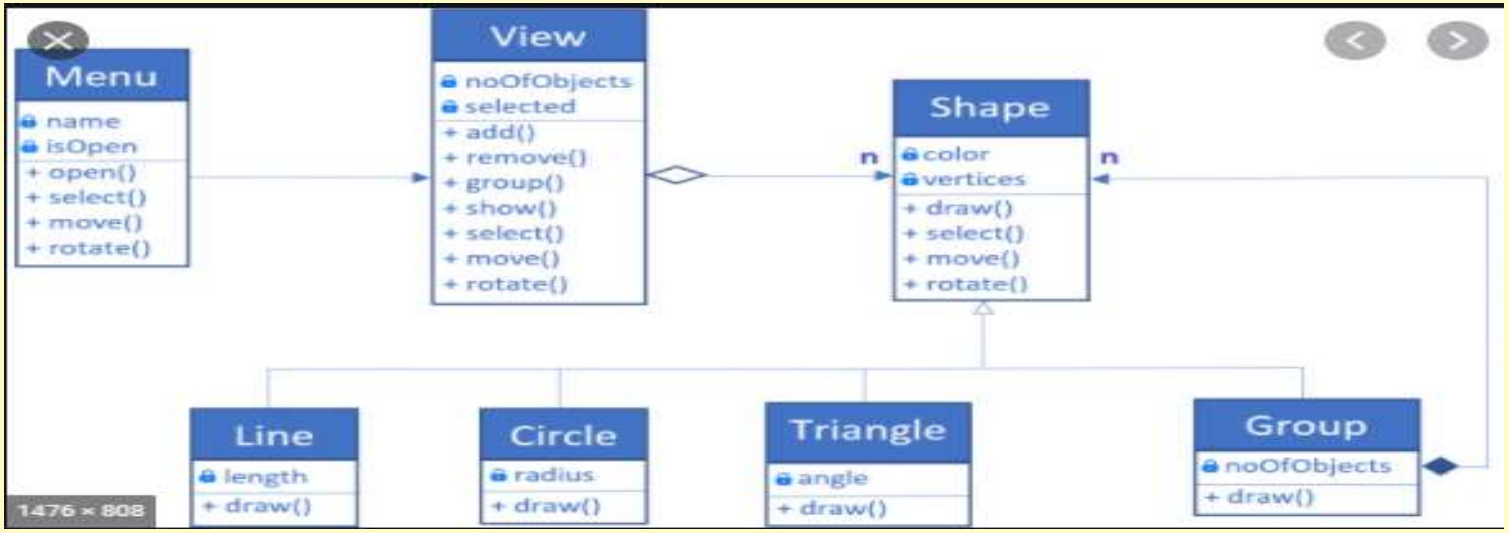
The Relational Model



- Relational DBMS is the most widely used DBMS model because it is one of the easiest. This model is based on normalizing data in the rows and columns of the tables. Relational model stored in fixed structures and manipulated using SQL.

- **Advantages**
- The Relational Model does not have any issues that we saw in the previous two models i.e. update, insert and delete anomalies have nothing to do in this model.
- Changes in the database do not require you to affect the complete database.
- Implementation of a Relational Model is easy.
- To maintain a Relational Model is not a tiresome task.
- **Disadvantages**
- Database inefficiencies hide and arise when the model has large volumes of data.
- The overheads of using relational data model come with the cost of using powerful hardware and devices.
-

Object-Oriented Model



- In Object-oriented Model data stored in the form of objects. The structure which is called classes which display data within it. It defines a database as a collection of objects which stores both data members values and operations.

- **Advantages**
- Complex data sets can be saved and retrieved quickly and easily.
- Object databases are not widely adopted.Object IDs are assigned automatically.
- Works well with object-oriented programming languages.

- **Disadvantages**
- Object databases are not widely adopted.
- In some situations, the high complexity can cause performance problems.

Advantages of DBMS

- DBMS offers a variety of techniques to store & retrieve data
- DBMS serves as an efficient handler to balance the needs of multiple applications using the same data
- Uniform administration procedures for data
- Application programmers never exposed to details of data representation and storage.
- A DBMS uses various powerful functions to store and retrieve data efficiently.
- Offers Data Integrity and Security
- The DBMS implies integrity constraints to get a high level of protection against prohibited access to data.
- A DBMS schedules concurrent access to the data in such a manner that only one user can access the same data at a time
- Reduced Application Development Time

Disadvantage of DBMS

- DBMS may offer plenty of advantages but, it has certain flaws-
- Cost of Hardware and Software of a DBMS is quite high which increases the budget of your organization.
- Most database management systems are often complex systems, so the training for users to use the DBMS is required.
- In some organizations, all data is integrated into a single database which can be damaged because of electric failure or database is corrupted on the storage media
- Use of the same program at a time by many users sometimes lead to the loss of some data.
- DBMS can't perform sophisticated calculations

Data abstraction:

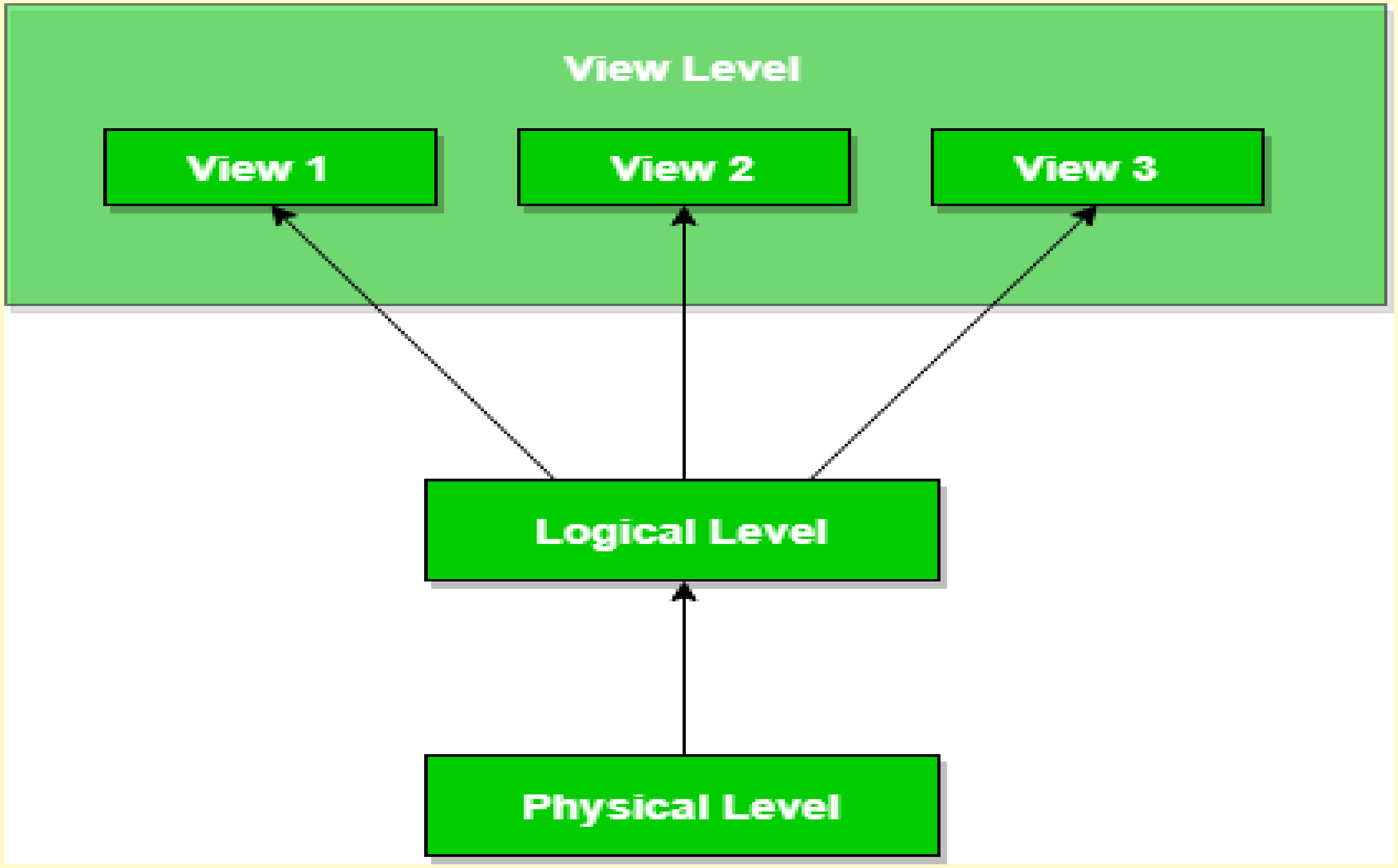
There are mainly **3** levels of data abstraction:
Physical: This is the lowest level of data abstraction. It tells us how the data is actually stored in memory. The access methods like sequential or random access and file organization methods like B+ trees, hashing used for the same. Usability, size of memory, and the number of times the records are factors which we need to know while designing the database. Suppose we need to store the details of an employee. Blocks of storage and the amount of memory used for these purposes is kept hidden from the user.

Logical: This level comprises of the information that is actually stored in the database in the form of tables. It also stores the relationship among the data entities in relatively simple structures. At this level, the information available to the user at the view level is unknown.

We can store the various attributes of an employee and relationships, e.g. with the manager can also be stored.

View: This is the highest level of abstraction. Only a part of the actual database is viewed by the users. This level exists to ease the accessibility of the database by an individual user.

Users view data in the form of rows and columns. Tables and relations are used to store data. Multiple views of the same database may exist. Users can just view the data and interact with the database, storage and implementation details are hidden from them.


View Level
View 1
View 2
View 3
Logical Level
Physical Level

Data Independence

The main purpose of data abstraction is achieving data independence in order to save time and cost required when the database is modified or altered.
We have namely two levels of data independence arising from these levels of abstraction :

Physical level data independence : It refers to the characteristic of being able to modify the physical schema without any alterations to the conceptual or logical schema, done for optimization purposes, e.g., Conceptual structure of the database would not be affected by any change in storage size of the database system server. Changing from sequential to random access files is one such example.

These alterations or modifications to the physical structure may include:
•Utilizing new storage devices.
•Modifying data structures used for storage.
•Altering indexes or using alternative file organization techniques etc.

- **Logical level data independence:** It refers characteristic of being able to modify the logical schema without affecting the external schema or application program. The user view of the data would not be affected by any changes to the conceptual view of the data. These changes may include insertion or deletion of attributes, altering table structures entities or relationships to the logical schema etc.

Different types of Database Users

- These are seven types of data base users in DBMS.
- **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.
-

- **Naïve(inexperience) / 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.
-

- **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.

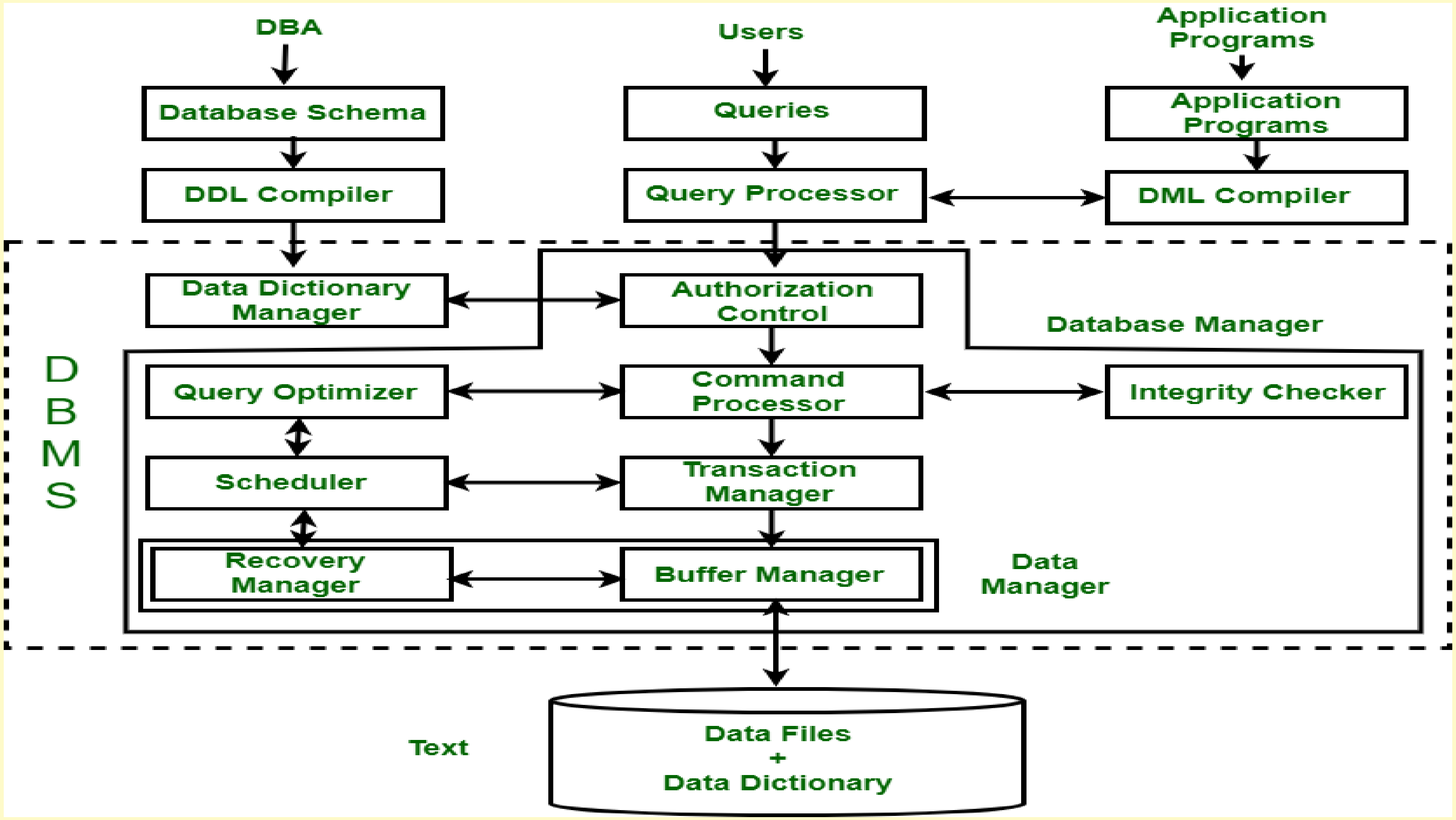
- **Sophisticated Users :**
  Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database. They can develop their own data base applications according to their requirement. They don’t write the program code but they interact the data base by writing SQL queries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

- **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.

- **Application Programmer :**
  Application Programmer 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.

- **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.

Structure--Data Base Management



- **1. Query Processor :**
  It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler. Query Processor contains the following components –

- **DML Compiler –**
  It processes the DML statements into low level instruction (machine language), so that they can be executed.

- **DDL Interpreter –**
  It processes the DDL statements into a set of table containing meta data (data about data).

- **Embedded DML Pre-compiler –**
  It processes DML statements embedded in an application program into procedural calls.

- **Query Optimizer –**
  It executes the instruction generated by DML Compiler.

- **2. Storage Manager :**
  Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.
  It contains the following components –

- **Authorization Manager –**
  It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

- **Integrity Manager –**
  It checks the integrity constraints when the database is modified.

-

- **Transaction Manager –**
  It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.

- **File Manager –**
  It manages the file space and the data structure used to represent information in the database.

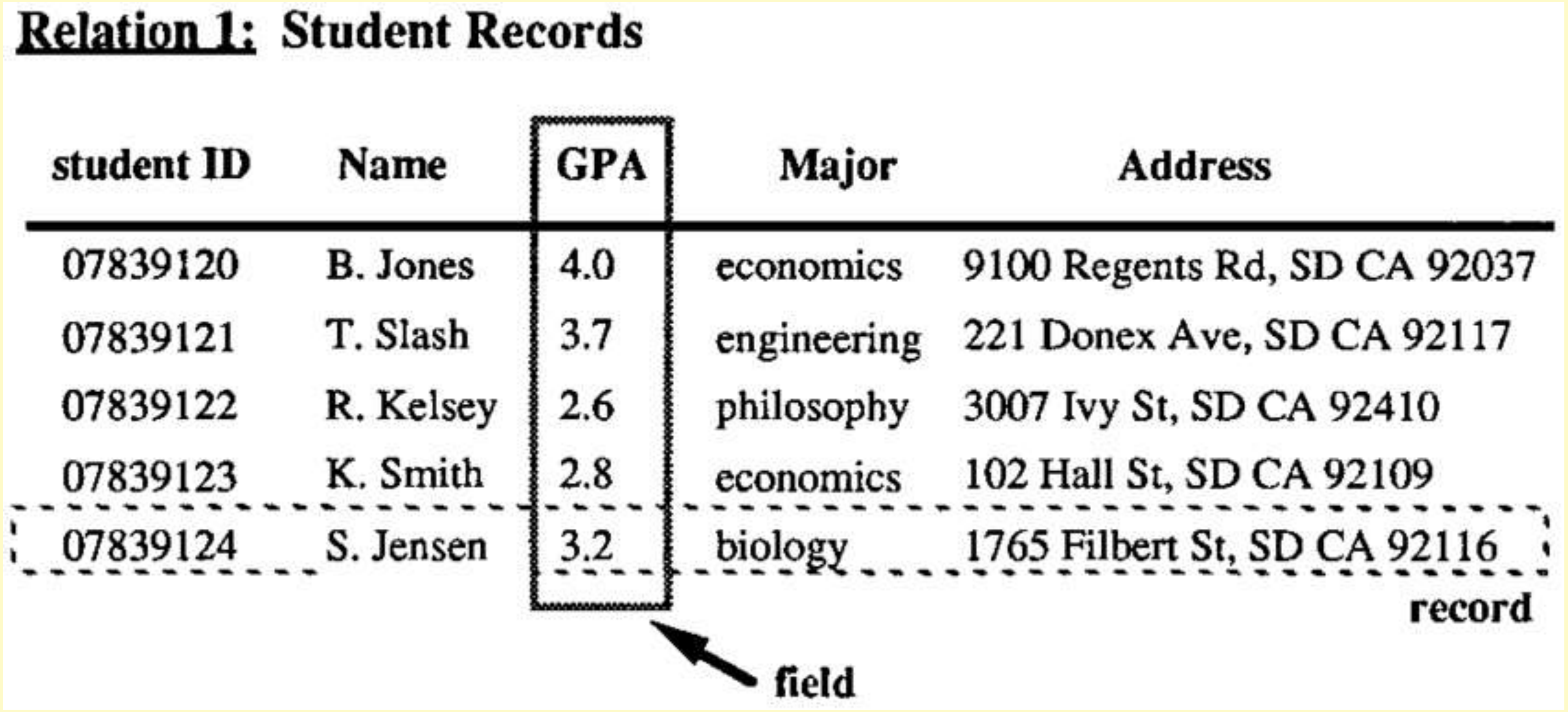
- **Buffer Manager –**
  It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

- **3. Disk Storage :**
  It contains the following components –

- **Data Files –**
  It stores the data.

- **Data Dictionary –**
  It contains the information about the structure of any database object. It is the repository of information that governs the metadata.

Database



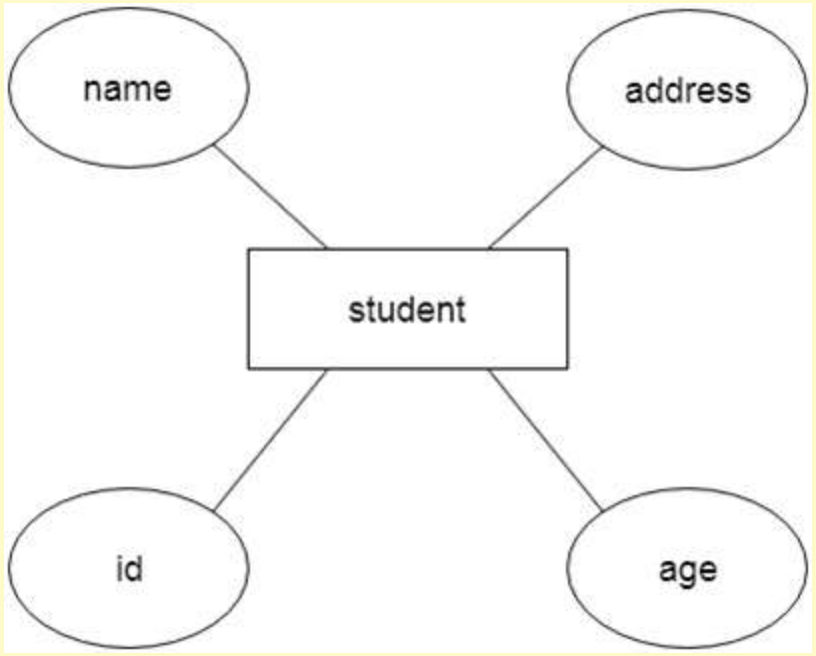
Relation 1: **Student Records**

student ID	Name	GPA	Major	Address
07839120	B. Jones	4.0	economics	9100 Regents Rd, SD CA 92037
07839121	T. Slash	3.7	engineering	221 Donex Ave, SD CA 92117
07839122	R. Kelsey	2.6	philosophy	3007 Ivy St, SD CA 92410
07839123	K. Smith	2.8	economics	102 Hall St, SD CA 92109
07839124	S. Jensen	3.2	biology	1765 Filbert St, SD CA 92116

record
field

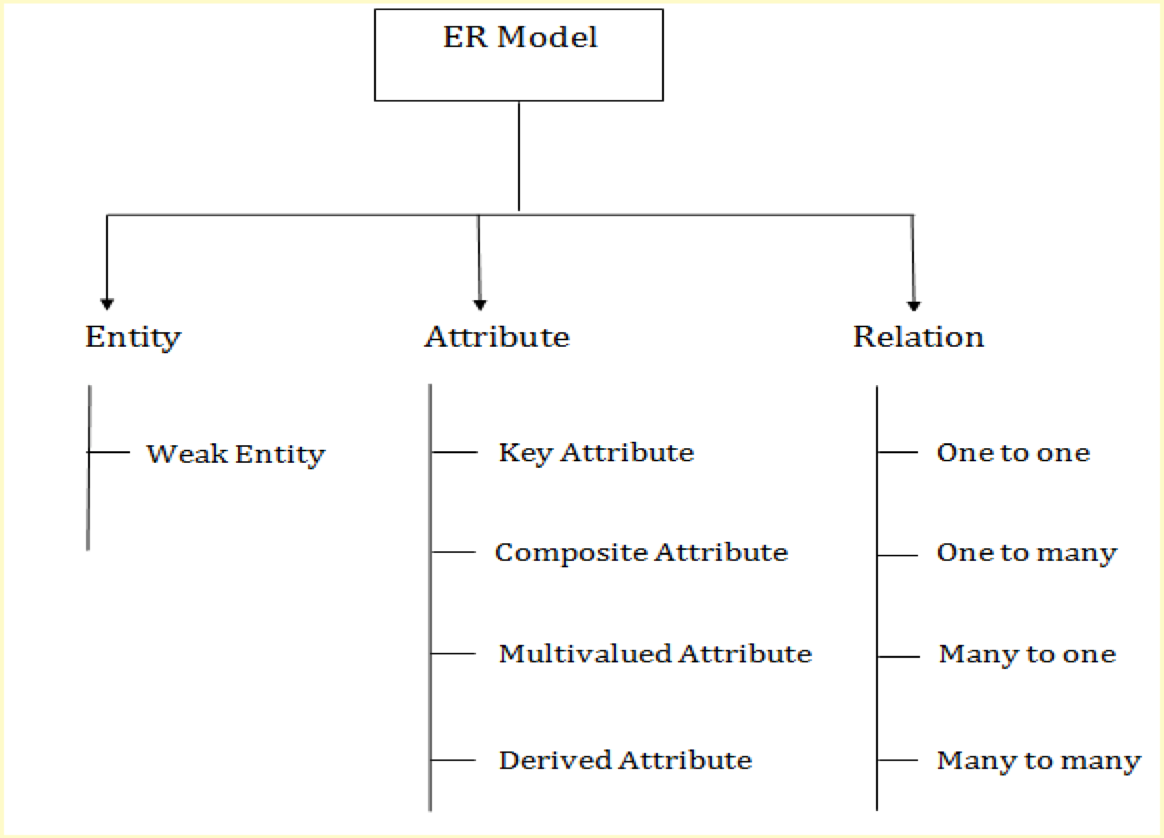
ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.
- **For example,** Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.
-

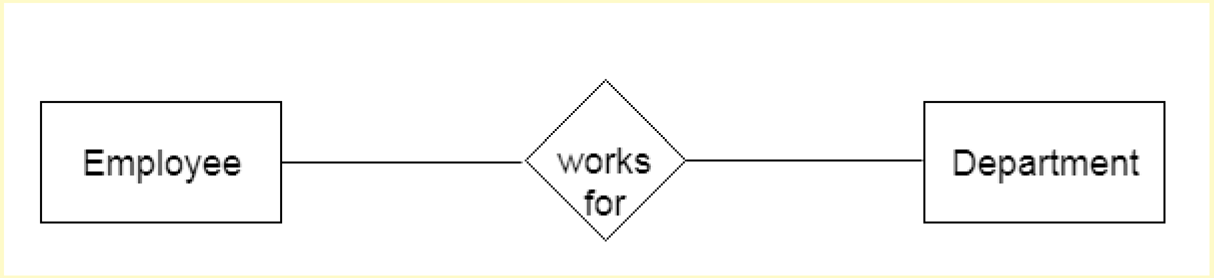
- 
name
address
student
id
age

Components of ER Model

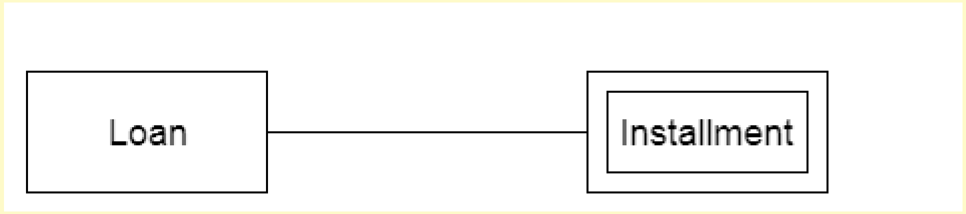
-

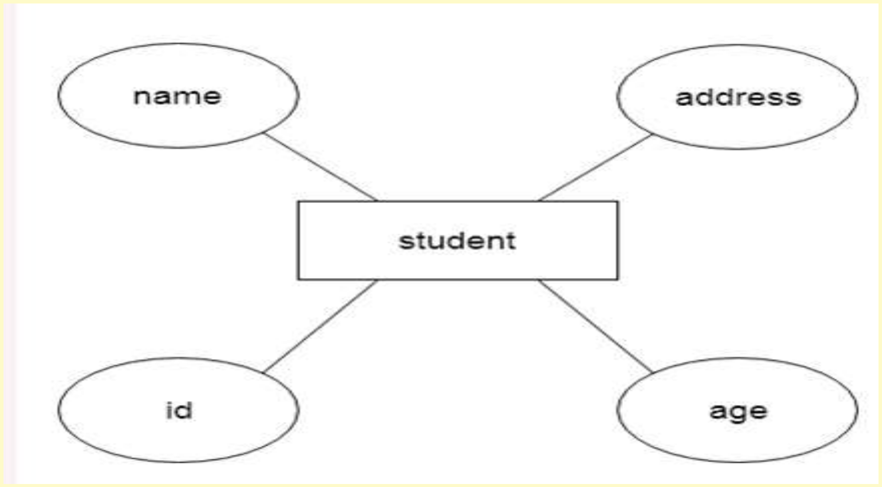


- 1. Entity:
- An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
- Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



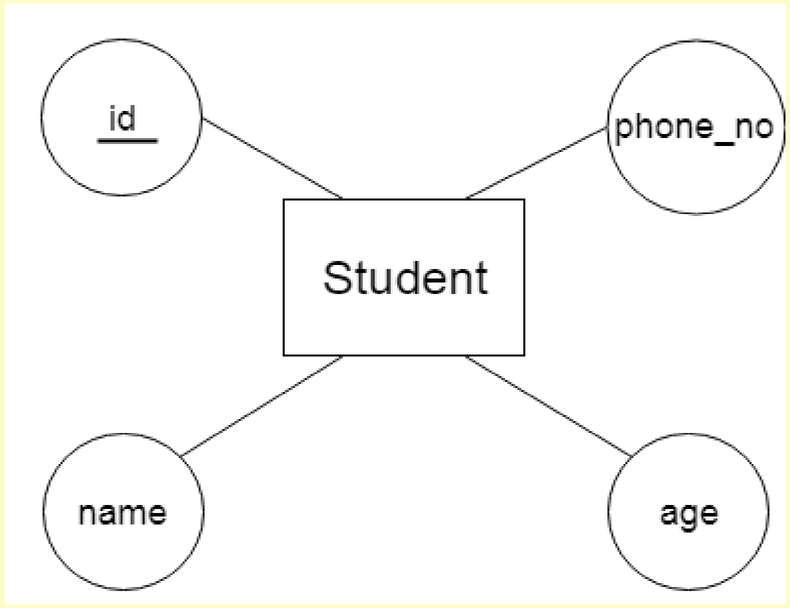
-

  a. Weak Entity
- An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.
- 

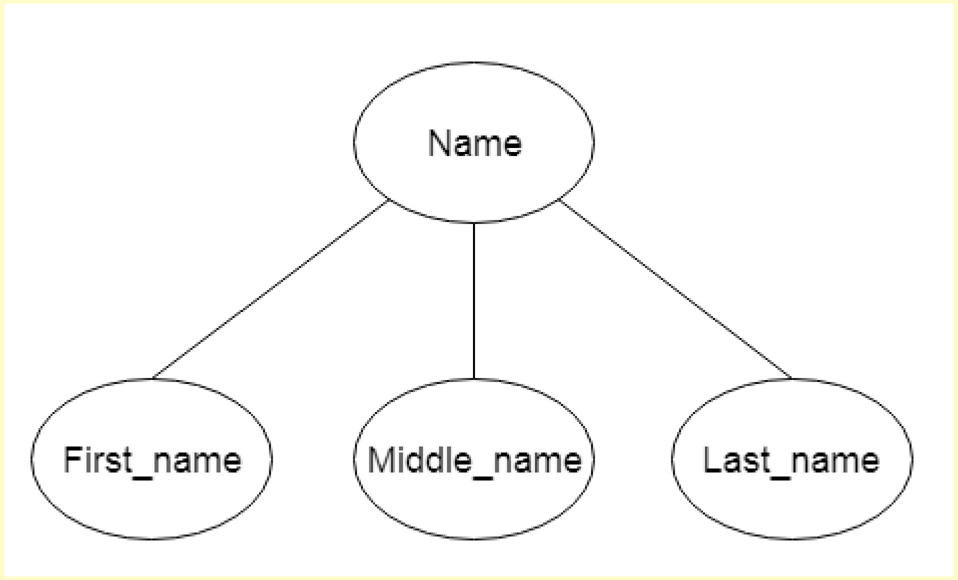
- 

  2. Attribute
- The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
- **For example,** id, age, contact number, name, etc. can be attributes of a student.

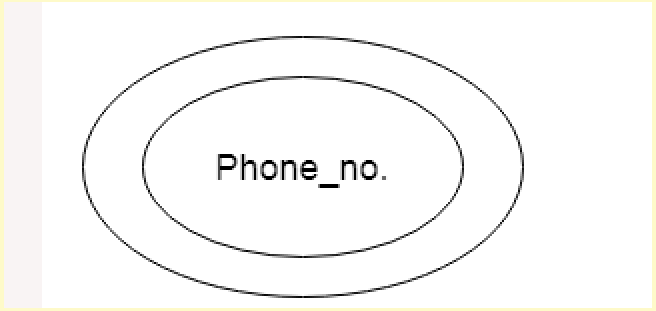
- **a. Key Attribute**
- The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.
-



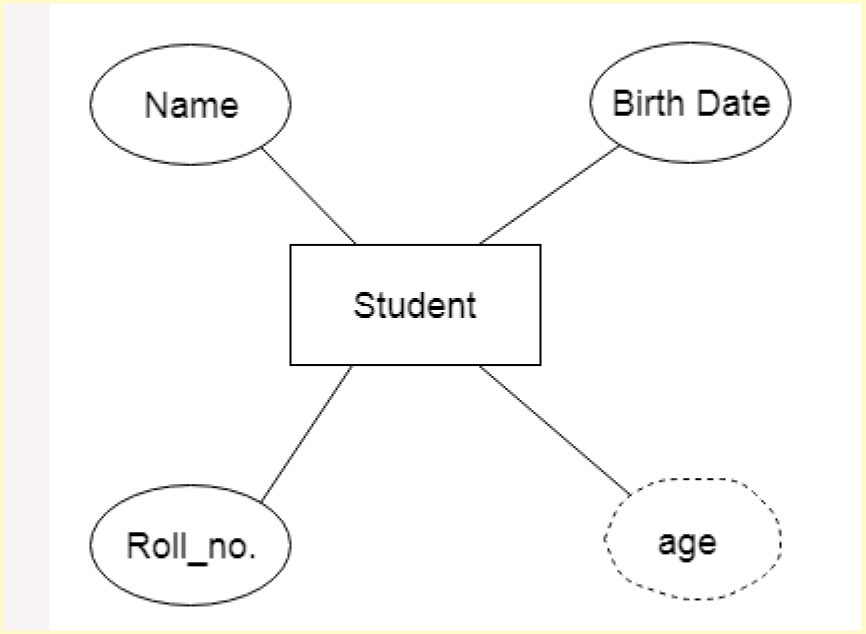
- **b. Composite Attribute**
- An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.
-

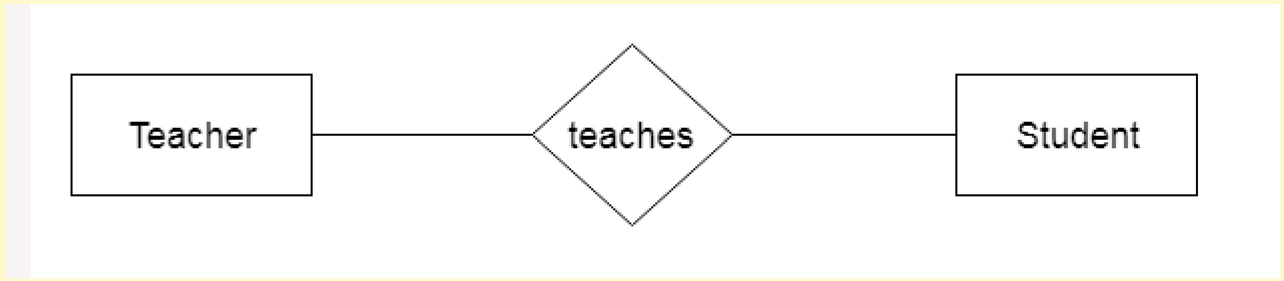


- **c. Multivalued Attribute**
- An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.
- **For example,** a student can have more than one phone number.

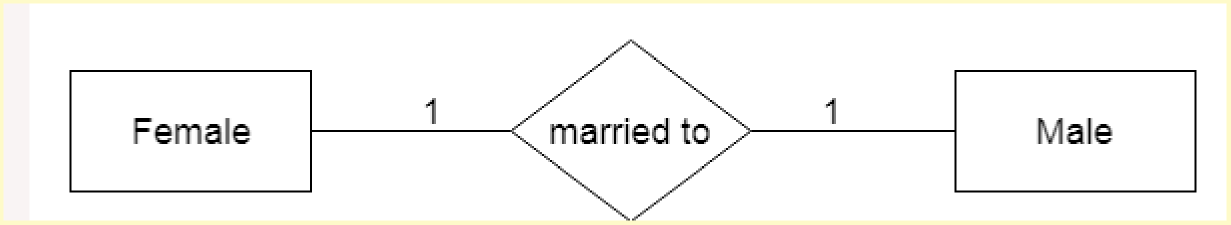


- **d. Derived Attribute**
- An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.
- **For example,** A person's age changes over time and can be derived from another attribute like Date of birth.
-

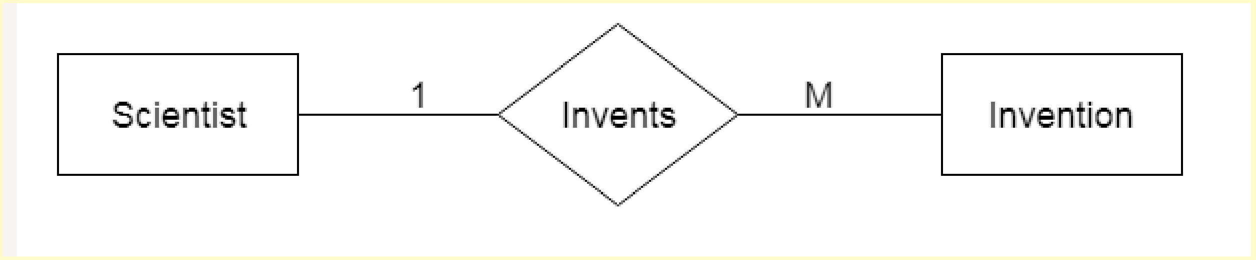


- 3. Relationship
- A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.
- 

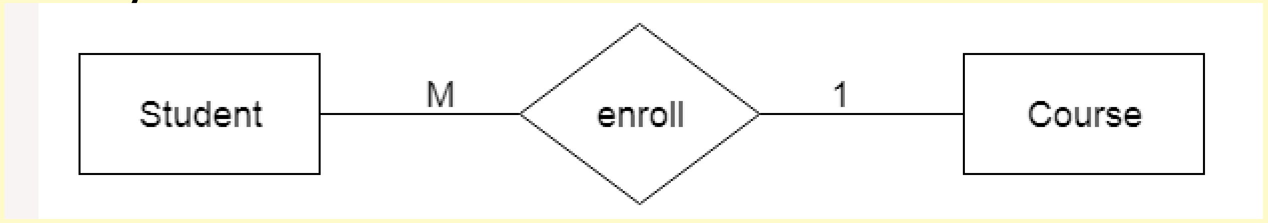
- Types of relationship are as follows:
- **a. One-to-One Relationship**
- When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.
- **For example,** A female can marry to one male, and a male can marry to one female.
-



- **b. One-to-many relationship**
- When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.
- **For example,** Scientist can invent many inventions, but the invention is done by the only specific scientist.

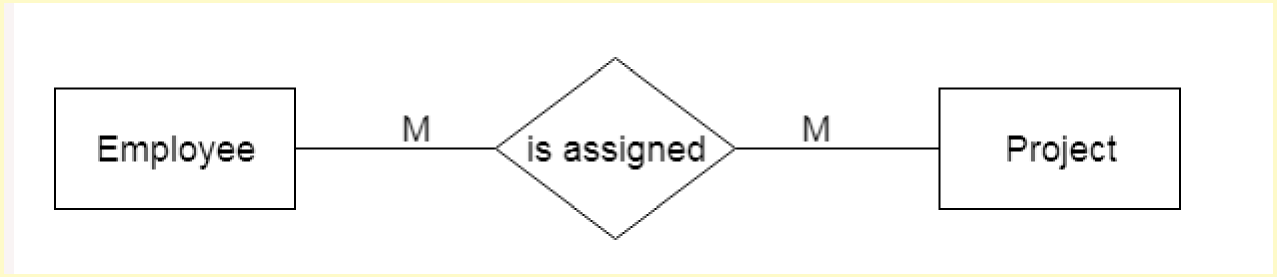


Women (one) marrying (many)men---**Polyandry**

- **c. Many-to-one relationship**
- When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.
- **For example,** Student enrolls for only one course, but a course can have many students.
- 

Many women **marrying one man---Polygyny**

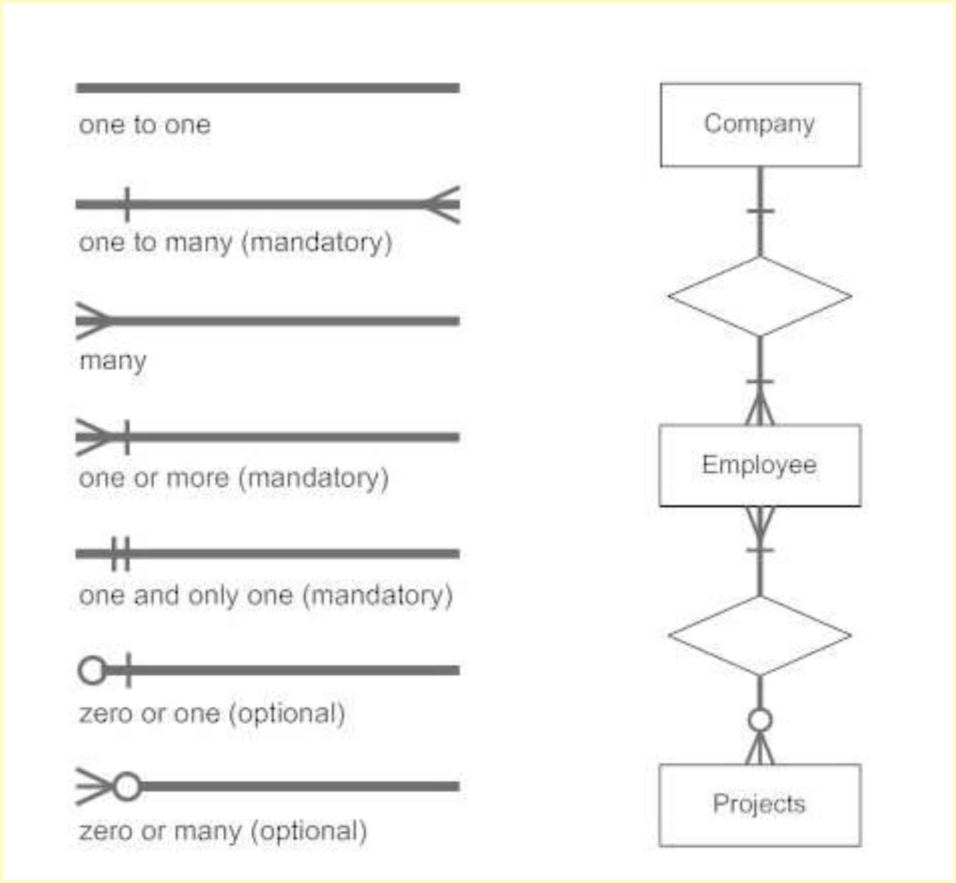
- **d. Many-to-many relationship**
- When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.
- **For example,** Employee can assign by many projects and project can have many employees.



Notation of ER diagram

- Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:
-

Fig: Notations of ER diagram



Mapping Constraints

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
  - One to one (1:1)
  - One to many (1:M)
  - Many to one (M:1)
  - Many to many (M:M)
- One-to-one
- In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.

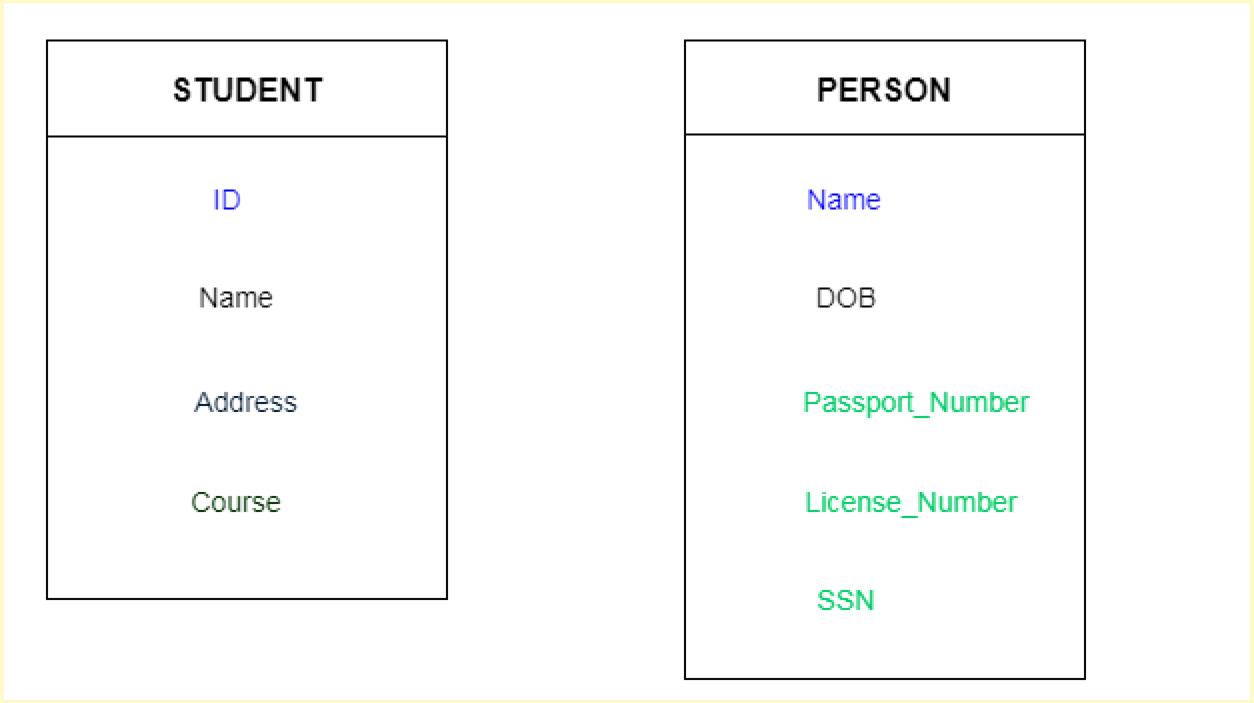
- One-to-many
- In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.

- Many-to-one
- In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.

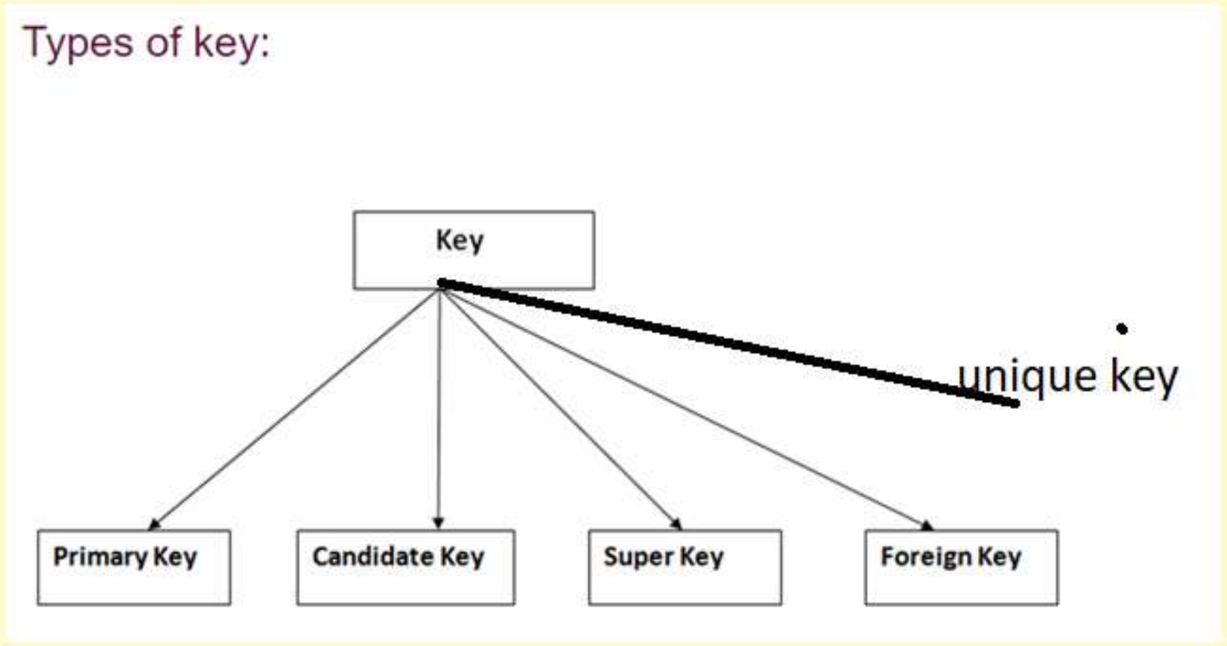
- Many-to-many
- In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.

- Keys
- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.
- **For example:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.
-

Keys

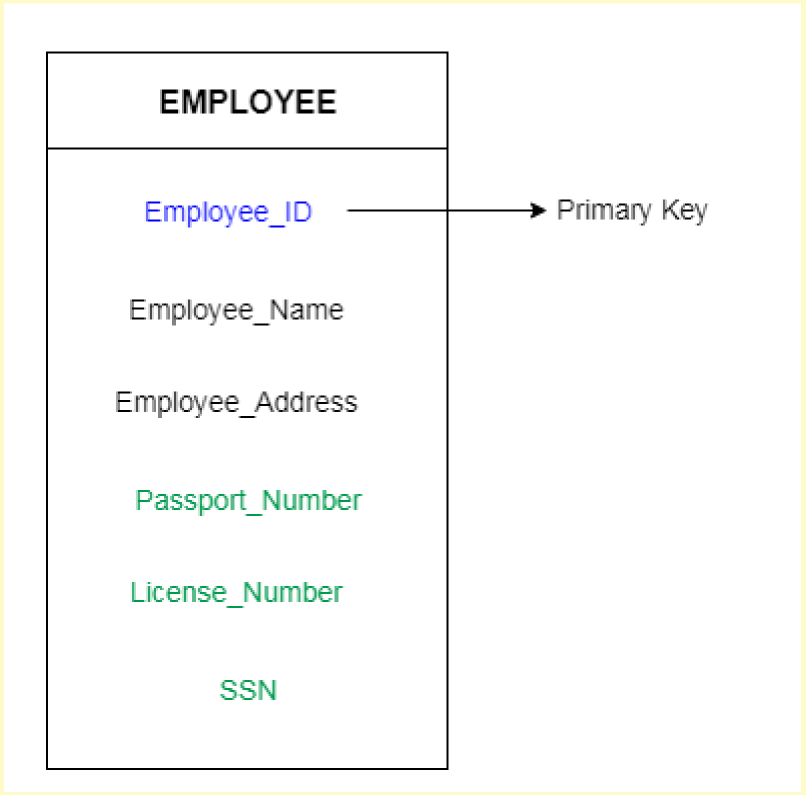

STUDENT
ID
Name
Address
Course
PERSON
Name
DOB
Passport_Number
License_Number
SSN

Types of key:


Types of key:
Key
unique key
Primary Key
Candidate Key
Super Key
Foreign Key

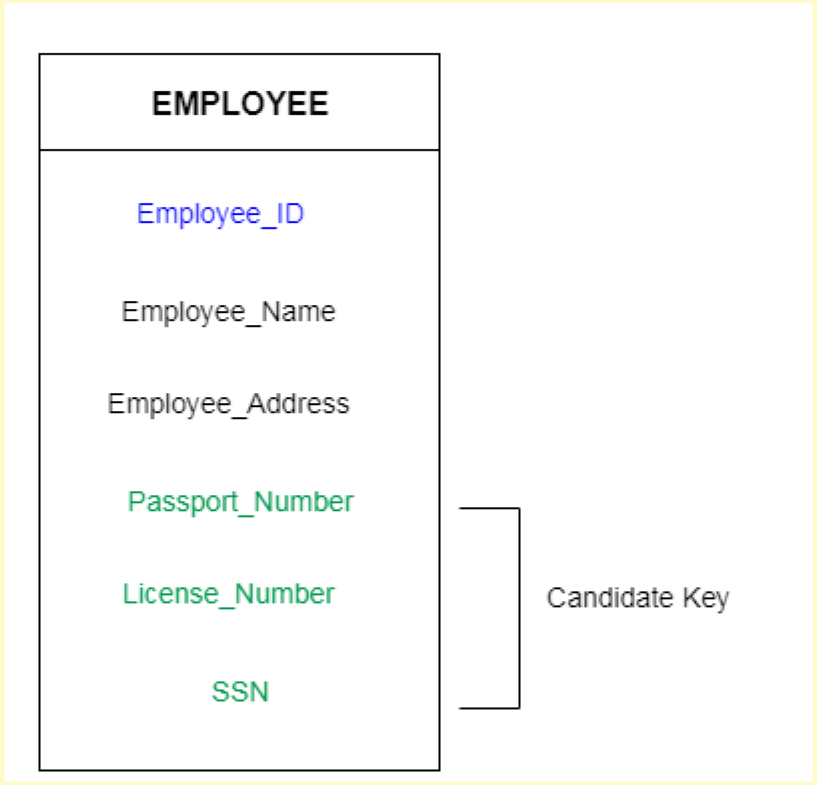
1. Primary key

- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.
-


EMPLOYEE
Employee_ID
Primary Key
Employee_Name
Employee_Address
Passport_Number
License_Number
SSN

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.
- **For example:** In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

2. Candidate key



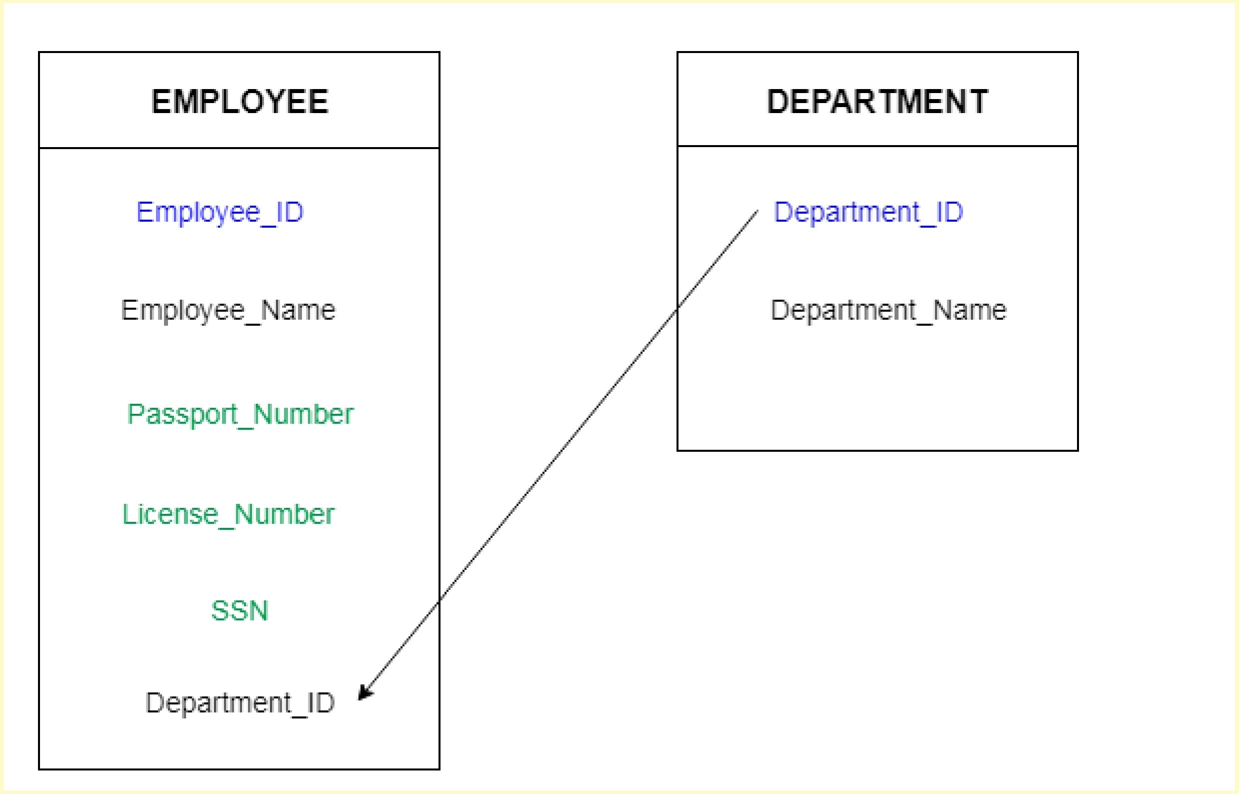
3. Super Key

- Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.
- **For example:** In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.
- The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

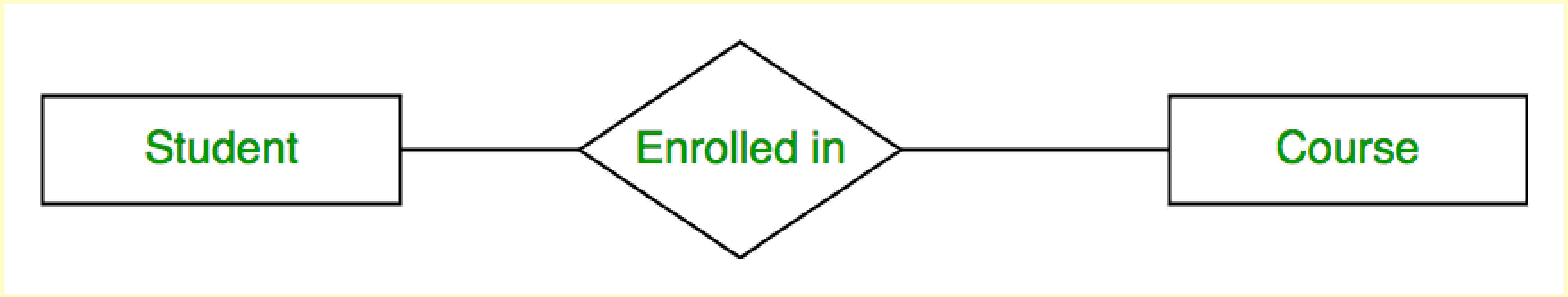
4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

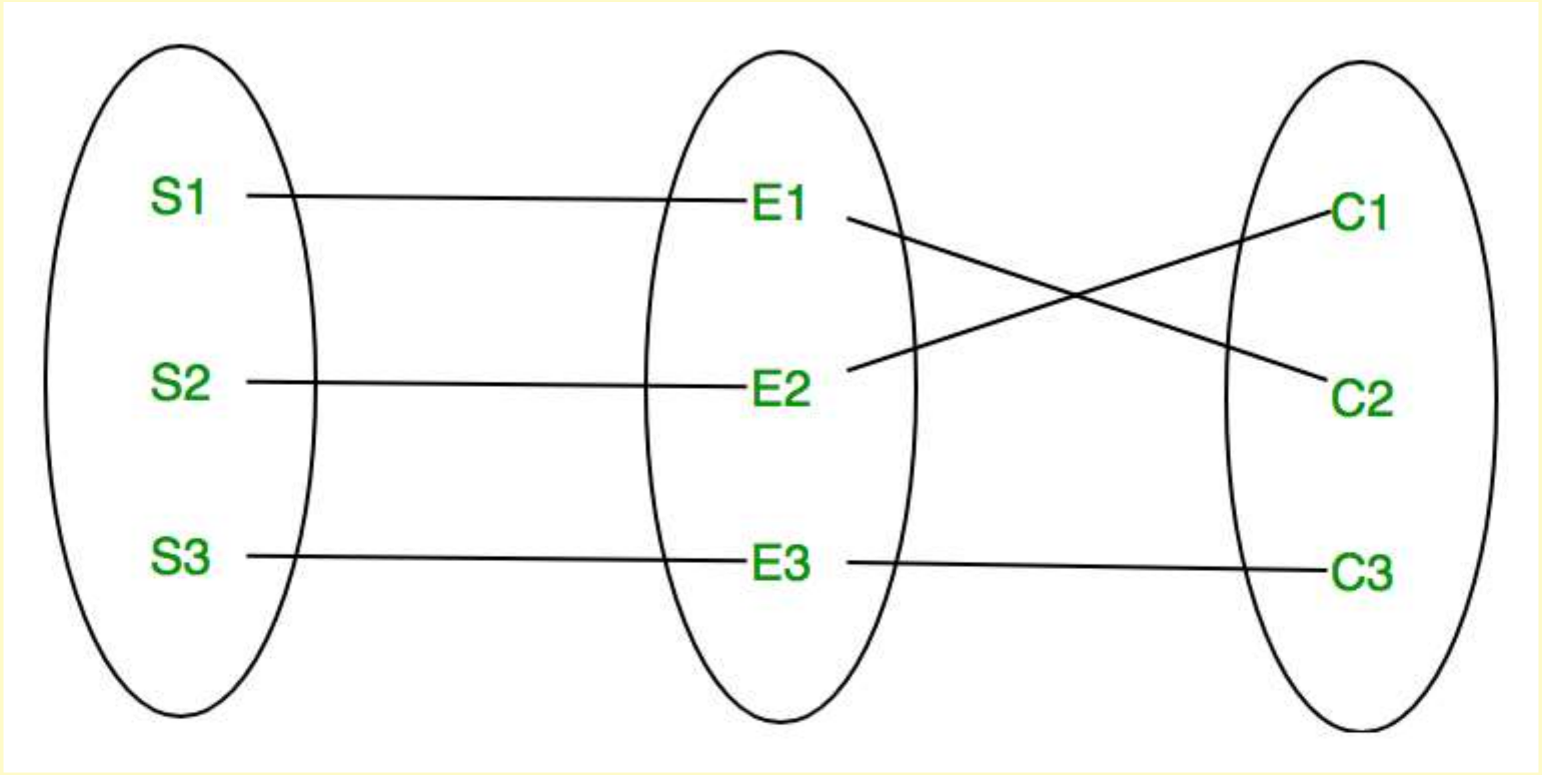
Foreign key


EMPLOYEE
Employee_ID
Employee_Name
Passport_Number
License_Number
SSN
Department_ID
DEPARTMENT
Department_ID
Department_Name

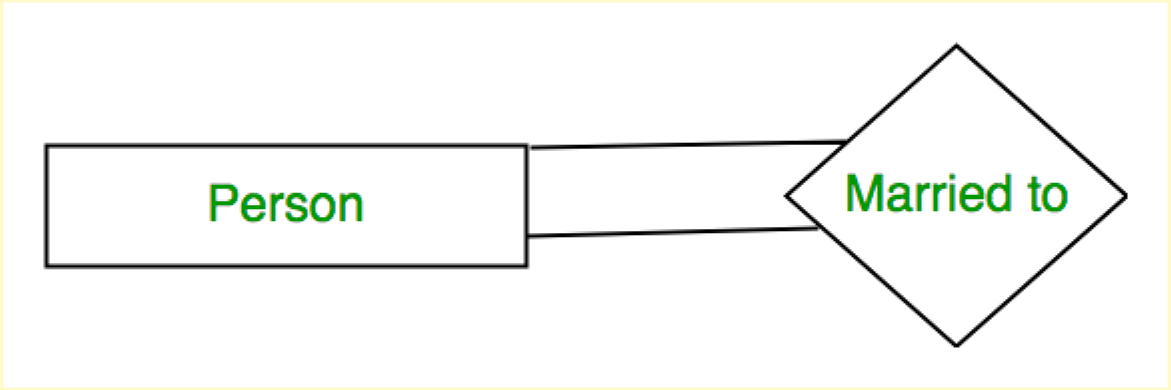
- **Relationship Type and Relationship Set:**
  A relationship type represents the **association between entity types**. For example,‘Enrolled in’ is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.


Student
Enrolled in
Course

- A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

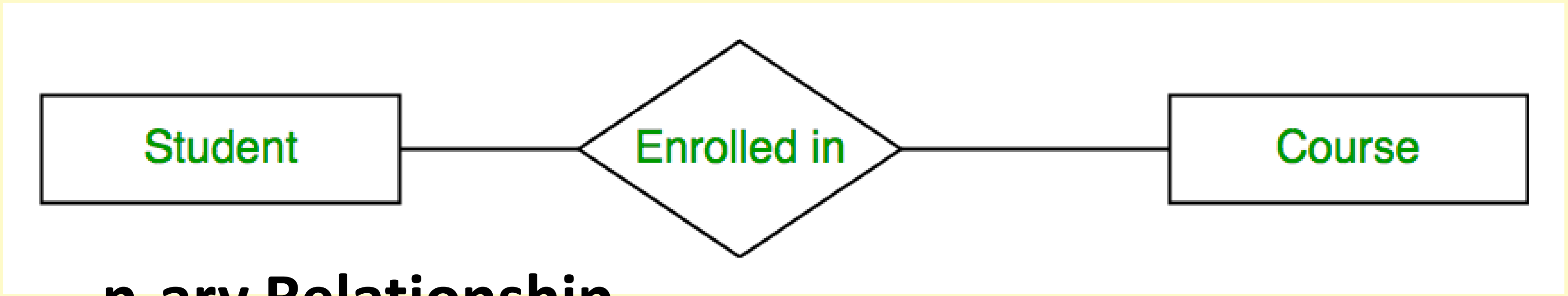


- **Degree of a relationship set:**
  The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.
- **Unary Relationship –**
  When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only one person.


Person
Married to

Binary’,Relationship

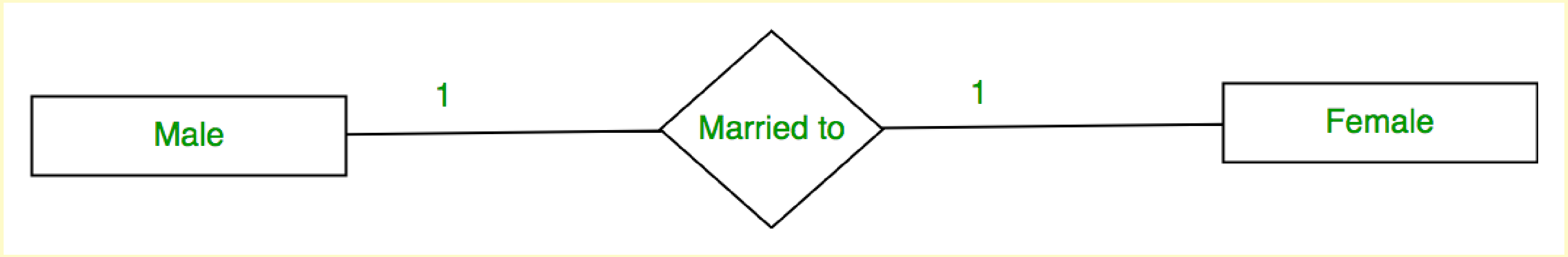
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.



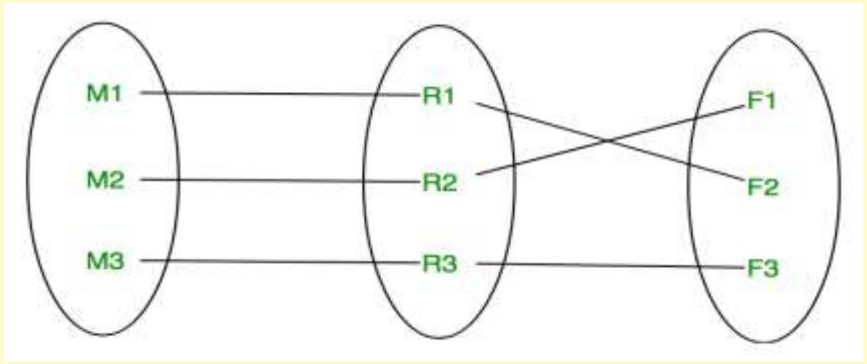
n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

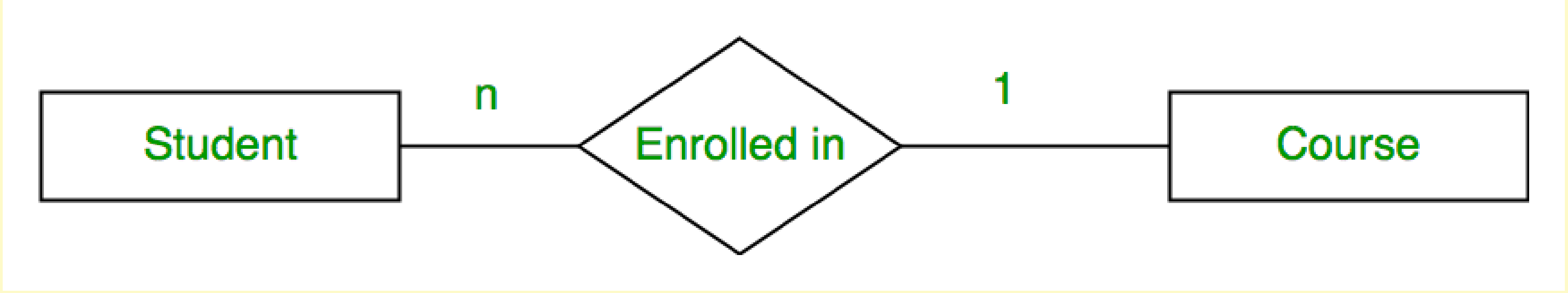
- **Cardinality:**
  The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:
- **One to one –** When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



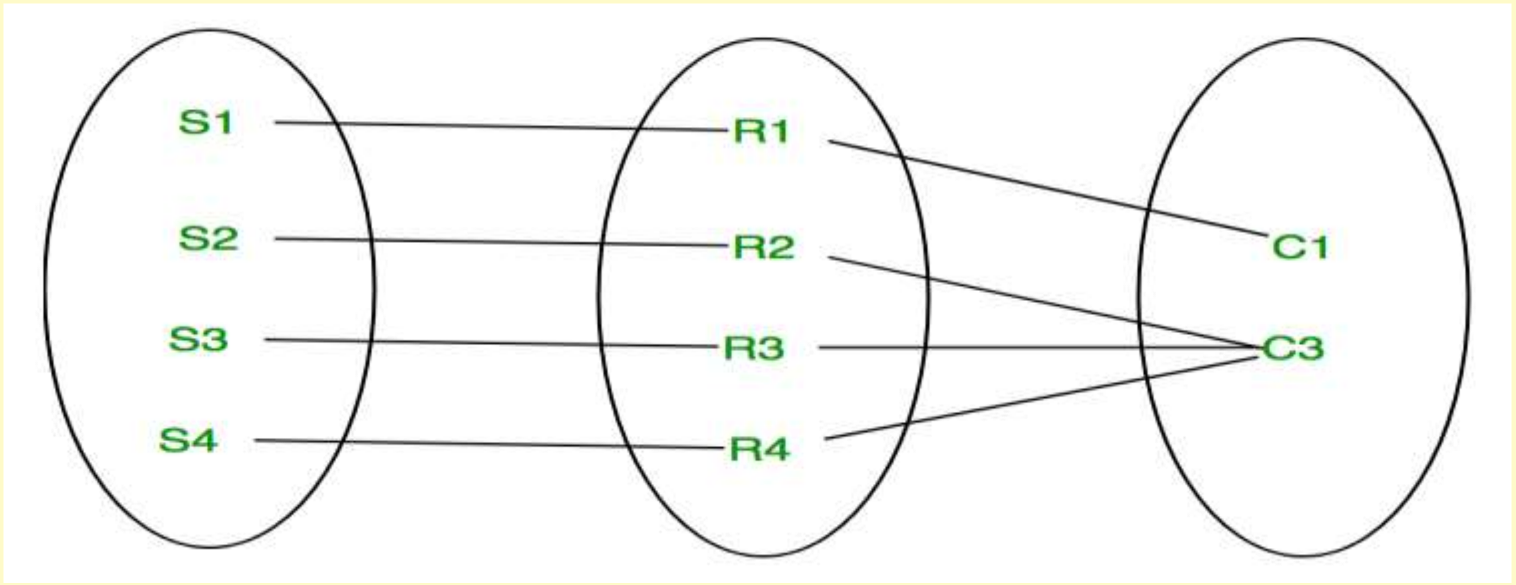
Using Sets, it can be represented as:



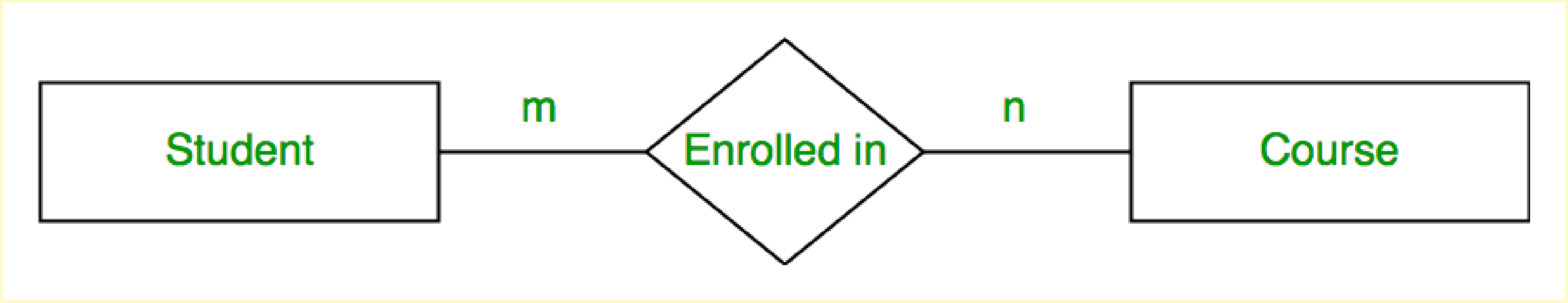
- **Many to one –** When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.
-



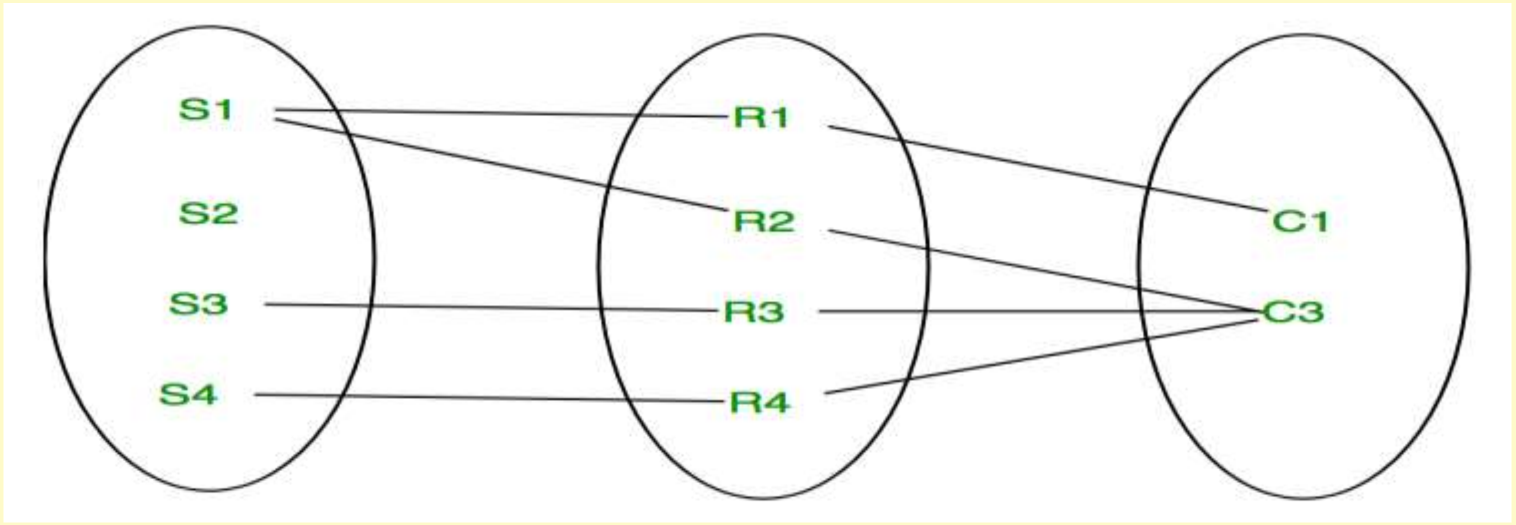
Using Sets, it can be represented as:



- **Many to many –** When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



Using Sets, it can be represented as:



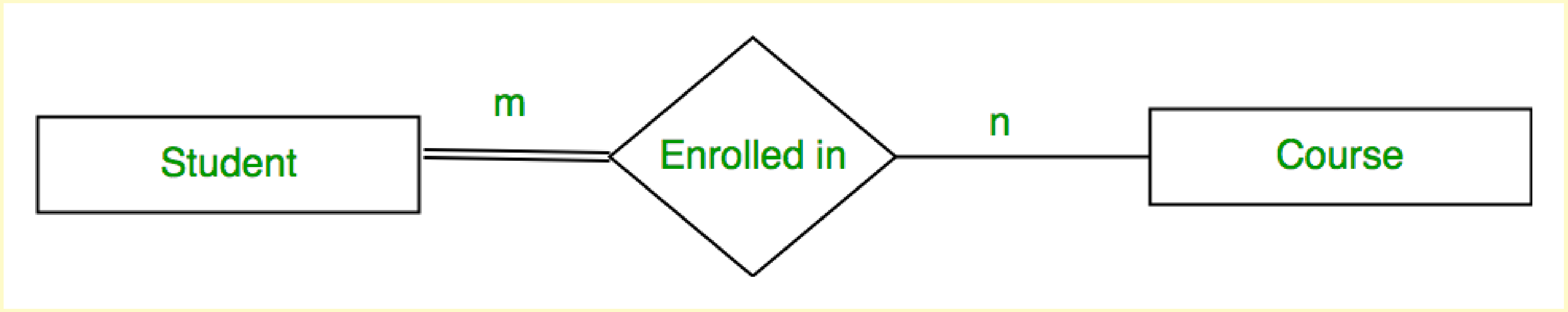
- **Participation.Constraint:**

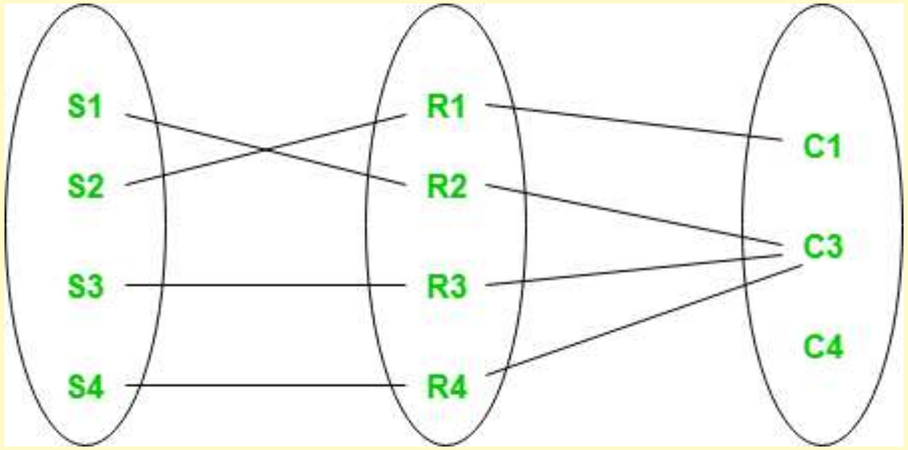
Participation Constraint is applied on the entity participating in the relationship set.

- **Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

- **Partial Participation –** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.The diagram depicts the ‘Enrolled in’ relationship set with Student Entity set having total participation and Course Entity set having partial participation.

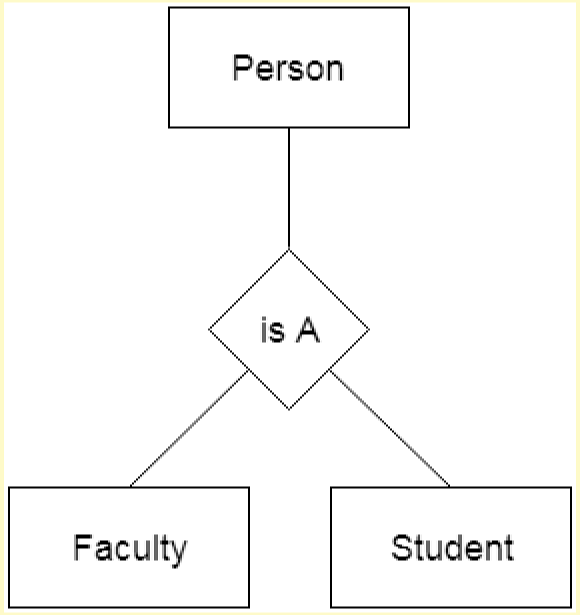
-


Student
m
Enrolled in
n
Course


S1
S2
S3
S4
R1
R2
R3
R4
C1
C3
C4

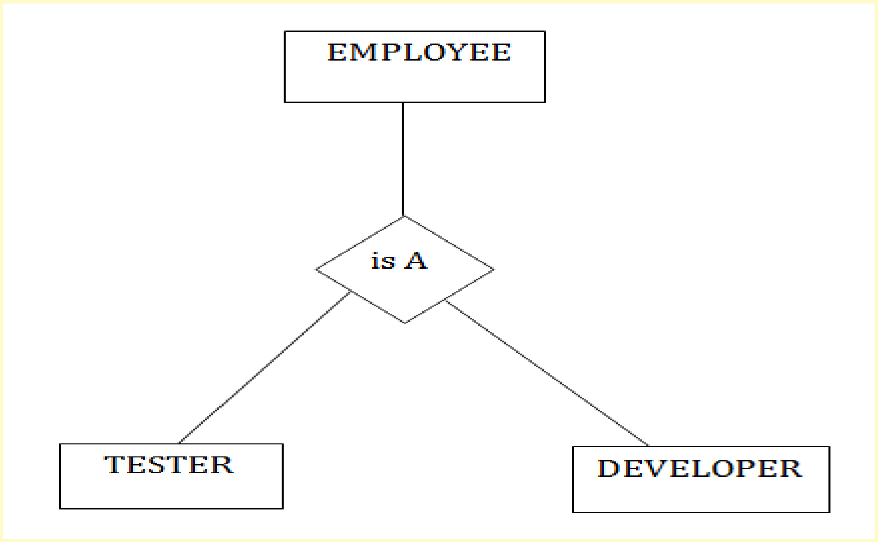
- **Binary Relationship –**
  When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.

- Generalization
- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
-



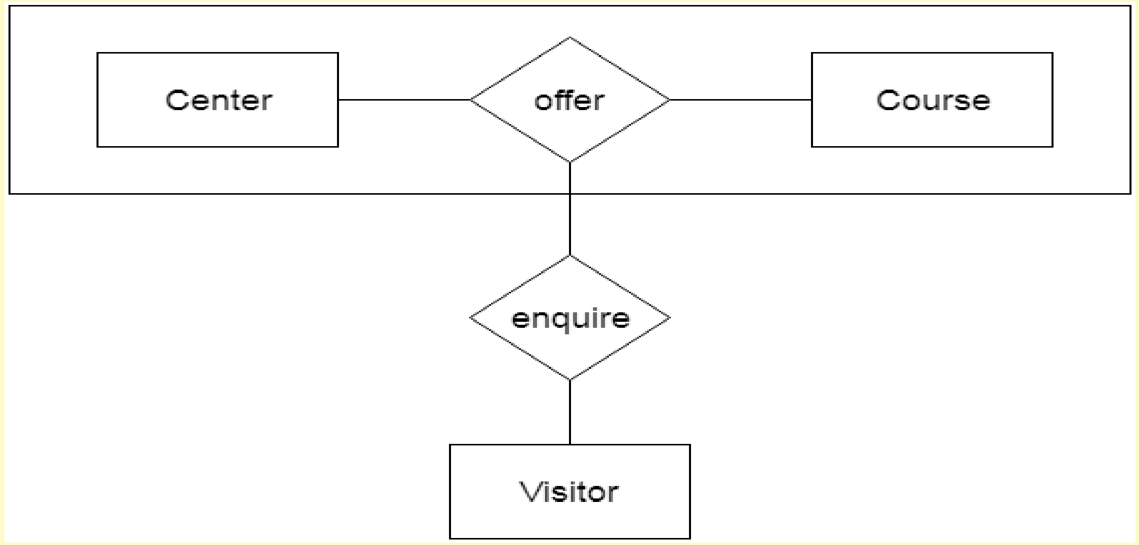
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.
- **For example,** Faculty and Student entities can be generalized and create a higher level entity Person.

- Specialization
- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

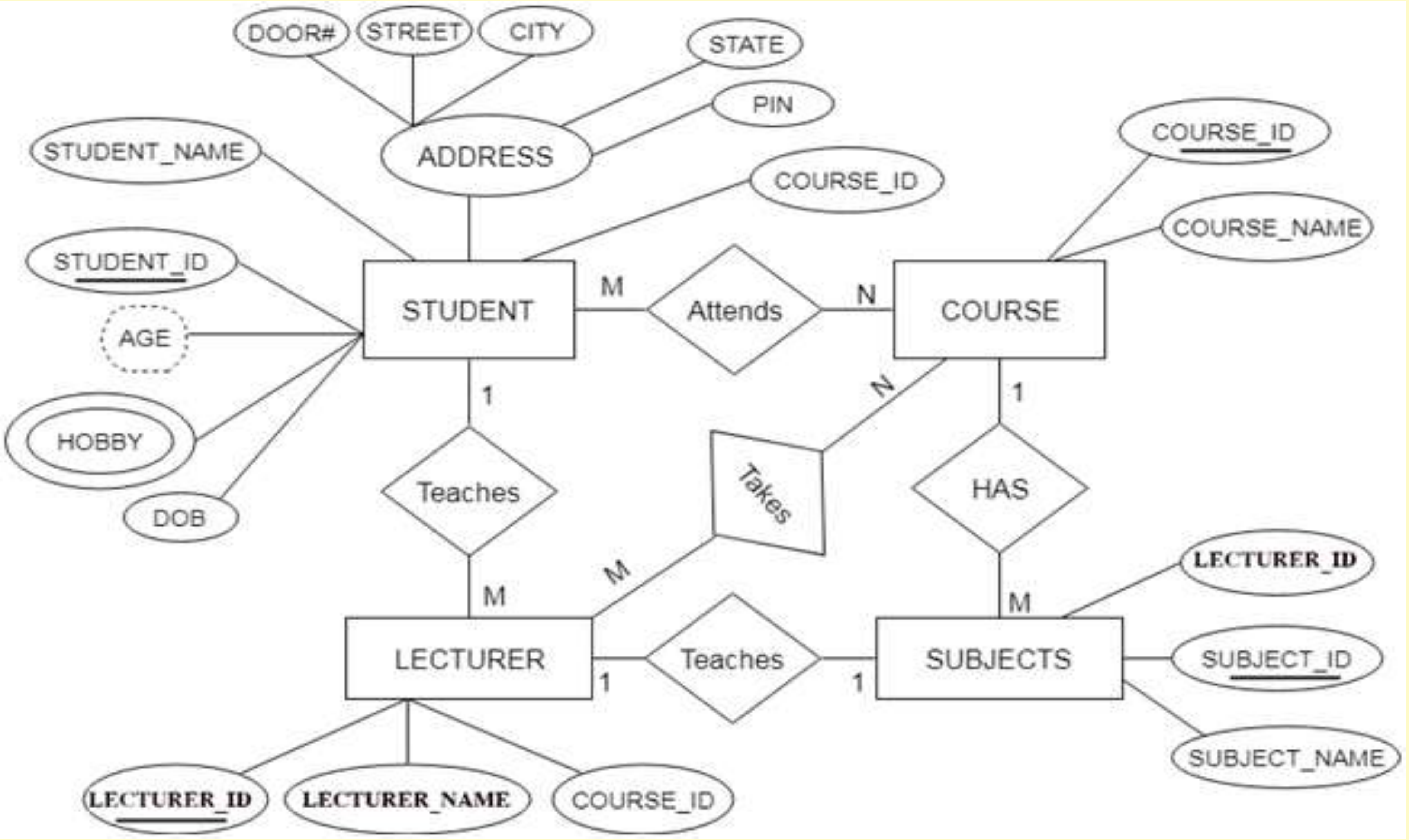


- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.
- **For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.

- Aggregation
- In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.



- Reduction of ER diagram to Table
- The database can be represented using the notations, and these notations can be reduced to a collection of tables.
- In the database, every entity set or relationship set can be represented in tabular form.
- **The ER diagram is given below:**

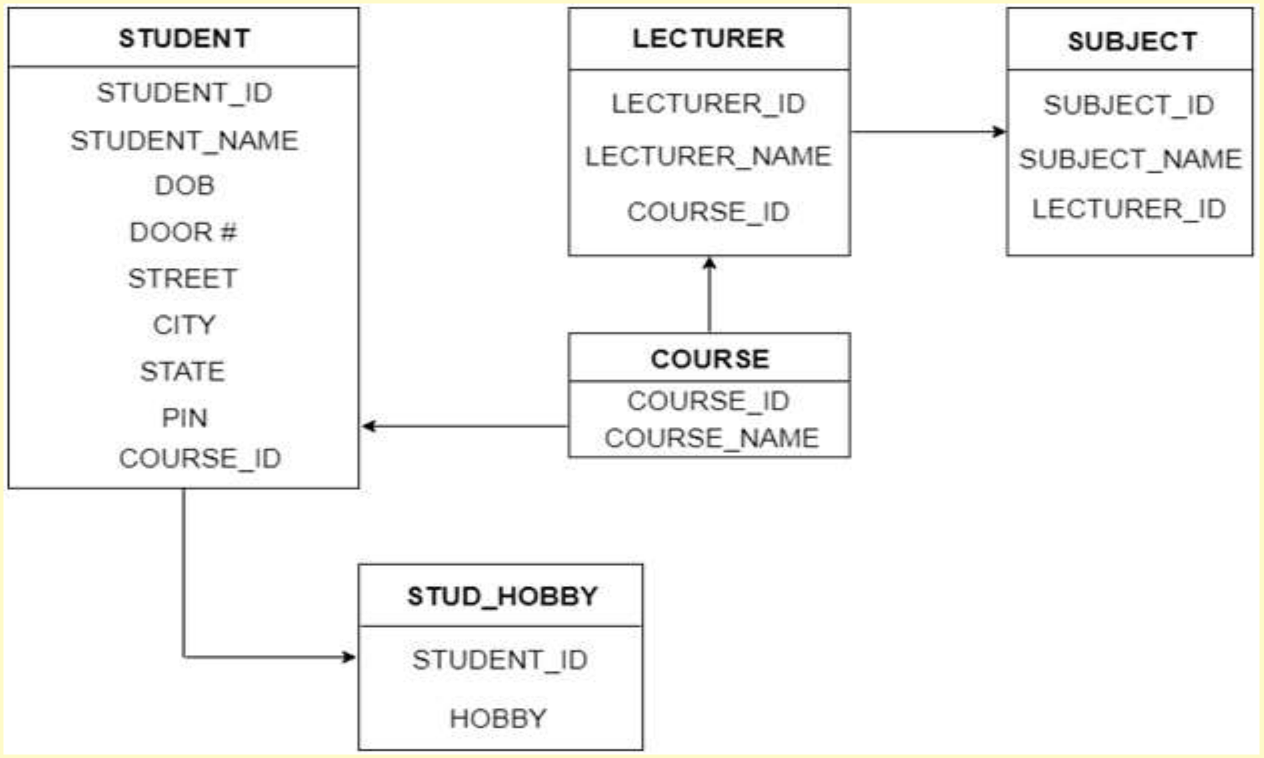

DOOR#
STREET
CITY
STATE
PIN
ADDRESS
STUDENT_NAME
COURSE_ID
COURSE_ID
COURSE_NAME
STUDENT_ID
STUDENT
M
Attends
N
COURSE
AGE
HOBBY
DOB
1
N
1
Teaches
Takes
HAS
M
M
M
LECTURER_ID
LECTURER
1
Teaches
1
SUBJECTS
SUBJECT_ID
SUBJECT_NAME
LECTURER_ID
LECTURER_NAME
COURSE_ID

- **<u>Conceptual database design steps are:</u>**
- Build a conceptual data model
- Recognize entity types
- Recognize the relationship types
- Identify and connect attributes with entity or relationship types
- Determine attribute domains
- Determine candidate, primary, and alternate key attributes
- Consider the use of improved modeling concepts (optional step)
- Check model for redundancy
- Validate the conceptual model against user transactions
- Review the conceptual data model with user
-

- **Building a Conceptual Data Model**
- The first step in conceptual database design is to build one (or more) conceptual data replica of the data requirements of the enterprise. A conceptual data model comprises these following elements:
- entity types
- types of relationship
- attributes and the various attribute domains
- primary keys and alternate keys
- integrity constraints
- The conceptual data model is maintained by documentation, including ER diagrams and a data dictionary, which is produced throughout the development of the model.

- There are some points for converting the ER diagram to the table:
- **Entity type becomes a table.**
- In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.
- **All single-valued attribute becomes a column for the table.**
- In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.
- **A key attribute of the entity type represented by the primary key.**
- In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.

- **The multivalued attribute is represented by a separate table.**
- In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.
- **Composite attribute represented by components.**
- In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.
- **Derived attributes are not considered in the table.**
- In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

- Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:
- 

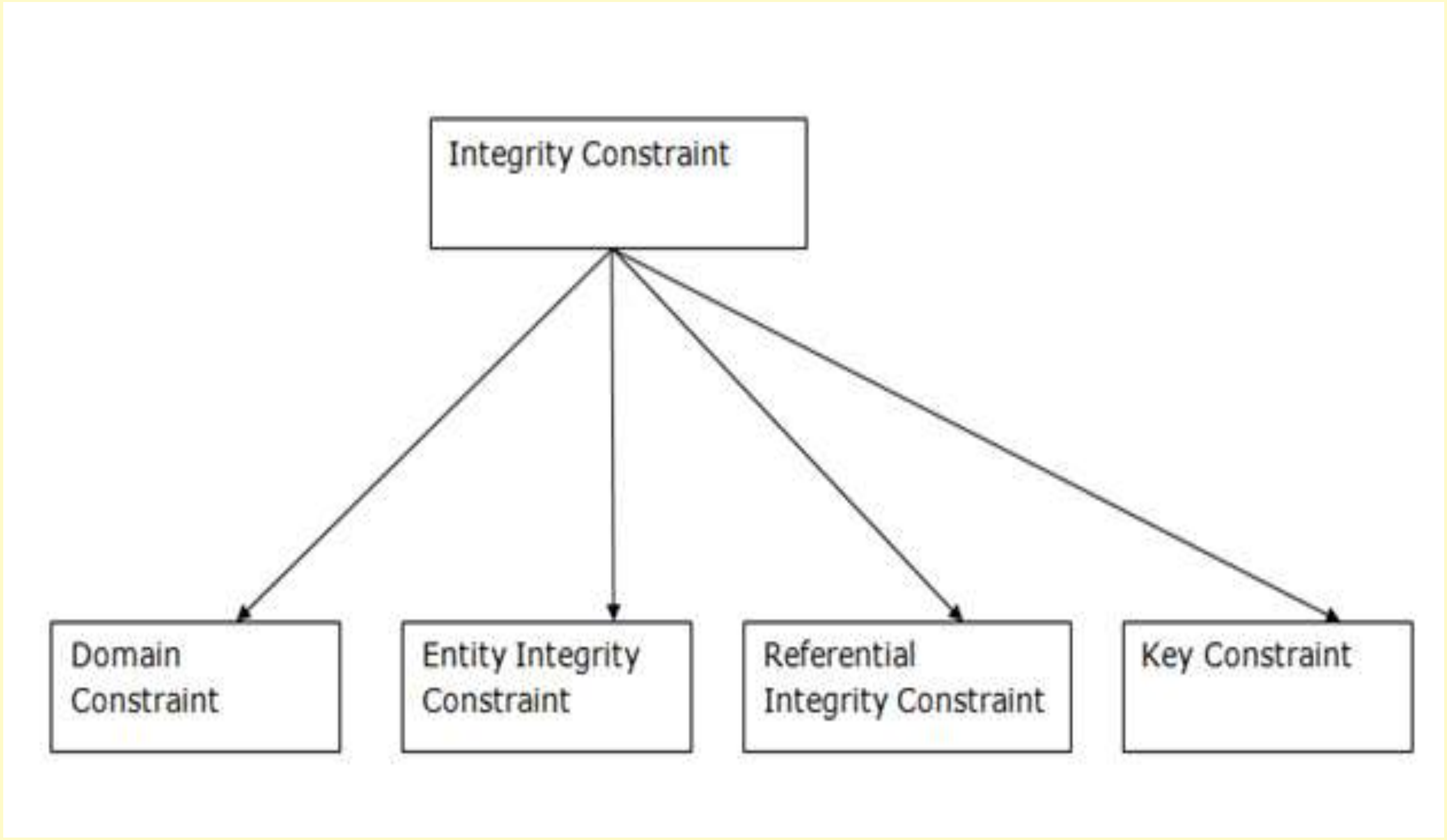
Integrity Constraints UNIT-2

Integrity constraints are a set of rules. It is used to maintain the quality of information.

Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.

Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint

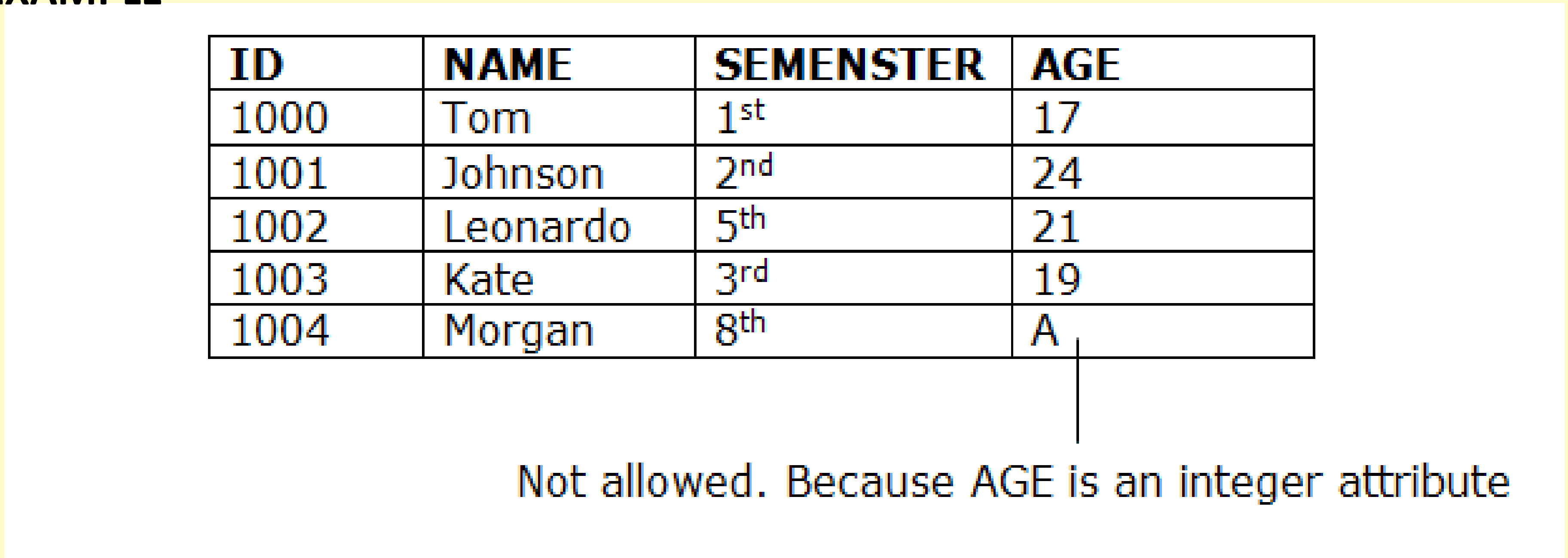


1. Domain constraints

Domain constraints can be defined as the definition of a valid set of values for an attribute.

The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

EXAMPLE

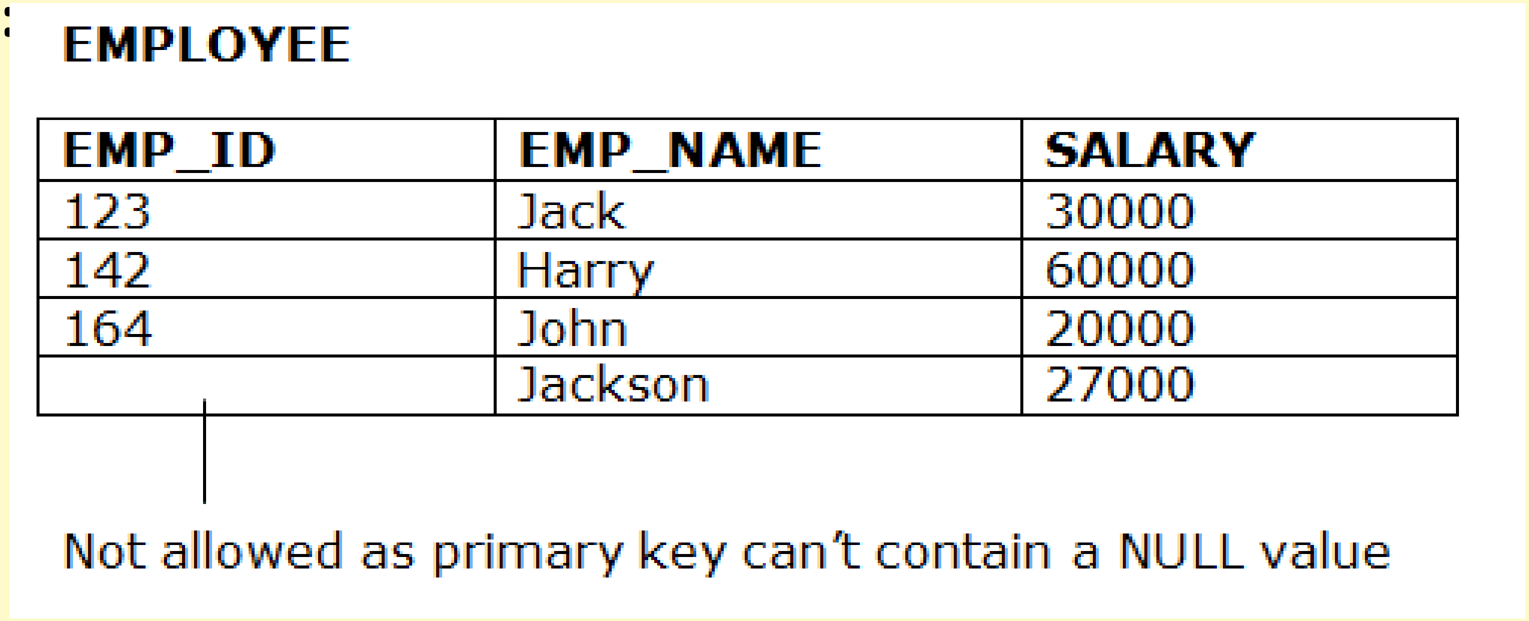


ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

The entity integrity constraint states that primary key value can't be null.
This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
A table can contain a null value other than the primary key field.
Example:



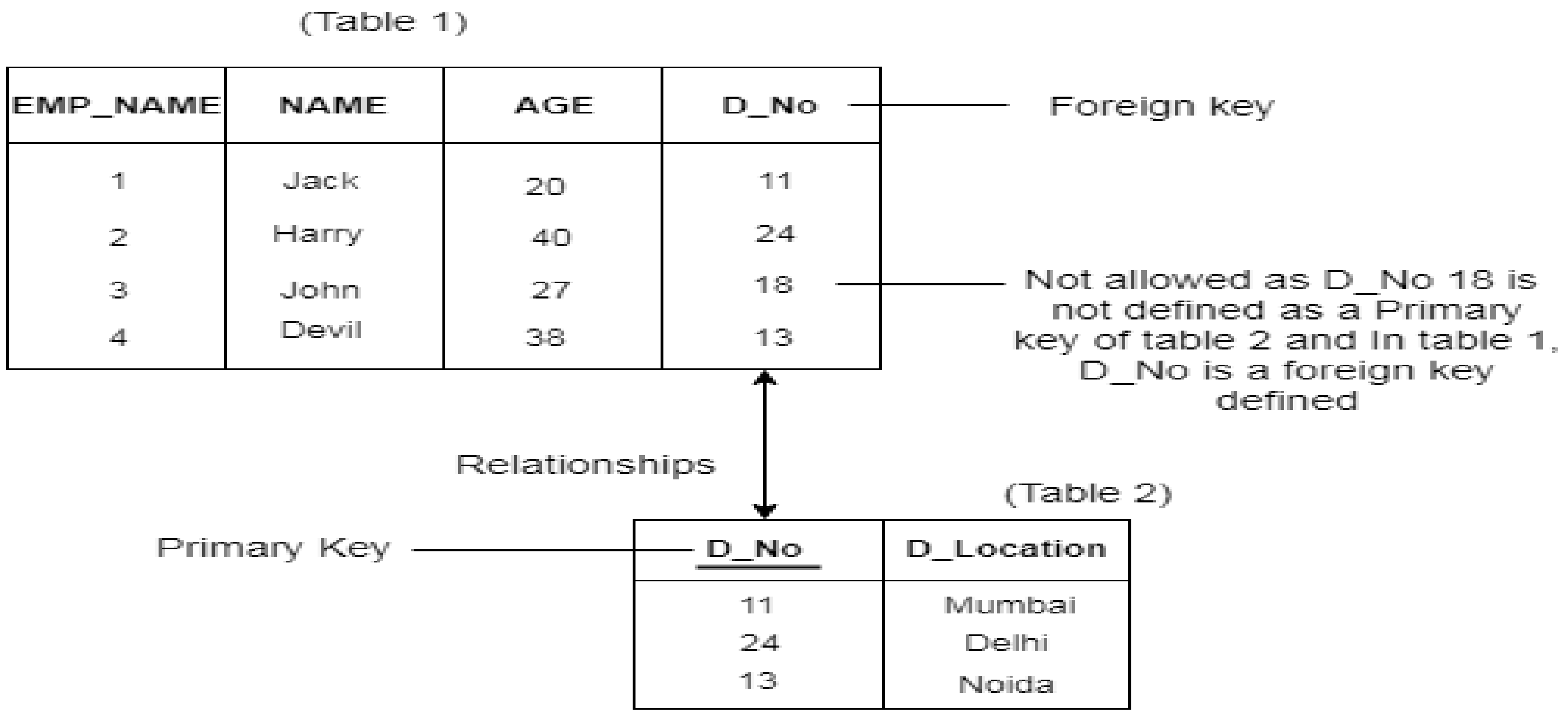
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

A referential integrity constraint is specified between two tables.
In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. **Example:**


(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

(Table 2)

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

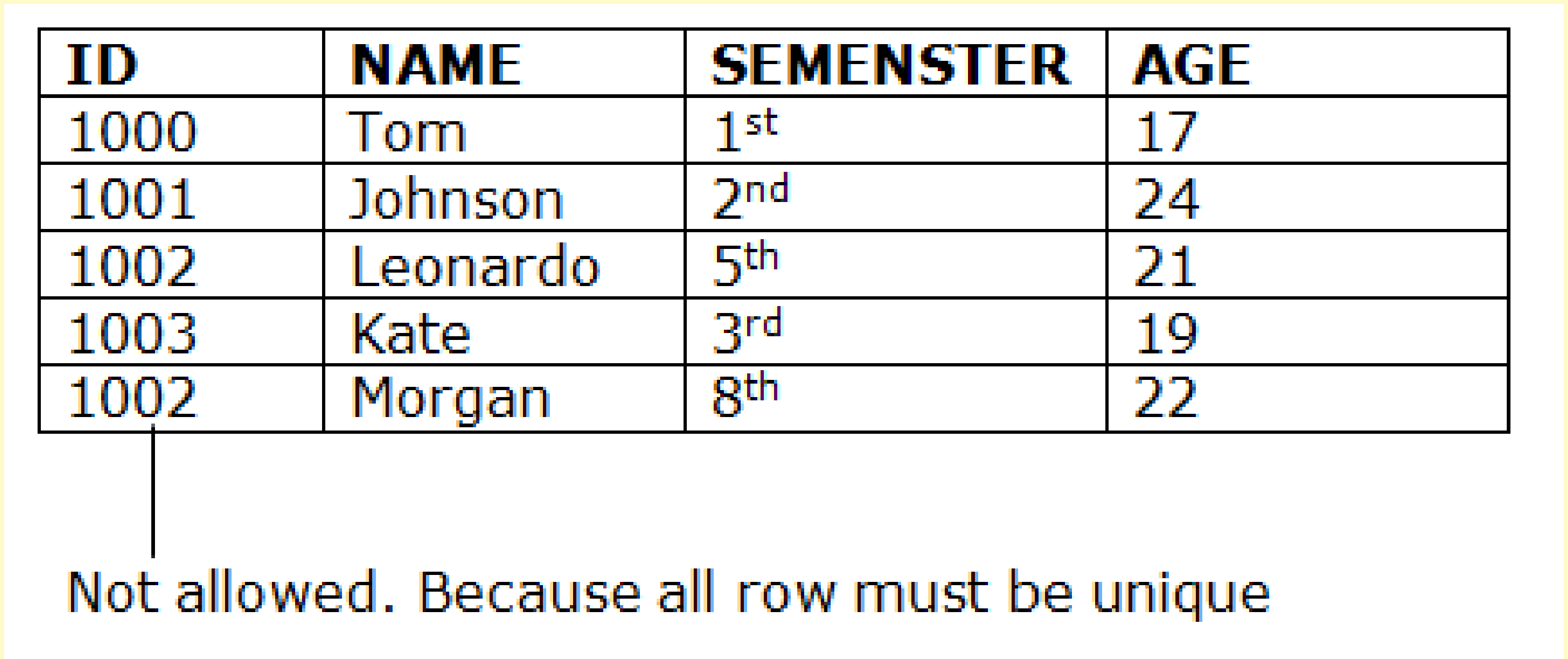
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
Primary Key

4. Key constraints

Keys are the entity set that is used to identify an entity within its entity set uniquely.

An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique value in the relational table.

Example:



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

SQL Statements

```
Show databases;
Create database db_name;
Use db_name;
Select database();
```

Creating Table

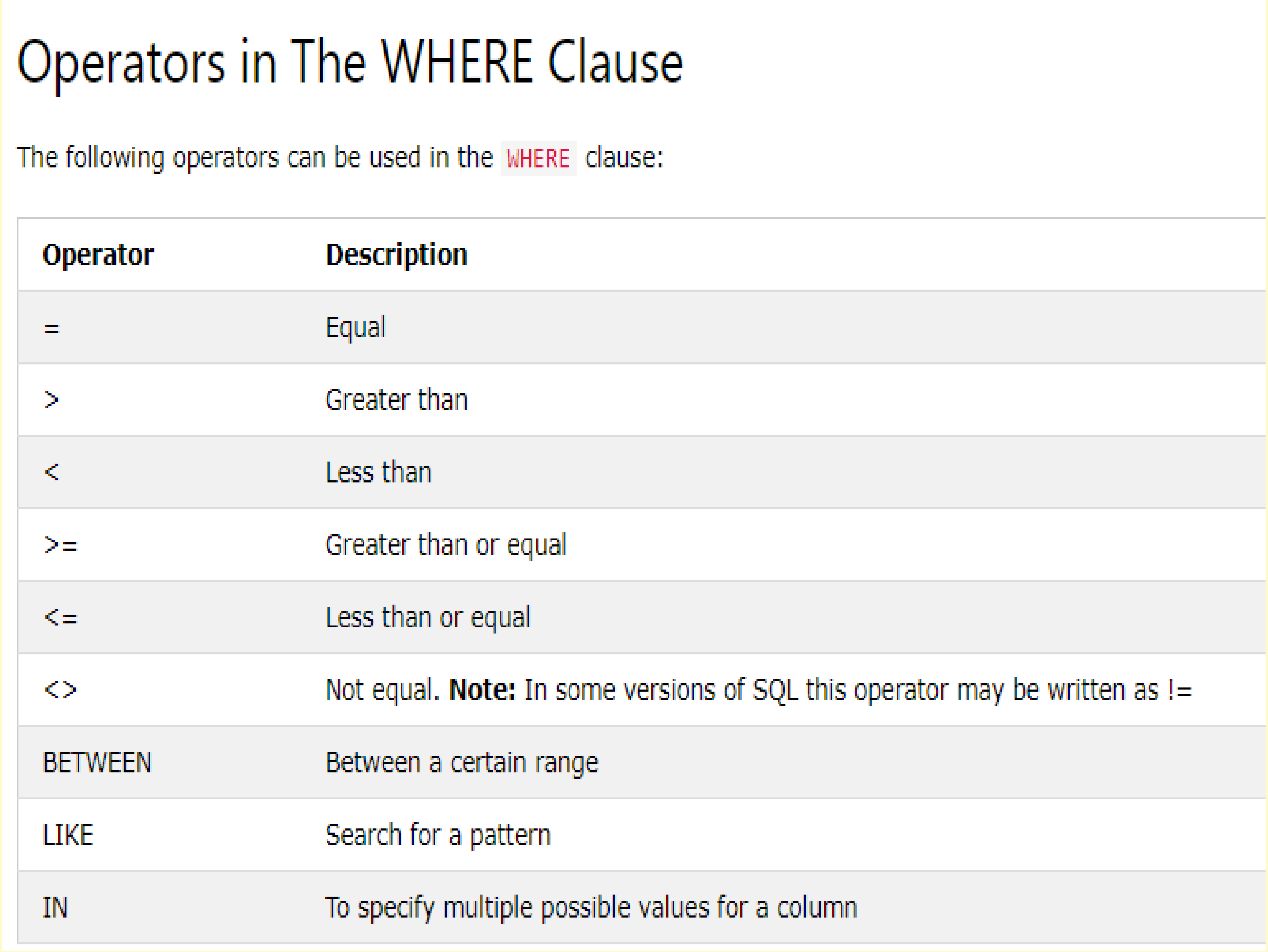
```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
 );
```

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Select * from table_nm

Insert into tn_name values(v1,v2,v3,v4,v5)

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;

SELECT * FROM Customers
WHERE Country='Mexico';
```



Operators in The WHERE Clause

The following operators can be used in the `WHERE` clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL Create Constraints

Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

SQL Constraints

SQL constraints are used to specify rules for the data in a table.
Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value

- `UNIQUE` - Ensures that all values in a column are different

- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table

- `FOREIGN KEY` - Prevents actions that would destroy links between tables

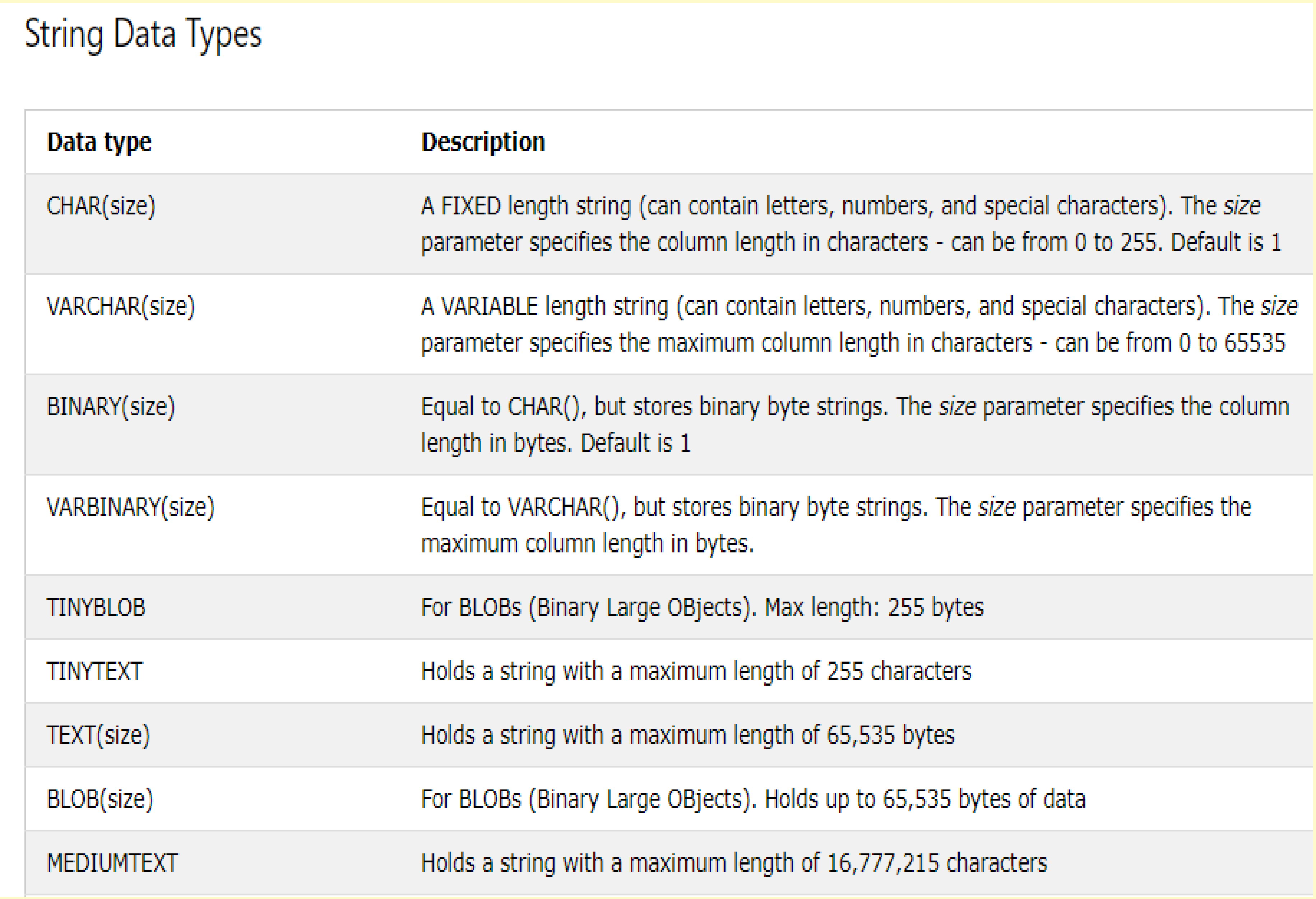
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

```
  CREATE TABLE Persons (
     ID int NOT NULL,
     LastName varchar(255) NOT NULL,
     FirstName varchar(255) NOT NULL,
     Age int
  );

CREATE TABLE Persons (
   ID int NOT NULL UNIQUE,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int
);

  CREATE TABLE Persons (
     ID int NOT NULL,
     LastName varchar(255) NOT NULL,
     FirstName varchar(255),
     Age int,
     PRIMARY KEY (ID)
  )
```

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);



CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);

 create table pp (mks int,rno int,foreign key(rno) references mm(rno));
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);

   CREATE INDEX idx_lastname
   ON Persons (LastName);


   Select * from where condn and
    Select f1,f2,f3  from where condn and

   Update table nm


   Delete from tn;
   Drop table tn;
   Truncate table tn;
```

```
SELECT * from tutorials_tbl -> WHERE tutorial_author LIKE '%jay';


select *  from mm where sname like 'a%';
select *  from mm where sname like 'a*';

SELECT * from tutorials_tbl ORDER BY tutorial_author ASC

UPDATE tutorials_tbl -> SET tutorial_title = 'Learning JAVA' -> WHERE tutorial_id = 3;


ALTER TABLE testalter_tbl DROP i;

ALTER TABLE testalter_tbl ADD i INT;

ALTER TABLE testalter_tbl RENAME TO alter_tbl;

SELECT * FROM tcount_tbl WHERE tutorial_count = NULL;
```



String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters

MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the

FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

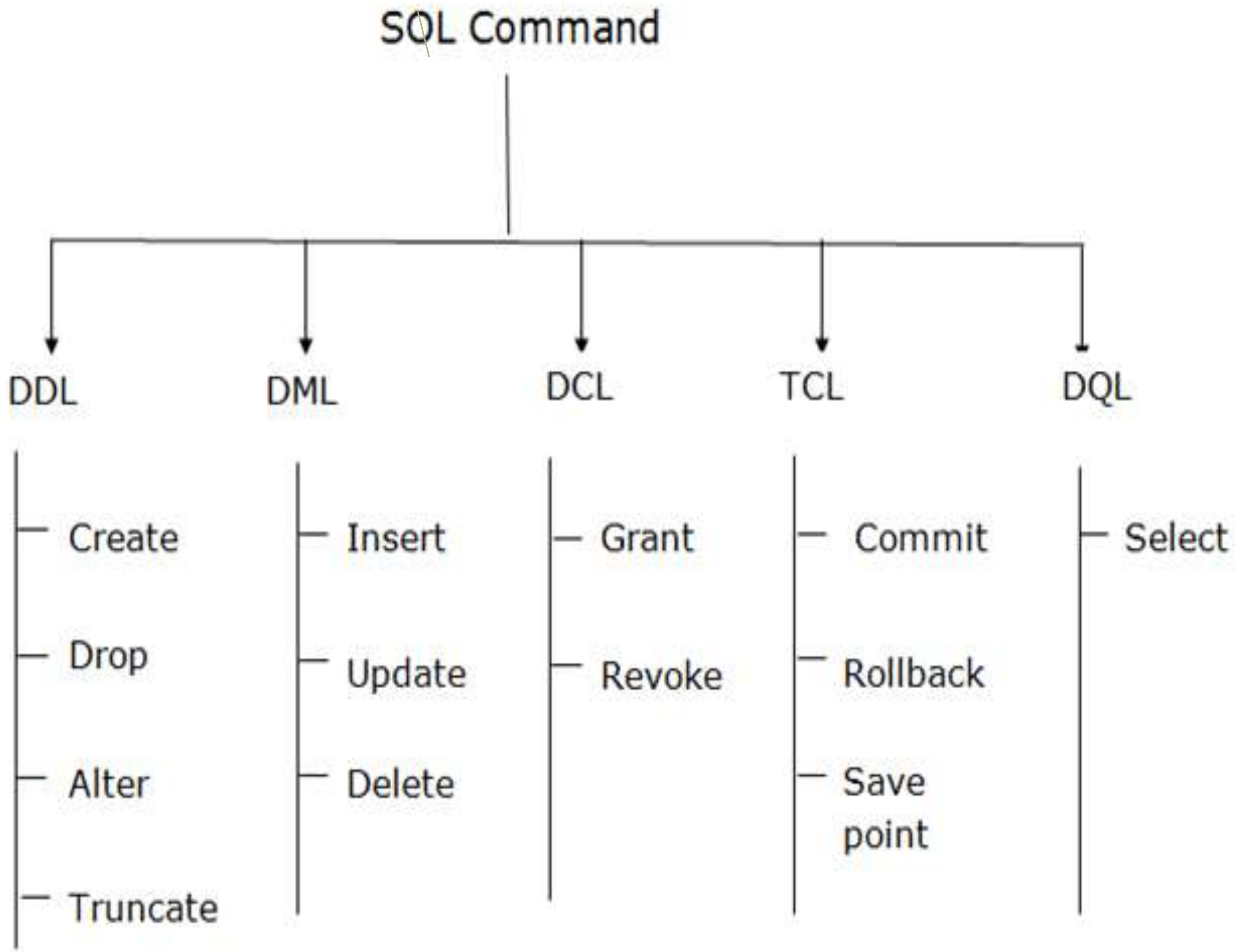
SQL Commands

SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.


SQL Command
DDL
DML
DCL
TCL
DQL
Create
Drop
Alter
Truncate
Insert
Update
Delete
Grant
Revoke
Commit
Rollback
Save point
Select

1. Data Definition Language (DDL)

DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.

All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

CREATE
ALTER
DROP
TRUNCATE

- Create table tname(c1 dt1,c2,dt2,c3 dt3);
- ALTER TABLE Customers
  ADD Email varchar(255);
- ALTER TABLE *table_name*
  DROP COLUMN *column_name*;
- ALTER TABLE *table_name*
  ALTER COLUMN *column_name datatype*;
- ALTER TABLE Persons
  ADD DateOfBirth date;

- DROP TABLE *table_name*;
- TRUNCATE TABLE *table_name*;

2. Data Manipulation Language

DML commands are used to modify the database. It is responsible for all form of changes in the database.

The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:
INSERT
UPDATE
DELETE

- UPDATE *table_name*
  SET *column1* = *value1*, *column2* = *value2*, ...
  WHERE *condition*;

- UPDATE Customers
  SET ContactName = 'Alfred Schmidt',
  City= 'Frankfurt'
  WHERE CustomerID = 1;
- UPDATE Customers
  SET ContactName='Juan';

- Change the sal of employees in emp table by 12%(plus) where desig=‘class-2’

- DELETE FROM *table_name* WHERE *condition*;
- DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
- DELETE FROM *table_name*;

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

Grant
Revoke

- **GRANT SELECT ON** employees **TO** bob@localhost;
- **GRANT INSERT**, **UPDATE**, **DELETE ON** employees **TO** bob@localhost;
- **GRANT DELETE ON** classicmodels.employees **TO** bob@localhsot;
- **GRANT INSERT ON** classicmodels.* **TO** bob@localhost;

- **REVOKE INSERT**, **UPDATE ON** classicmodels.* **FROM** rfc@localhost;

- REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
- REVOKE 'role1', 'role2' FROM 'user1'@'localhost', 'user2'@'localhost';

- REVOKE SELECT ON world.* FROM 'role3';

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

COMMIT
ROLLBACK
SAVEPOINT

5. Data Query Language

DQL is used to fetch the data from the database.
It uses only one command:

SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Views in SQL

Views in SQL are considered as a virtual table. A view also contains rows and columns.

To create the view, we can select the fields from one or more tables present in the database.

A view can either have specific rows based on certain condition or all the rows of a table.

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

SELECT * FROM DETAILSVIEW
```

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

ALTER salesOrders **AS SELECT** orderNumber,
customerNumber, productCode, quantityOrdered,
priceEach, **status FROM** orders

```
DROP VIEW view_name;
```

LOGICAL DATABASE DESIGN

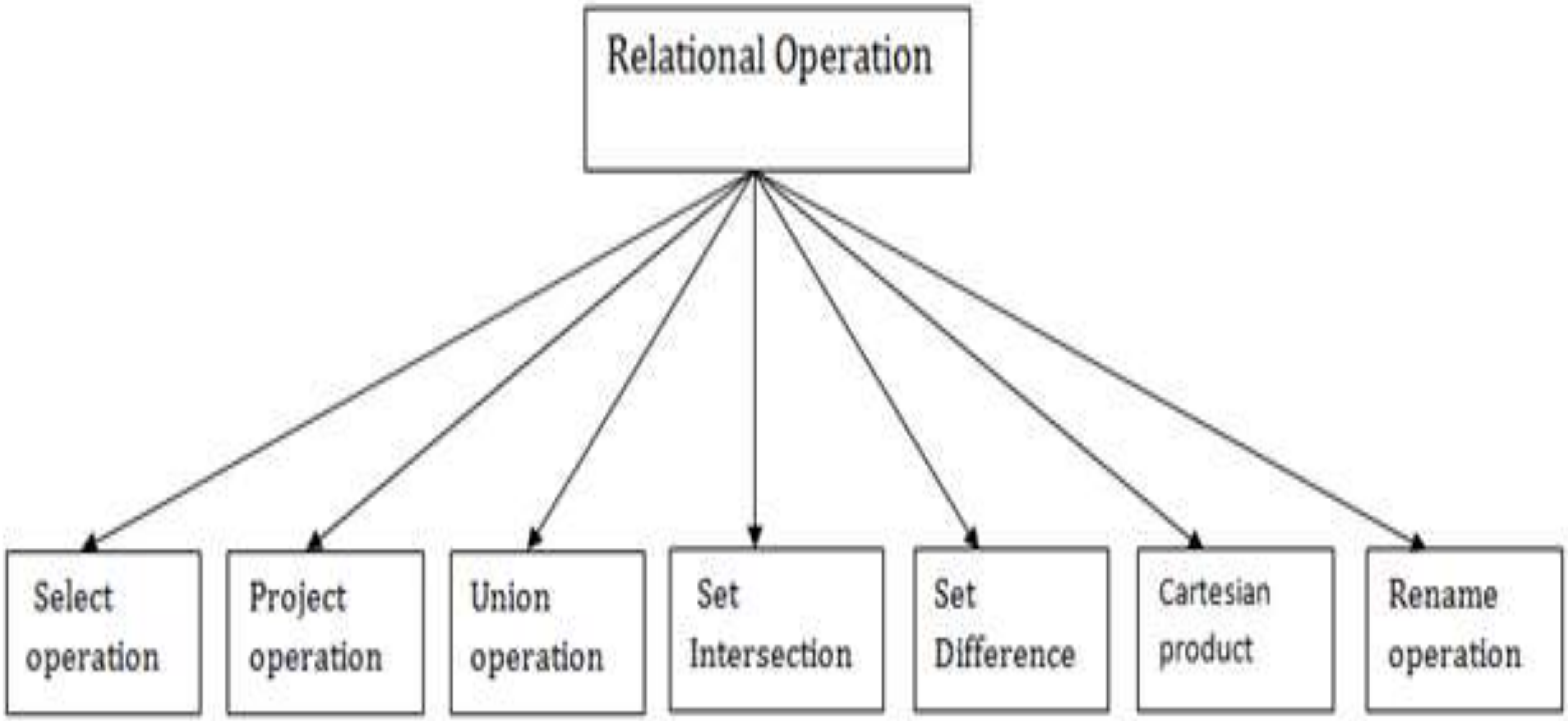
Logical database design is the process of deciding how to arrange the attributes of the entities in a given business environment into database structures, such as the tables of a relational database. The goal of logical database design is to create well structured tables that properly reflect the company's business environment. The tables will be able to store data about the company's entities in a non-redundant manner and foreign keys will be placed in the tables so that all the relationships among the entities will be supported. Physical database design, which will be treated in the next chapter, is the process of modifying the logical database design to improve performance.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.
The fundamental operations of relational algebra are as follows −

Select
Project
Union
Set different
Cartesian product
Rename

Types of Relational operation



Select Operation (σ)

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

Notation – $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example –

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

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

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

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

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

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

Project Operation (∏)

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

Notation – $\prod_{A1, A2, An} (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.
Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\prod_{subject, author}$ (Books) Selects and projects columns named as subject and author from the relation Books.

Union Operation (U)

It performs binary union between two given relations and is defined as −

r U s = { t | t ∈ r or t ∈ s} **Notation** − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

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

r, and **s** must have the same number of attributes.

Attribute domains must be compatible.

Duplicate tuples are automatically eliminated

$\prod_{\text{author}}$ (Books) ∪ $\prod_{\text{author}}$ (Articles)

Output – Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – **r** – **s**

Finds all the tuples that are present in **r** but not in **s**.
$\prod_{\text{author}}$ (Books) − $\prod_{\text{author}}$ (Articles) **Output** – Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.
Notation − r X s

Where **r** and **s** are relations and their output will be defined as −
r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{\text{author = 'tutorialspoint'}}$(Books X Articles)

− Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation – ρ_{x} (E)]

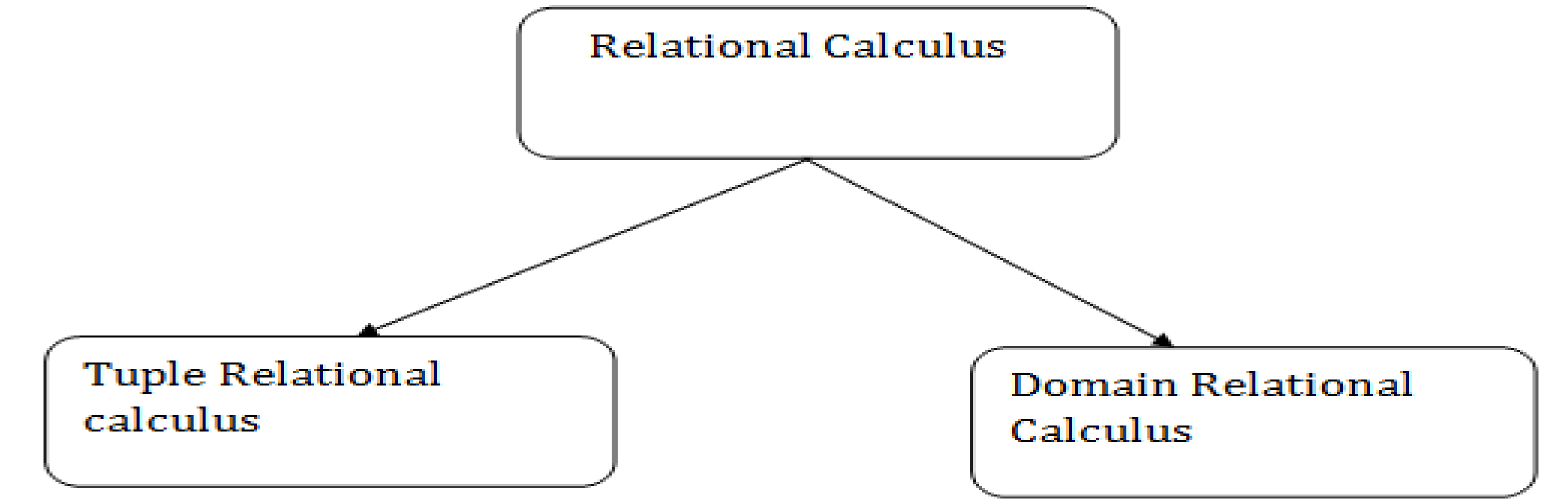
ρ(STUDENT1, STUDENT)

Relational Calculus

Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.

The relational calculus tells what to do but never explains how to do.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.

The result of the relation can have one or more tuples.

Notation:

{T | P (T)}   or {T | Condition (T)}

Table-1: Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table-3: Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table-2: Branch

Branch name	Branch city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-4: Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

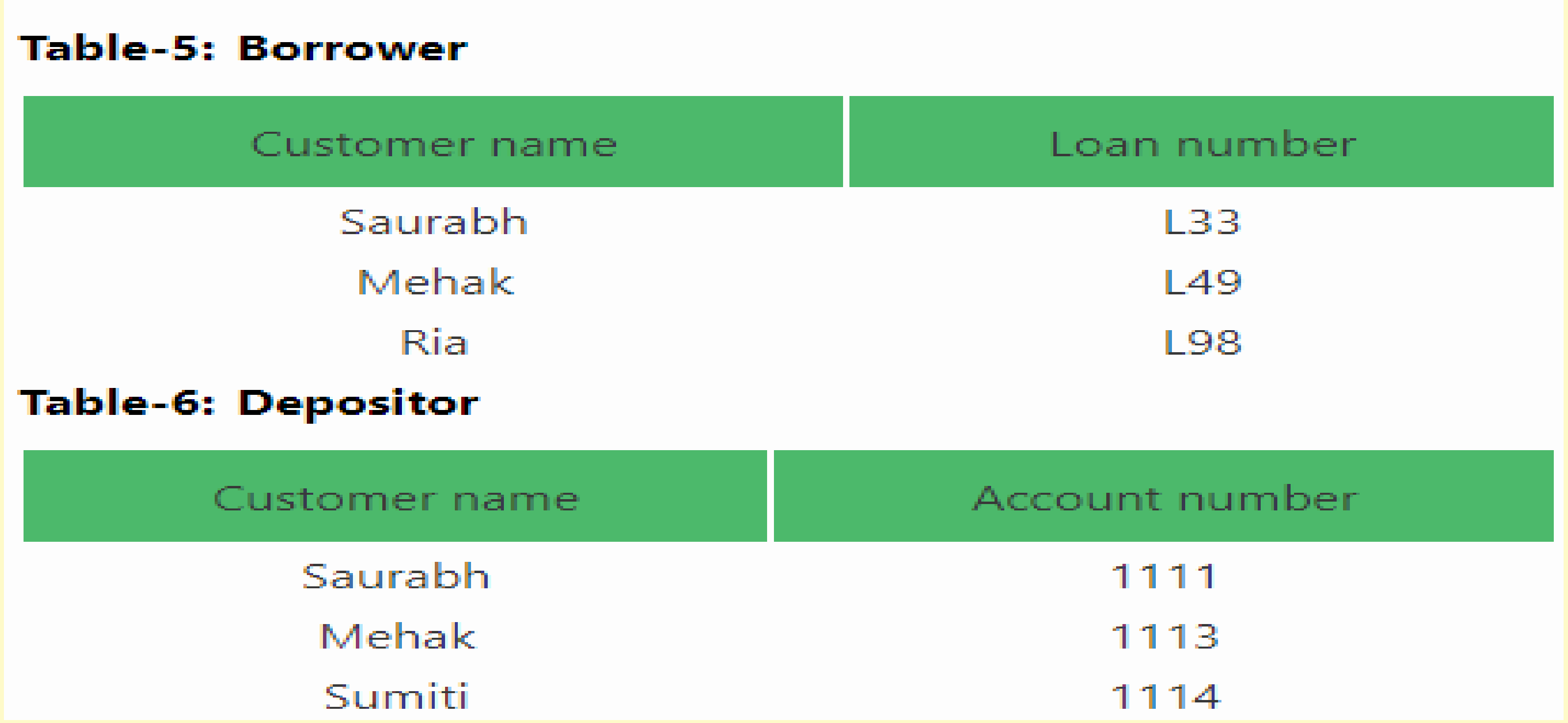


Table-5: Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table-6: Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Sumiti	1114

Cust(cname,street,city)

Branch(br-name,br-city)

Account(ano,br-name,bal)
Loan(lno,br-nm,amt)
Borrow(c-name,lno)

Depo(c-name,accno)

`display the details of the customer having an account

Display the details of the customer having an account and balance >=20000

Queries-1: Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$$\{t \mid t \in \text{loan} \land t[\text{amount}] >= 10000\}$$

Queries-2: Find the loan number for each loan of an amount greater or equal to 10000.

$$\{t \mid \exists\, s \in \text{loan}(t[\text{loan number}] = s[\text{loan number}] \land s[\text{amount}] >= 10000)\}$$

Queries-3: Find the names of all customers who have a loan and an account at the bank.

$$\{t \mid \exists\, s \in \text{borrower}(\, t[\text{customer-name}] = s[\text{customer-name}]) \land \exists\, u \in \text{depositor}(\, t[\text{customer-name}] = u[\text{customer-name}])\}$$

Queries-4: Find the names of all customers having a loan at the “ABC” branch.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name] ∧ ∃ u ∈

loan(u[branch-name] = “ABC” ∧ u[loan-number] = s[loan-number]))}

Domain Relational Calculus

Find loan-no,br-name,amt for loan above 15000

{<l,b,a>|<l,b,a> E loan ^ a >15000}

Find loan-no for loan >20000

{<l>|<l,b,a> E loan ^ a>20000}

Find the name of all the customers who have a loan from hyd br and find the loan amt

{<n,a> |∃ L (<n,l>E borrower ^ ∃ b(<l,b,a>E loan ^ b=‘hyd’’))}

Domain Relational Calculus (DRC)

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

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

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

For example,

$\{< name, age > \mid \in Student \land age > 17\}$

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

Domain Relational Calculus Query Examples-1

Query : Find the loan number, branch name and amount for loans of over \$1500

$$\{\langle l, b, a\rangle \mid \langle l, b, a\rangle \in loan \wedge a > 1500\}$$

Query : Find all loan numbers for loans with an amount greater than \$1500.

$$\{\langle l\rangle \mid \exists\, b, a\,(\langle l, b, a\rangle \in loan \wedge a > 1500)\}$$

UNIT-3

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

UNION Syntax

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

- UNION ALL Syntax
- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

- SELECT *column_name(s)* FROM *table1*
  UNION ALL
  SELECT *column_name(s)* FROM *table2*;

- The SQL SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values.

- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

- SELECT DISTINCT Syntax

- SELECT DISTINCT *column1*, *column2, ...*
  FROM *table_name*;

-

- The SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.

- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.
- ORDER BY Syntax

- SELECT *column1, column2, ...*
  FROM *table_name*
  ORDER BY *column1, column2, ...* ASC|DESC;

- IS NULL Syntax
- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NULL;

- IS NOT NULL Syntax

- SELECT *column_names*
  FROM *table_name*
  WHERE *column_name* IS NOT NULL;

- The SQL UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.

- UPDATE Syntax

- UPDATE *table_name*
  SET *column1* = *value1*, *column2* = *value2*, ...
  WHERE *condition*;

- SELECT TOP 3 * FROM Customers;

- The SQL MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.

- The MAX() function returns the largest value of the selected column.

- MIN() Syntax
- SELECT MIN(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- MAX() Syntax
- SELECT MAX(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- SELECT MIN(Price) AS SmallestPrice
  FROM Products;

- SELECT MAX(Price) AS LargestPrice
  FROM Products;

- The SQL COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criterion.

- COUNT() Syntax
- SELECT COUNT(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- The AVG() function returns the average value of a numeric column.
- AVG() Syntax

- SELECT AVG(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- The SUM() function returns the total sum of a numeric column.

- SUM() Syntax

- SELECT SUM(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- The SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

• SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.
- Alias Column Syntax

- SELECT *column_name* AS *alias_name*
  FROM *table_name;*

- The SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

- The IN operator is a shorthand for multiple OR conditions.

- IN Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*value1*, *value2*, ...);

- SELECT * FROM Customers
  WHERE Country IN ('Germany', 'France', 'UK');

- SELECT * FROM Customers
  WHERE Country NOT IN ('Germany', 'France', 'UK');

- SELECT * FROM Customers
  WHERE Country IN (SELECT Country FROM Supplie
  rs);
-

- The SQL BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

- The BETWEEN operator is inclusive: begin and end values are included.

- BETWEEN Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* BETWEEN *value1* AND *value2;*

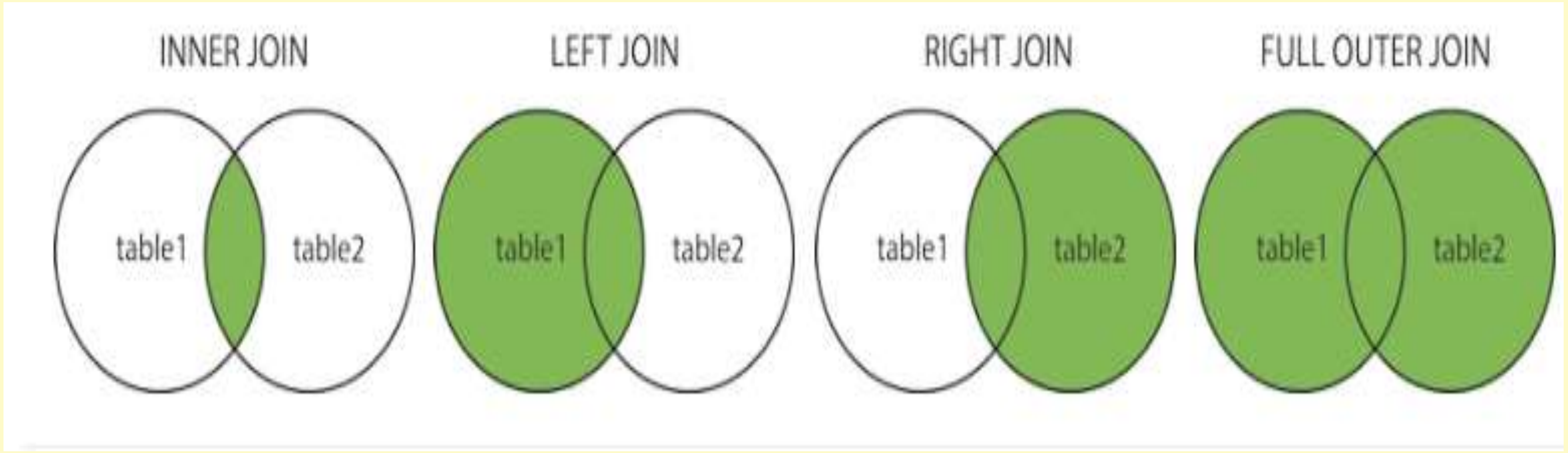
- SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
-

- SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- Different Types of SQL JOINs
- Here are the different types of the JOINs in SQL:
- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



- SQL INNER JOIN Keyword

- The INNER JOIN keyword selects records that have matching values in both tables.

- INNER JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name = table2.column_na
  me;*

- SELECT Orders.OrderID,
  Customers.CustomerName
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID
  = Customers.CustomerID;

- SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).
- The result is 0 records from the right side, if there is no match.

- LEFT JOIN Syntax
- SELECT *column_name(s)*
  FROM *table1*
  LEFT JOIN *table2*
  ON *table1.column_name* = *table2.column_name*;

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
  ORDER BY Customers.CustomerName;

- SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

- RIGHT JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  RIGHT JOIN *table2*
  ON *table1.column_name* = *table2.column_name*;

-
  ```
  SELECT Orders.OrderID, Employees.LastName,
  Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID
  = Employees.EmployeeID
  ORDER BY Orders.OrderID;
  ```

- SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

- **Tip:** FULL OUTER JOIN and FULL JOIN are the same.
- FULL OUTER JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  FULL OUTER JOIN *table2*
  ON *table1.column_name* = *table2.column_name*
  WHERE *condition*;

-
  ```sql
  SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;
  ```

- SQL Self Join

- A self join is a regular join, but the table is joined with itself.

- Self Join Syntax

- SELECT *column_name(s)*
  FROM *table1 T1, table1 T2*
  WHERE *condition*;

- SELECT A.CustomerName AS CustomerName1,
  B.CustomerName AS CustomerName2, A.City
  FROM Customers A, Customers B
  WHERE A.CustomerID <> B.CustomerID
  AND A.City = B.City
  ORDER BY A.City;

- create table s22 (eno int,ename VARCHAR(20),mngrno int);

- SELECT A.ENAME AS 'EMPLOYEE',B.ENAME AS 'WORKS FOR'
  FROM S22 A,S22 B WHERE A.MNGRNO=B.ENO;

- The SQL GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

- GROUP BY Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  ORDER BY *column_name(s);*

- SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country;

- SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT(CustomerID) DESC;

- The SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

- HAVING Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  HAVING *condition*
  ORDER BY *column_name(s);*

-
  ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5;
  ```
-

- The SQL EXISTS Operator
- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.
- EXISTS Syntax
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE EXISTS
  (SELECT *column_name* FROM *table_name* WHERE *conditio
  n*);
-

-
  ```
  SELECT SupplierName
  FROM Suppliers
  WHERE EXISTS (SELECT ProductName FROM P
  roducts WHERE Products.SupplierID =
  Suppliers.supplierID AND Price < 20);
  ```

- The SQL ANY and ALL Operators

- The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

- The SQL ANY Operator

- The ANY operator:

- returns a boolean value as a result

- returns TRUE if ANY of the subquery values meet the condition
- ANY means that the condition will be true if the operation is true for any of the values in the range.

-
  ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name operator ALL
    (SELECT column_name
    FROM table_name
    WHERE condition);
  ```

- SQL SELECT INTO Examples
- The following SQL statement creates a backup copy of Customers:
- SELECT * INTO CustomersBackup2017
  FROM Customers;

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

- Purpose of Normalization
- **Normalization** is the process of structuring and handling the relationship between data to minimize redundancy in the relational table and avoid the unnecessary anomalies properties from the database like insertion, update and delete.

- It helps to divide large database tables into smaller tables and make a relationship between them. It can remove the redundant data and ease to add, manipulate or delete table fields.

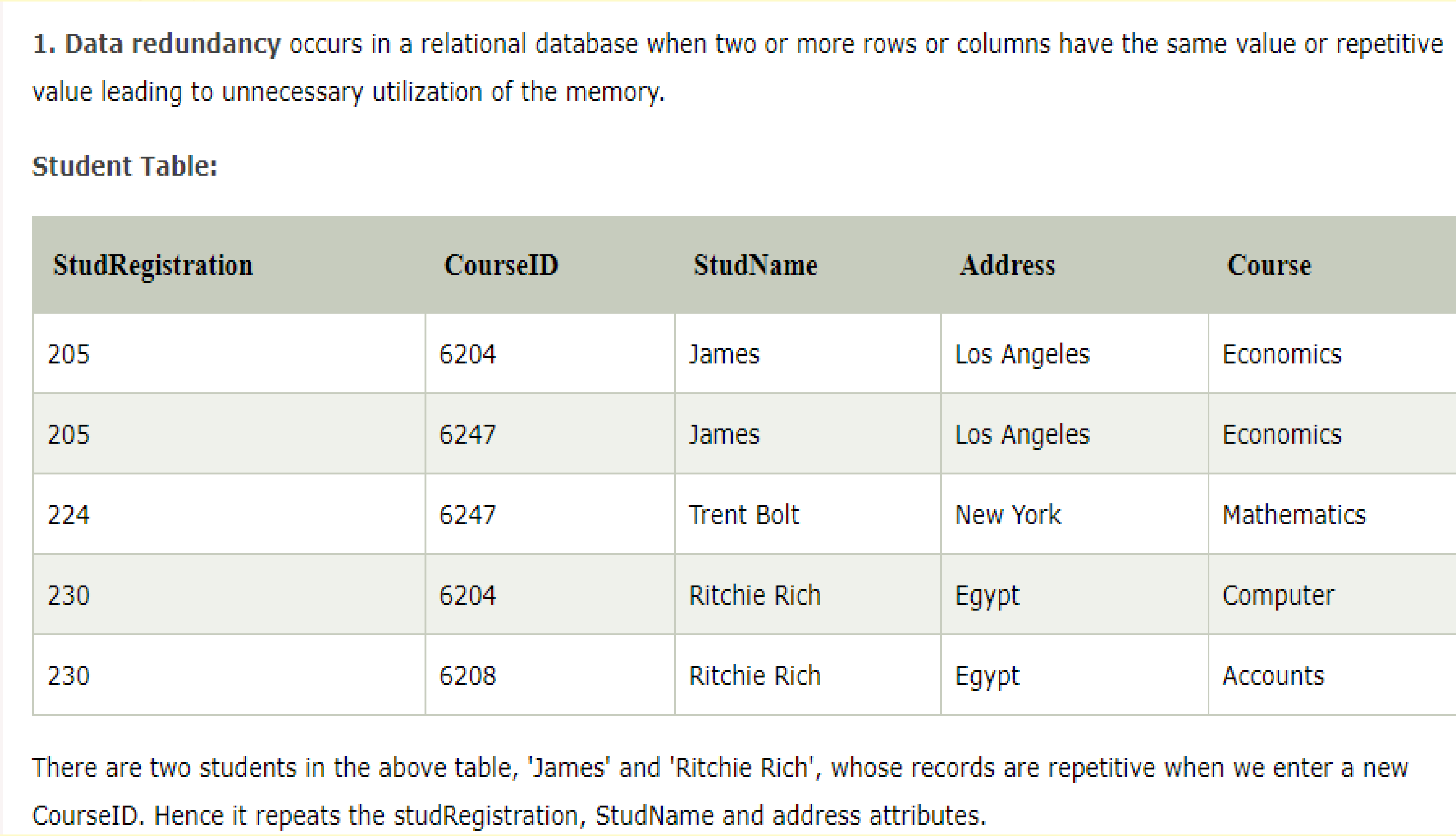
- A normalization defines rules for the relational table as to whether it satisfies the normal form.
- A **normal form** is a process that evaluates each relation against defined criteria and removes the multivalued, joins, functional and trivial dependency from a relation.
- If any data is updated, deleted or inserted, it does not cause any problem for database tables and help to improve the relational table' integrity and efficiency.

- Objective of Normalization

- It is used to remove the duplicate data and database anomalies from the relational table.
- Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
- It is helpful to divide the large database table into smaller tables and link them using relationship.
- It avoids duplicate data or no repeating groups into a table.
- It reduces the chances for anomalies to occur in a database.

- Types of Anomalies

Following are the types of anomalies that make the table inconsistency, loss of integrity, and redundant data.



1. Data redundancy occurs in a relational database when two or more rows or columns have the same value or repetitive value leading to unnecessary utilization of the memory.

Student Table:

StudRegistration	CourseID	StudName	Address	Course
205	6204	James	Los Angeles	Economics
205	6247	James	Los Angeles	Economics
224	6247	Trent Bolt	New York	Mathematics
230	6204	Ritchie Rich	Egypt	Computer
230	6208	Ritchie Rich	Egypt	Accounts

There are two students in the above table, 'James' and 'Ritchie Rich', whose records are repetitive when we enter a new CourseID. Hence it repeats the studRegistration, StudName and address attributes.

- **2. Insert Anomaly:**

- An insert anomaly occurs in the relational database when some attributes or data items are to be inserted into the database without existence of other attributes.

- For example, In the Student table, if we want to insert a new courseID, we need to wait until the student enrolled in a course. In this way, it is difficult to insert new record in the table. Hence, it is called insertion anomalies.

- **3. Update Anomalies:**

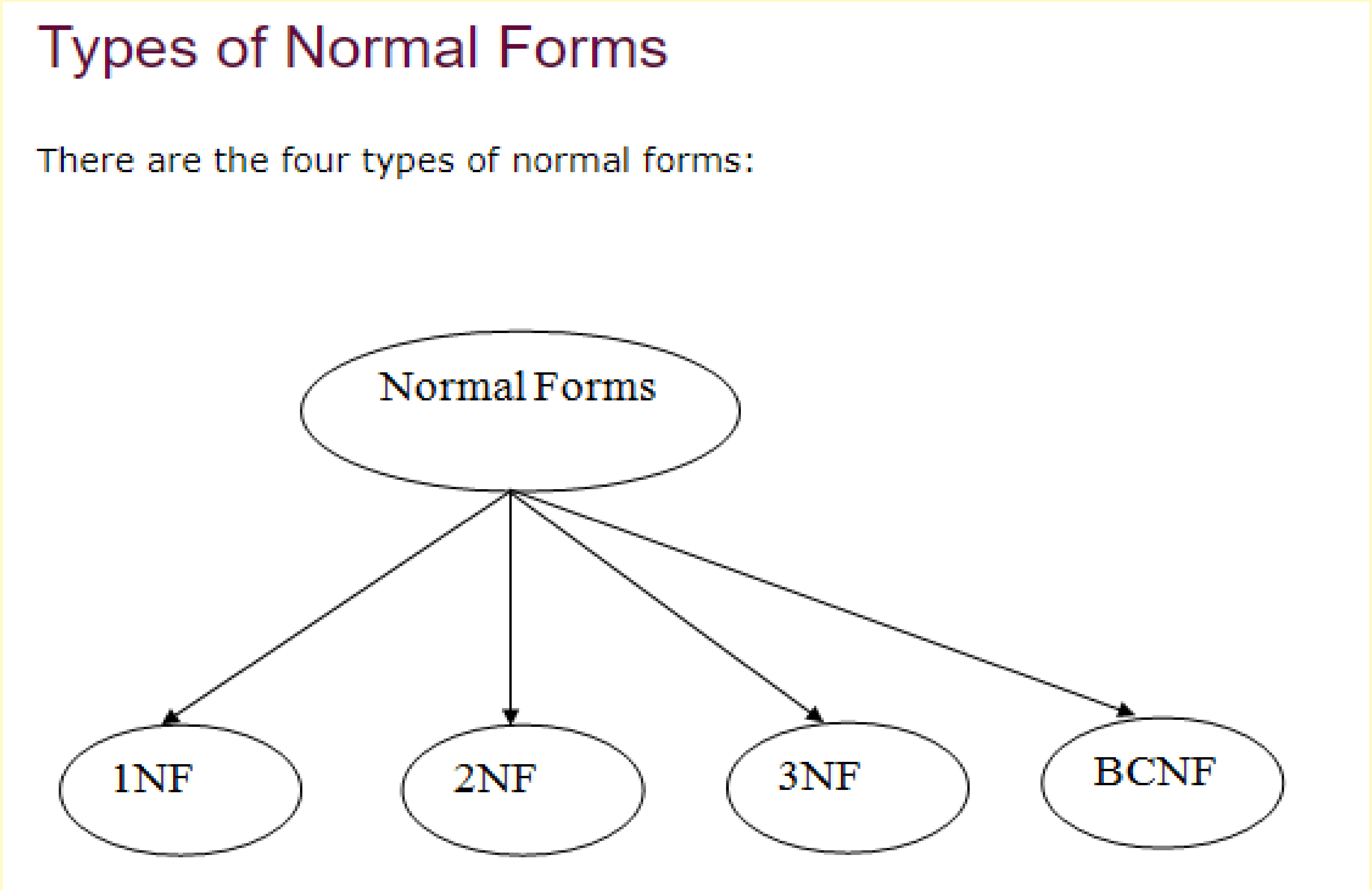
- The anomaly occurs when duplicate data is updated only in one place and not in all instances. Hence, it makes our data or table inconsistent state.

- For example, suppose there is a student 'James' who belongs to Student table. If we want to update the course in the Student, we need to update the same in the course table; otherwise, the data can be **inconsistent**. And it reflects the changes in a table with updated values where some of them will not.

- **4. Delete Anomalies:**
- An anomaly occurs in a database table when some records are lost or deleted from the database table due to the deletion of other records. For example, if we want to remove Trent Bolt from the Student table, it also removes his address, course and other details from the Student table. Therefore, we can say that deleting some attributes can remove other attributes of the database table.
- So, we need to avoid these types of anomalies from the tables and maintain the integrity, accuracy of the database table. Therefore, we use the normalization concept in the database management system.

Types of Normal Forms

There are the four types of normal forms:


Types of Normal Forms

There are the four types of normal forms:

- **Closure Set of an Attribute**

- It is used to find out how many attributes can be searched .
- (AB)+ ---closure set of AB

- With AB what other attributes we can search.
- This can be used for finding the candidate key which is very important to understand normalization.

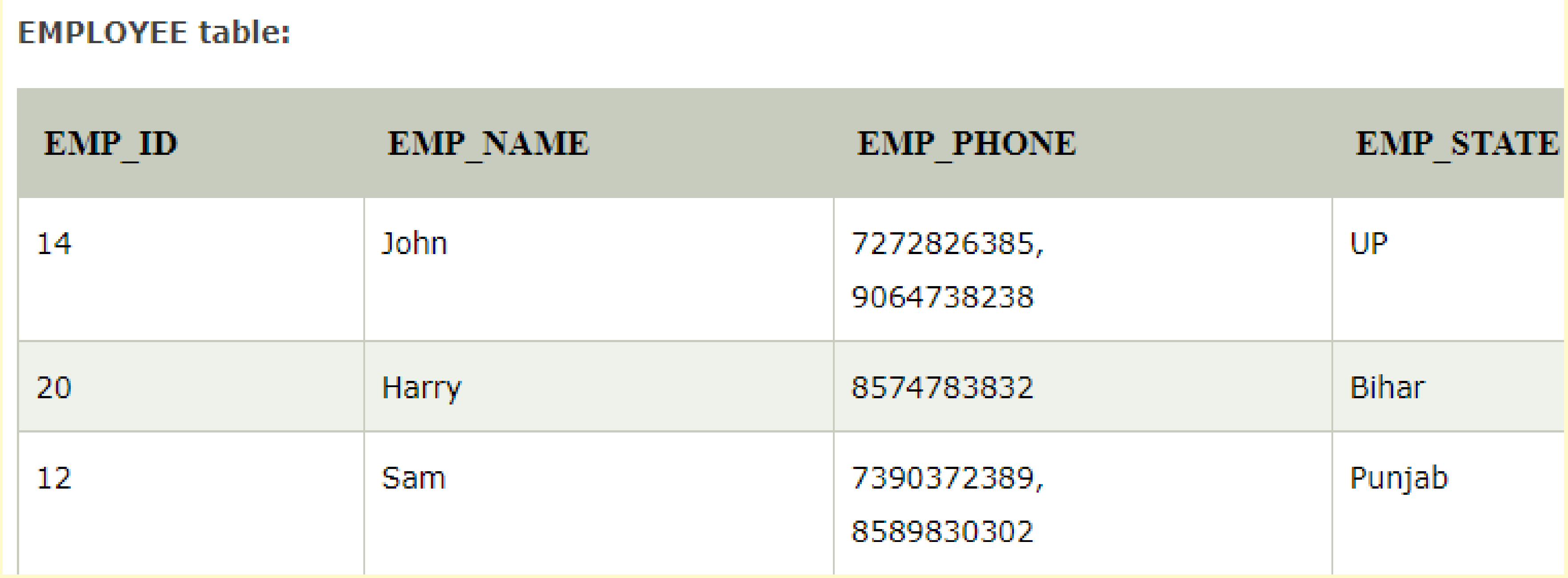
First Normal Form

- **Rules for First Normal Form**
- The first normal form expects you to follow a few simple rules while designing your database, and they are:
- Rule 1: Single Valued Attributes
- Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.
- Rule 2: Attribute Domain should not change
- This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

- **For example:** If you have a column dob to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

- Rule 3: Unique name for Attributes/Columns
- This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.
- If one or more columns have same name, then the DBMS system will be left confused.

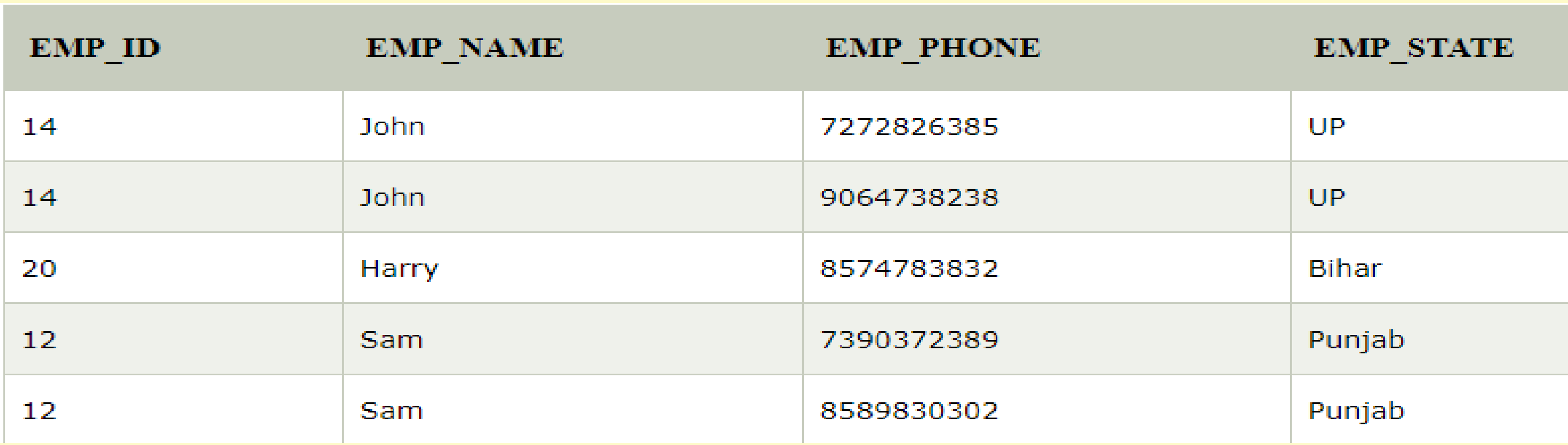
- Rule 4: Order doesn't matters
- This rule says that the order in which you store the data in your table doesn't matter.
-



EMPLOYEE table:

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

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



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

- **Second Normal Form (2NF)**

- In the 2NF, relation must be in 1NF.
- All non prime attributes should depend on whole of candidate key and not on partial key.
- If an attribute depends on only part of candidate key then it is partial dependency.
- In 2NF no partial dependency should exists

- **Example:**

- R(A B C D)
- AB->D
- B->C

- Find essential attribute and C.key
- (AB)=ABCD

- Prime attribute—A B
- Non Prime attribute --- C D

- Prime attribute—A B   Non Prime attribute --- C D
- **R(A B C D)**
- AB->D
- B->C

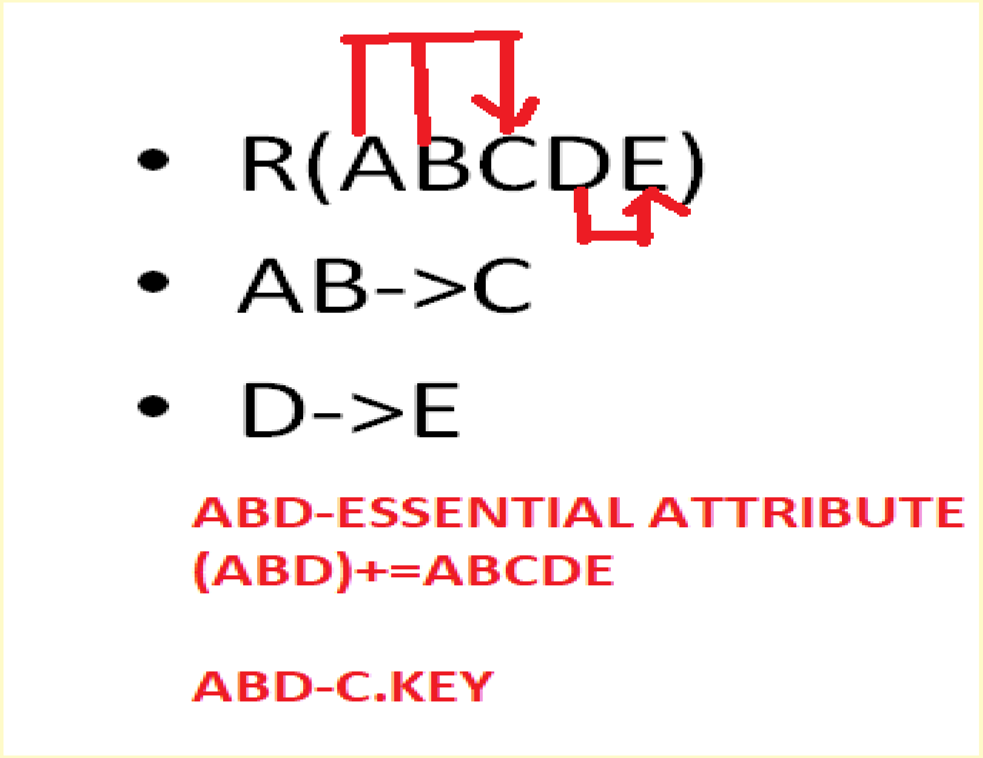
- AB->D   D depends on AB ---no issue
- B->C   C depends on only B and not AB, so it is partial dependency.
- So table is not in 2NF

- So get this in 2NF(decomposition)
- **R(A B C D)**
- AB->D
- B->C
-
- R1(A B D)   R2(B C)

- R(ABCDE)
- AB->C
- D->E

- AB->C IS PD
- D->E Is PD
- So not in 2NF

- So decompose



- R(ABCDE)
- AB->C
- D->E

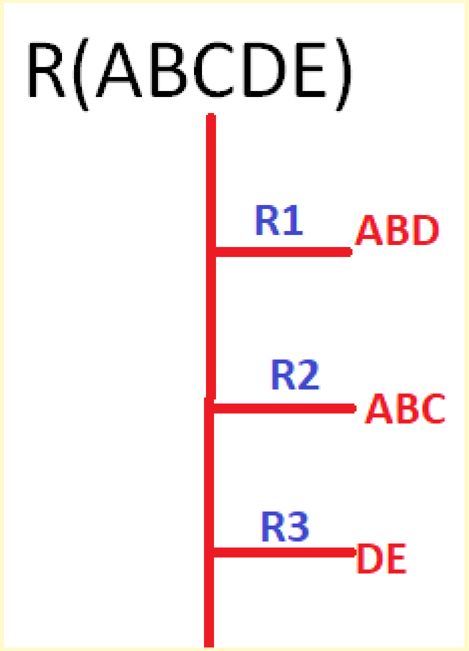
ABD-ESSENTIAL ATTRIBUTE
(ABD)+=ABCDE

ABD-C.KEY

R(ABCDE)

AB->C

D->E



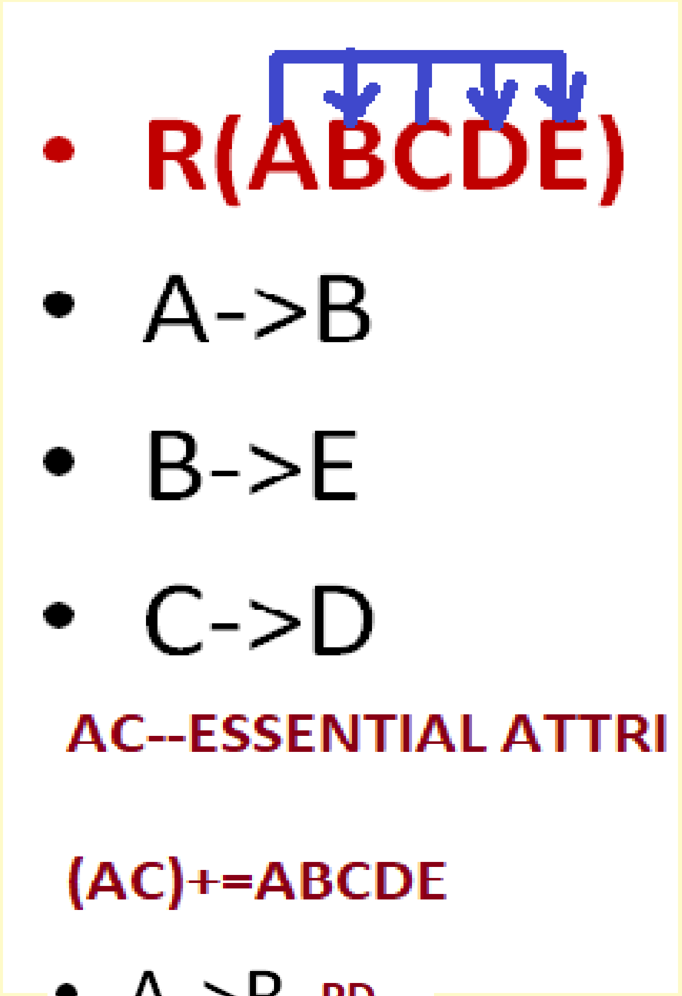
Try This

- **R(ABCDE)**
- A->B
- B->E
- C->D

AC ---PA

B DE--NPA

- NO NOT IN 2NF



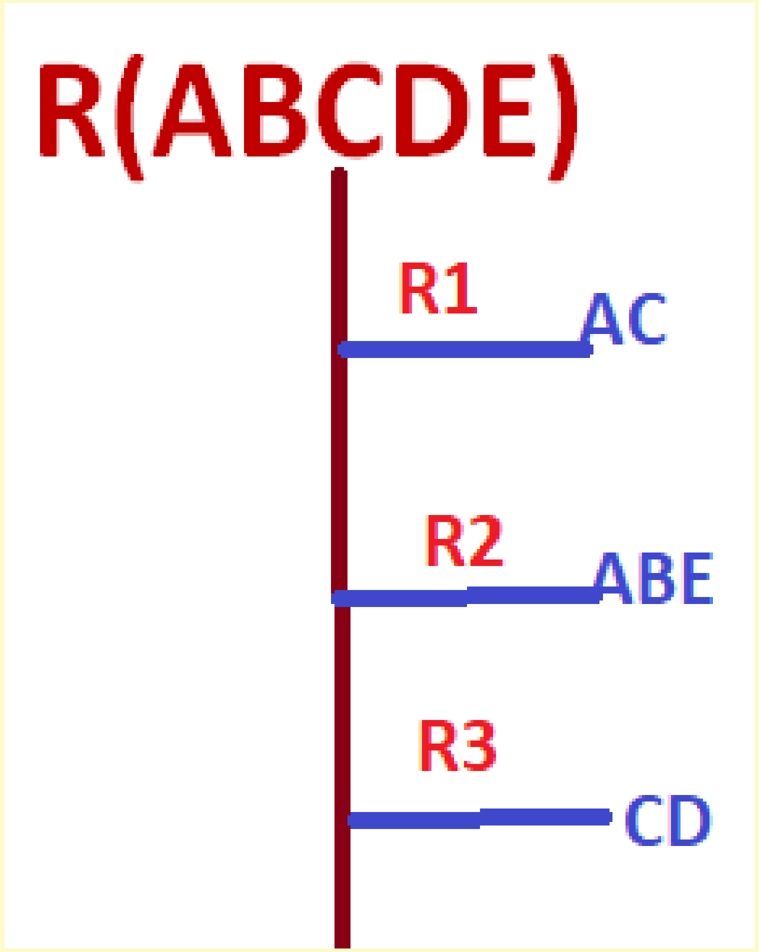
- **R(ABCDE)**
- A->B
- B->E
- C->D

AC--ESSENTIAL ATTRI

(AC)+=ABCDE

- A->B **PD**
- B->E
- C->D **PD**

Decomposition



R(ABCDEFGHIJ)

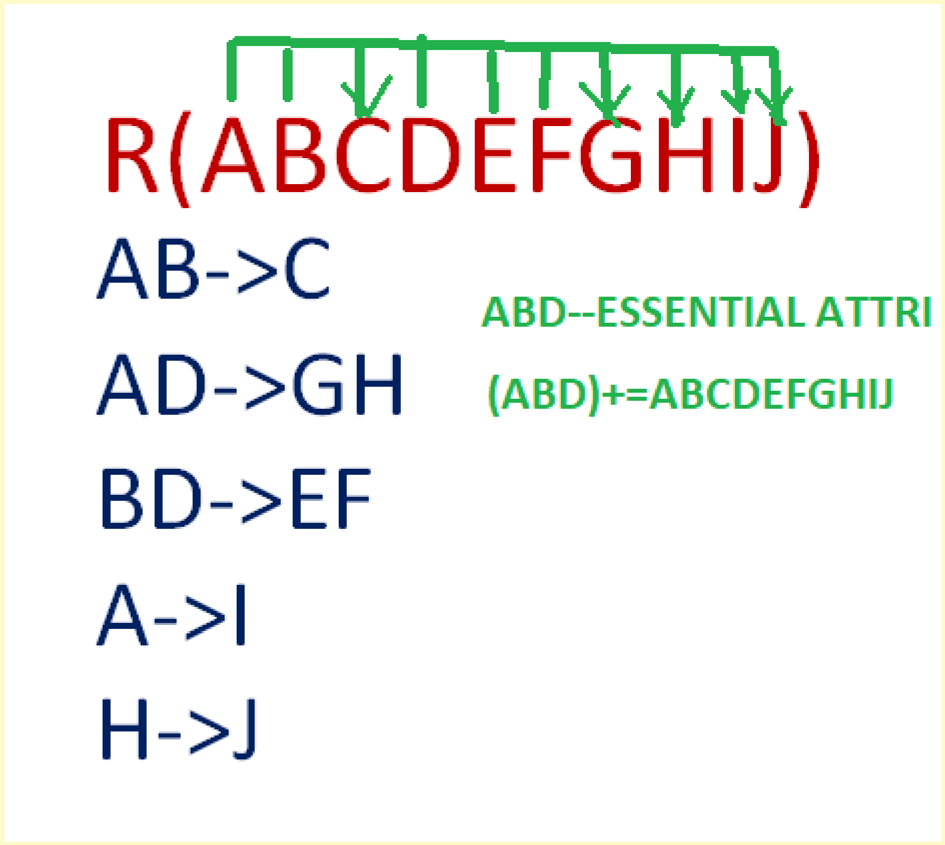
AB->C

AD->GH

BD->EF

A->I

H->J



R(ABCDEFGHIJ)

AB->C

AD->GH

BD->EF

A->I

H->J

ABD--ESSENTIAL ATTRI

(ABD)+=ABCDEFGHIJ

R(ABCDEFGHIJ)

AB->C PD

AD->GH PD

BD->EF PD

A->I PD

H->J PD

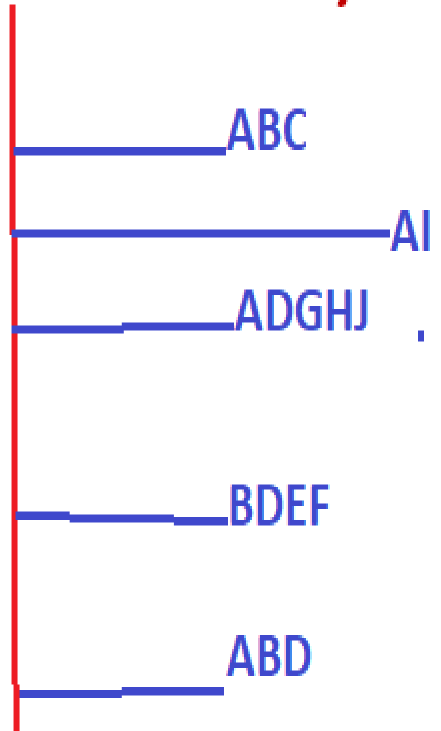
ABD--CKEY

PA-ABD NPA-CEFGHIJ

SO NOT IN 2NF

DECOMPOSITION

R(ABCDEFGHIJ)



Third Normal Form (3NF)

A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.

3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A Table is said to be in 3NF only when it is in 2NF and should not have any transitive dependency.

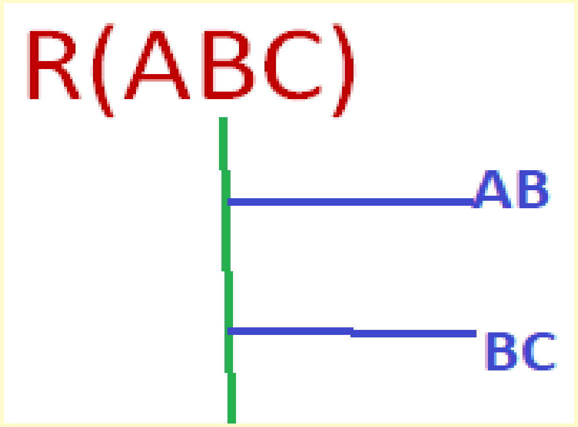
TD means a non prime attribute depending on non prime attribute(like an irregular stud depends on another irregular stud)

R(ABC)

A->B

B->C

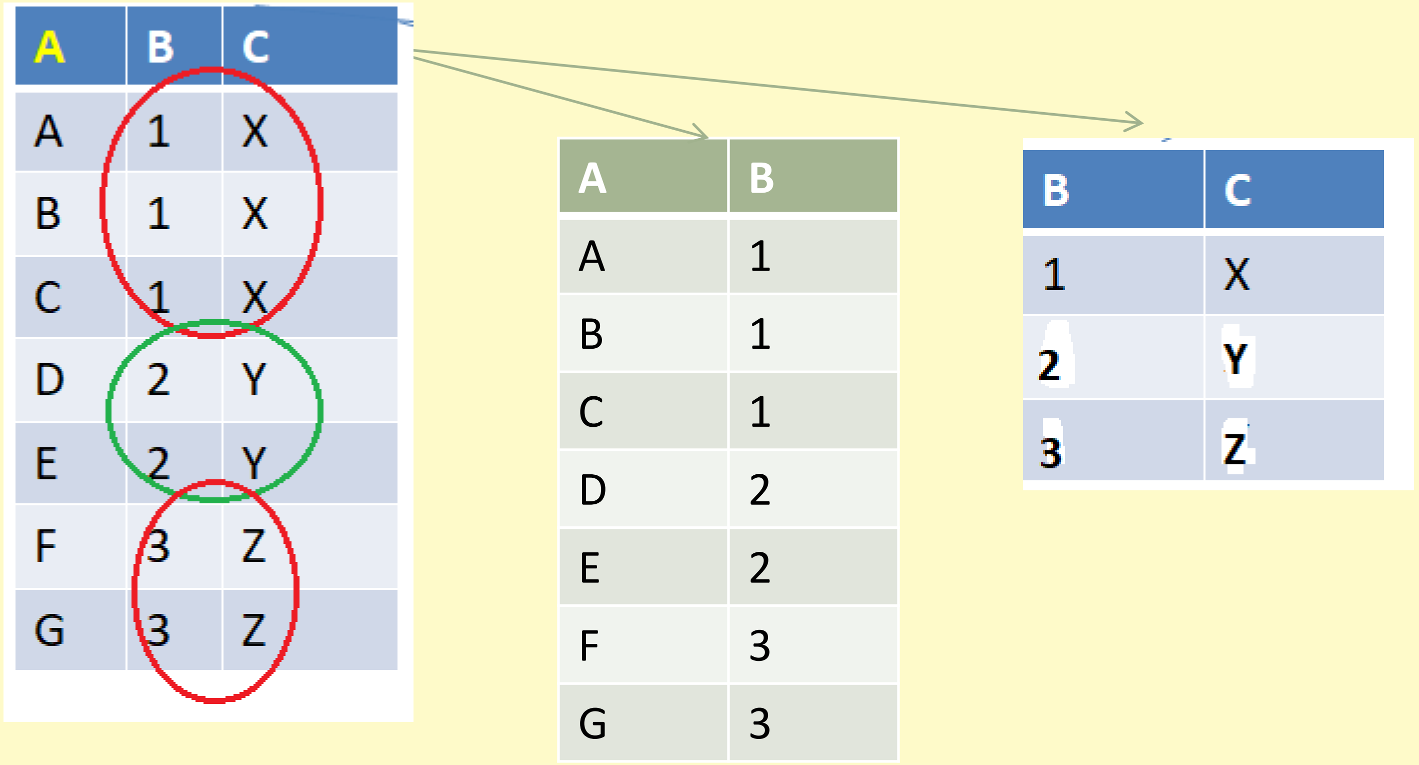
A- CKEY PA=A NPA=BC



R(ABC) CKEY =A

A->B PD NO

B->C PD NO BUT TD EXISTS



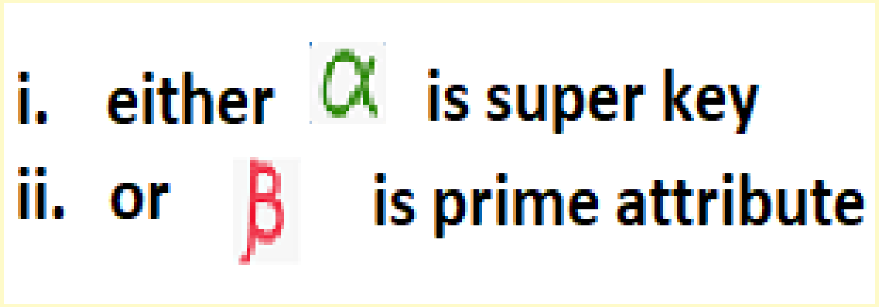
A	B	C
A	1	X
B	1	X
C	1	X
D	2	Y
E	2	Y
F	3	Z
G	3	Z

A	B
A	1
B	1
C	1
D	2
E	2
F	3
G	3

B	C
1	X
2	Y
3	Z

2nd Def of 3rd Normal form:

Every dependency from α to β



i. either α is super key
ii. or β is prime attribute

3 NF

R(ABCBDE)

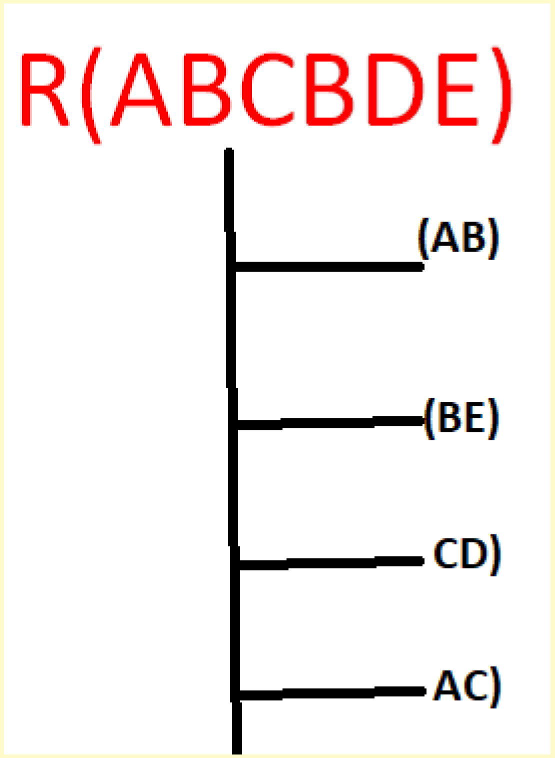
A->B

B->E

C->D

AC-ESSENTIAL

(AC)=ABCDE---C.KEY



3 NF

R(ABCBDEFGHIJ)

AB->C

A->DE

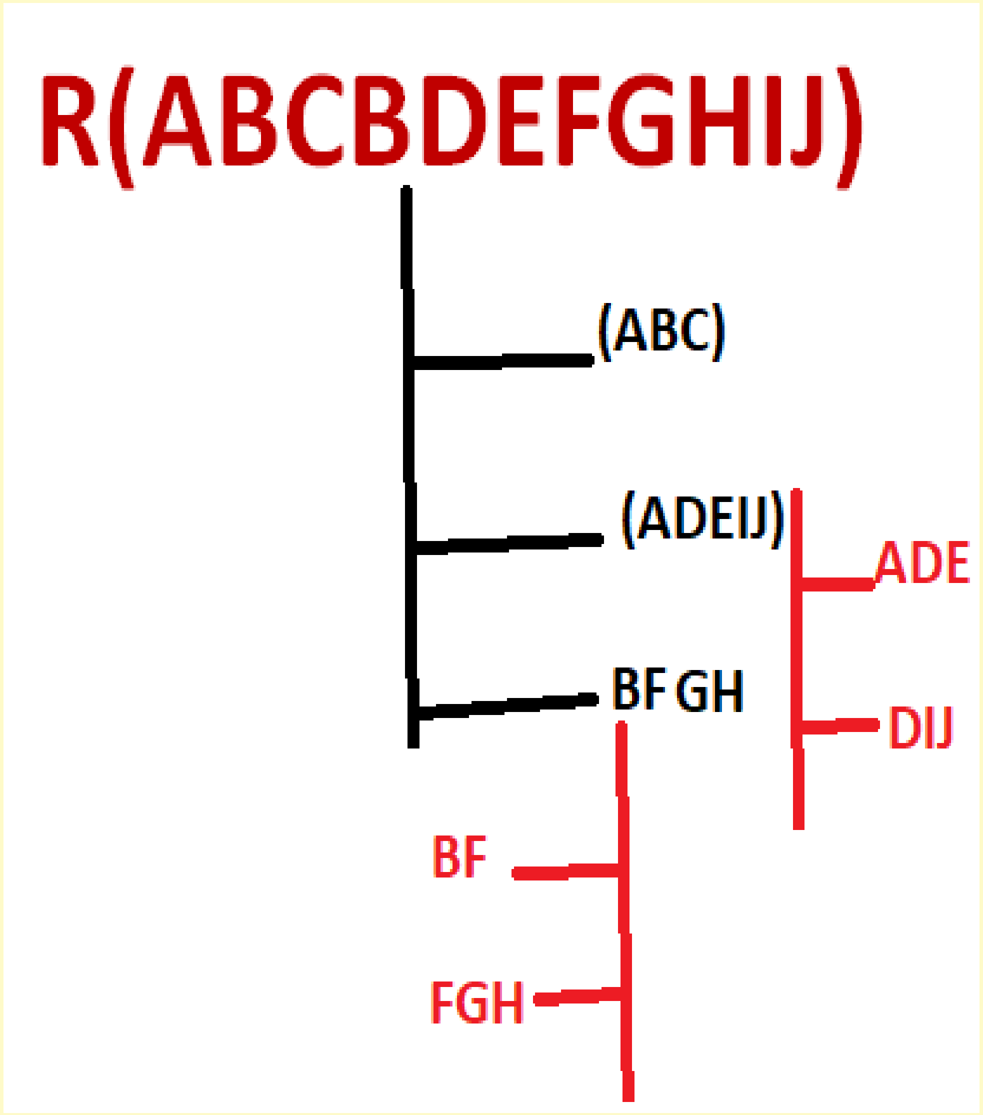
B->F

F->GH

D->IJ

AB-ESSENTIAL ATTRI

(AB)=ABCDEFGHIJ



R(ABCDE)

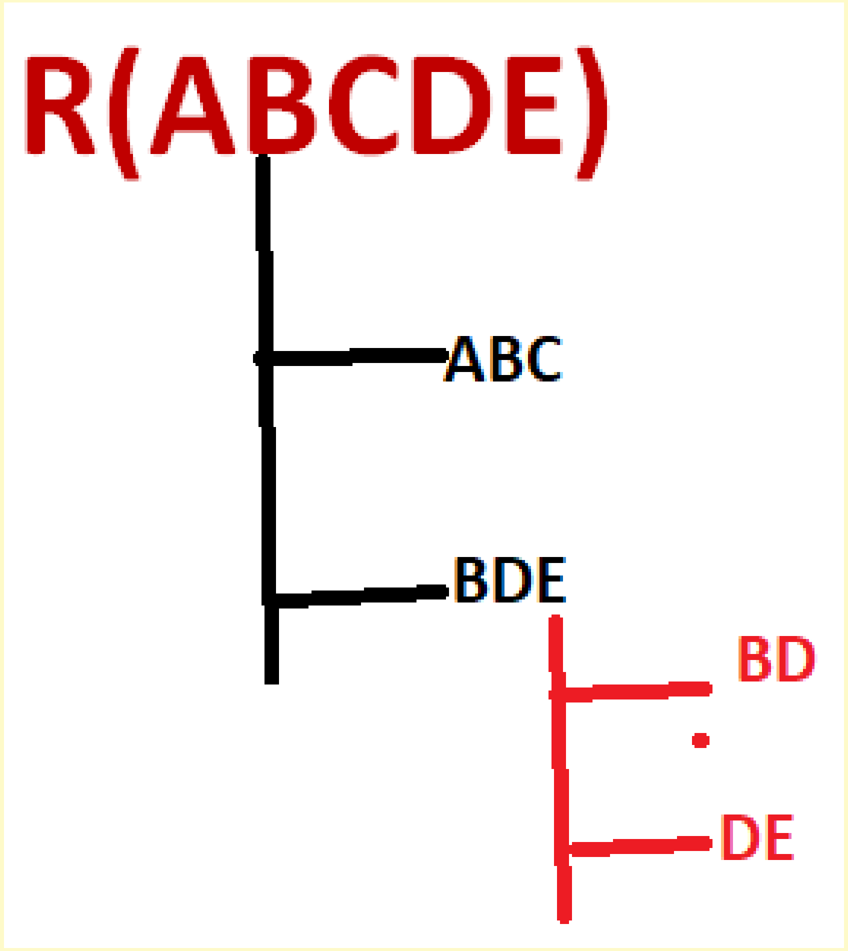
AB->C

B->D

D->E

AB-ESSENTIAL

(AB)-ABCDE



BCNF—BOYCE CODD NORMAL FORM

R(ABC)

AB->C
C->B

(A)=*
(AB)=ABC
(AC)=ABC

AB-> NO PD
C->B NO PD BCOS C-PRIME B-IS PRIME
SO IN 2NF

AB->C   --- NO TD

C->B----NO TD

If there exists FD from a->b

Then a should be super key and b can be anything

A prime attribute shud not depend on prime attribute

Decompose—R1(ab) R2(AC) R3(BC)

R(ABCDE)---CHECK THIS

AB->CD

D->A

BC->DE

R(ABCDE) -----CHECK THIS

BC->ADE

D->B

FUNCTIONAL DEPENDENCY

IF there exists FD from a->b

In the following cases

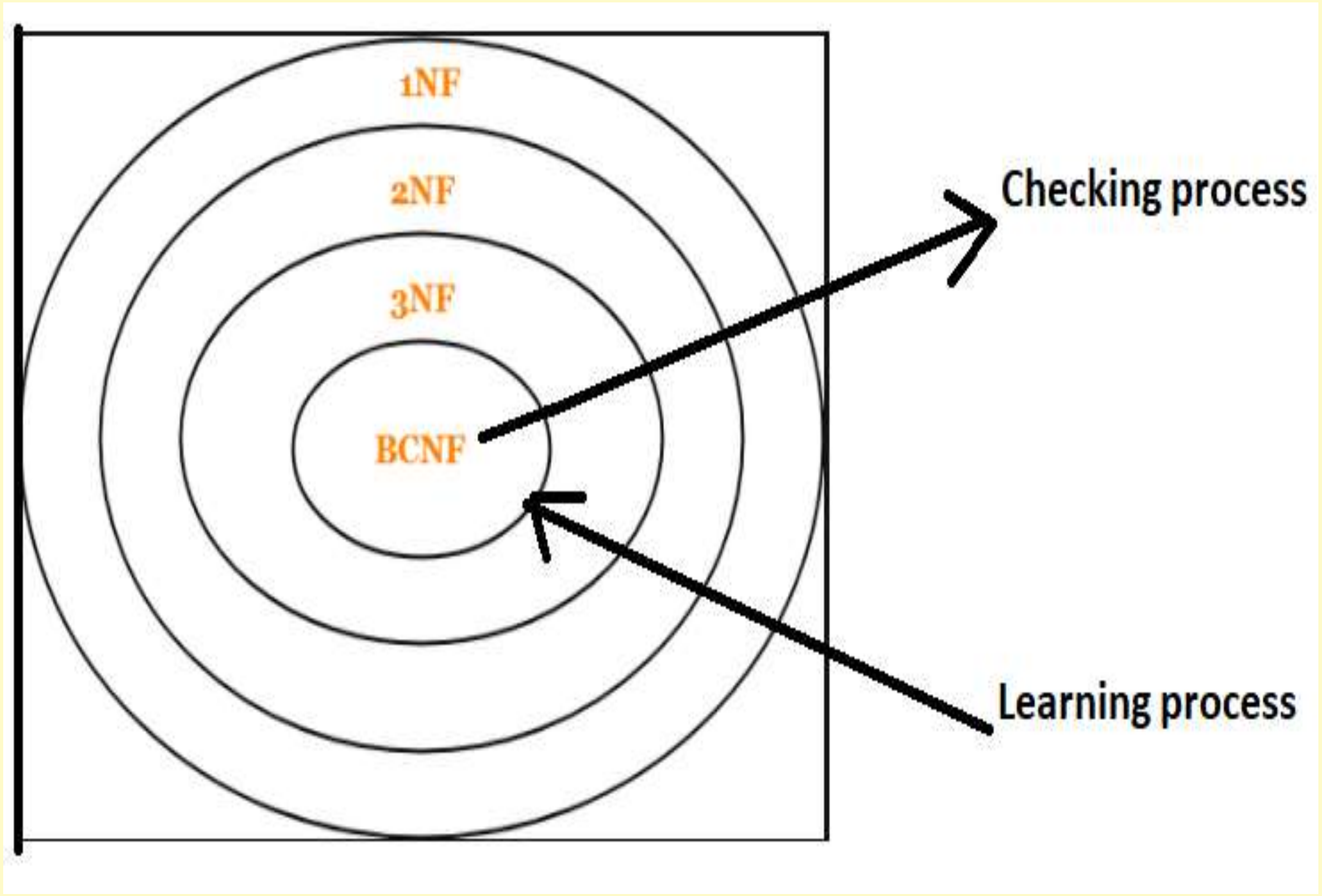
A	B
1	A
2	B
3	C
4	D
5	E
6	F

A	B
1	1
2	1
3	1
4	1
5	1
6	1

A	B
1	A
1	B
1	C
1	D
1	E
1	F

A	B
1	A
2	B
1	A
2	B
1	A
2	B

How to identify the Normal Form of a Relation



To solve the problem of identifying whether the Relation is in 1/2/3/BCNF we can start from BCNG then 3NF and so on. Exactly reverse of the way we learned them.

Lets understand how to identify whether the Relation is in

BCNF.

If there exists a FD from

α to β

then α should be super key/C.key

and β can be anything

2NF

In 2NF there should

be no partial dependency

3NF (No Transitive Dependency)

Either

if α is super key then BCNF and 3 NF . otherwise

if β is Prime Attribute

1)

R(A BC D E F G H)

AB->C

A->DE

B->F

F->GH

Essential Attributes: AB

(AB)+= ABCDEFGH

Candidate Key: AB

CHK BCNF

R(A BC D E F G H)

AB->C -- OK

A->DE -- NO   SO NOT IN BCNF

B->F

F->GH

CHK 3NF

R(A BC D E F G H)

AB->C -- OK

A->DE -- NO SO NOT IN 3NF

B->F

F->GH

CHK 2NF

R(A BC D E F G H)

AB->C -- OK

A->DE -- PD SO NOT IN 2NF

B->F

F->GH

SO IN 1NF

2 . R(A B C D E F)

AB->C
DC->AE
E->F

---

3. R(A B C D E)

CE->D
D->B
C->A

---

4. R(A B C D E F G H I)

AB->C
BD->EF
AD->GH
A->I

5. R(A B C D E)

AB->CD
D->A
BC->DE

6. R(A B C D E)

BC->ADE
D->B

7. R(V W X Y Z)

X->YV
Y->Z
Z->Y
VW->X

8. R(A B C D E F)

ABC->D

ABD->E

CD->F

CDF->B

BF->D

9. R(A B C)

A->B

B->C

C->A

Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key.

It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF).

It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

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

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

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

Here for certain values of course there are certain values for instructor and textbook, but there exists no relation between instructor and textbook. So it is called multi value dependency.

Course->-> instructor

Course ->-> textbook

COURSE	INSTRUCTOR	TEXTBOOK
MANAGEMENT	WHITE	ROBIN
	GREEN	PETER
	BLACK	
FINANCE	GRAY	WESTERN
		GILFORD

So decompose so that those multivalue dependency is removed as TEACHER RELATION AND TEXTBOOK RELATION

COURSE	INSTRUCTOR	TEXTBOOK
MANAGEMENT	**WHITE**	**ROBIN**
	GREEN	**PETER**
	BLACK	
FINANCE	**GRAY**	**WESTERN**
		GILFORD

COURSE	INSTRUCTOR
MANAGEMENT	WHITE
MANAGEMENT	GREEN
MANAGEMENT	BLACK
FINANCE	GRAY

COURSE	TEXTBOOK
MANAGEMENT	ROBIN
MANAGEMENT	PETER
FINANCE	WESTERN
FINANCE	GILFORD

Fifth normal form (5NF)

A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

5NF is also known as Project-join normal form (PJ/NF).

R Is decompose into R1 R2 R3

Then R1 join R2 or R1 join R3 or R2 join R3 should get back the original relation R

There should not any loss of data in terms of tuples.

Loss Less Join Decomposition

If we decompose a Relation R into R1 and R2 then when we join them again we should not have any loss of data , then it is termed as loss less decomposition.

R(ABCD) decomposed as R1(AB) and R2(D) then there is loss of data because we do not have C in R1 nor R2.

If a relation R is decomposed into 2 relations.

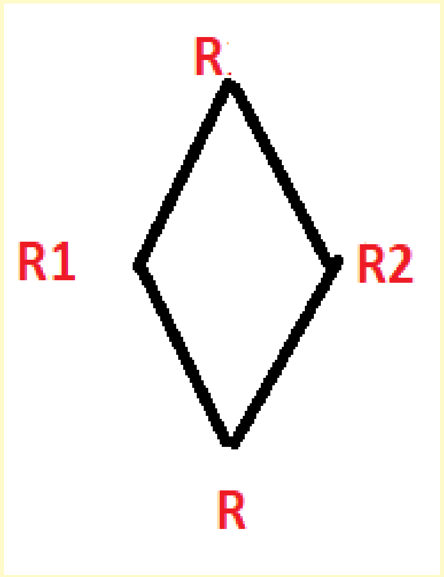
R1 & R2 then it will be lossless if

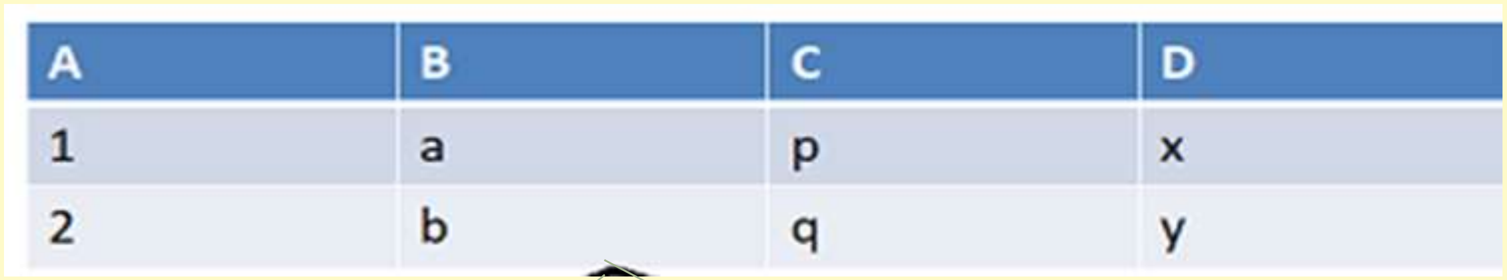
1) attribute (R1) U attribute (R2) = attribute (R)

2) attribute (R1) $\cap$ attribute(R2) <> NULL

3) attribute (R1) $\cap$ attribute (R2) → attribute(R1)

or attribute(R1) $\cap$ attribute(R2) → attribute(R2)



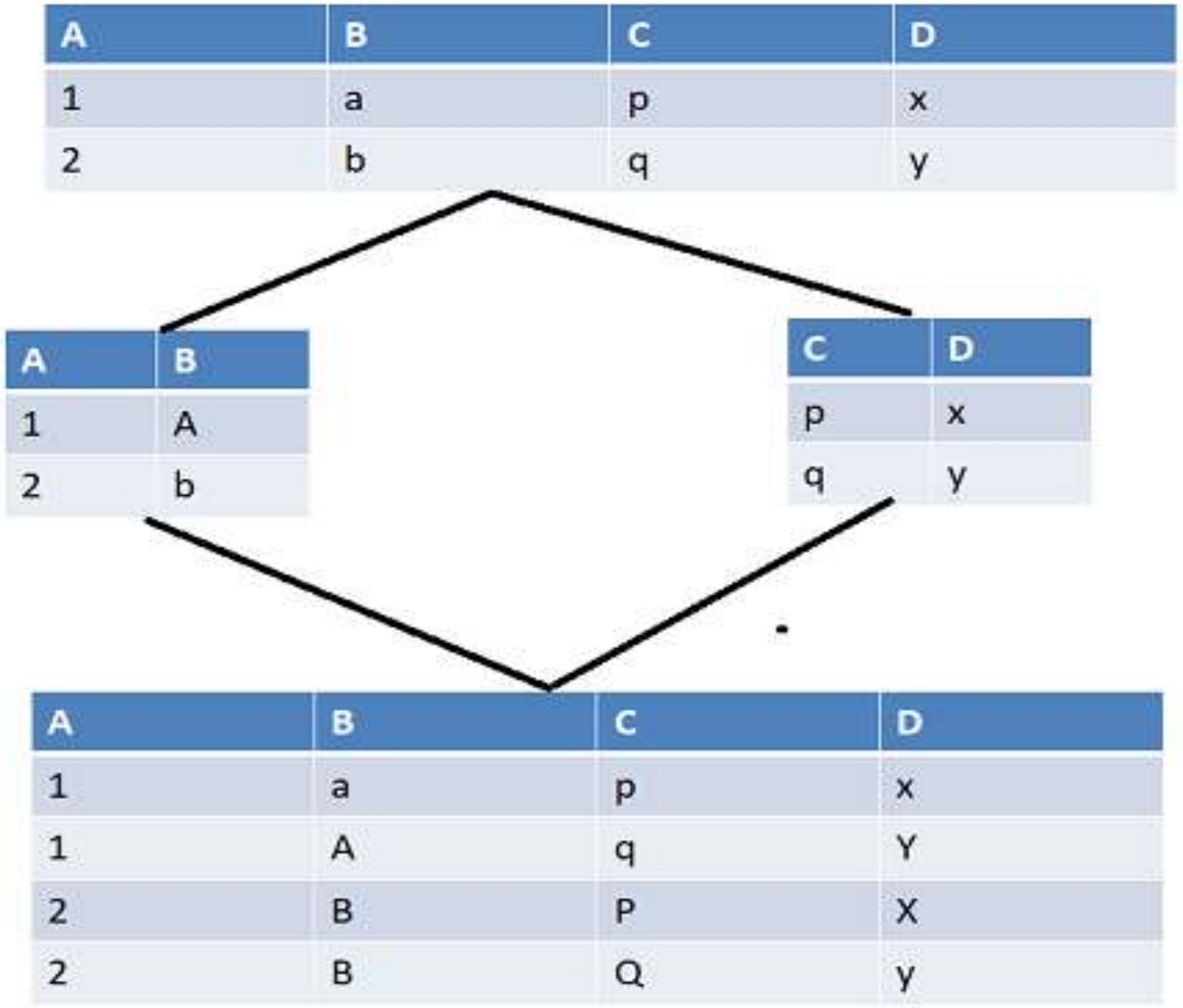


A	B	C	D
1	a	p	x
2	b	q	y

A	B
1	A
2	b

D
X
y

We are in loss of one column data ,so it is lossY decomposition.
Which is not allowed



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

A	B
1	A
2	b

C	D
p	x
q	y

A	B	C	D
1	a	p	x
1	A	q	Y
2	B	P	X
2	B	Q	y

A	B	C
1	a	P
2	B	Q
3	A	R

A	B
1	A
2	B
3	A

B	C
A	P
B	Q
A	r

A	B	C
1	A	P
1	A	R
2	B	Q
3	A	P
3	A	R

R

A	B	C	D	E
A	122	1	P	W
B	234	2	Q	X
A	568	1	R	Y
C	347	3	S	Z

R1(ABC) R2(BCD)   R3(DE) --RIGHT

UNIT-4

Transaction Management

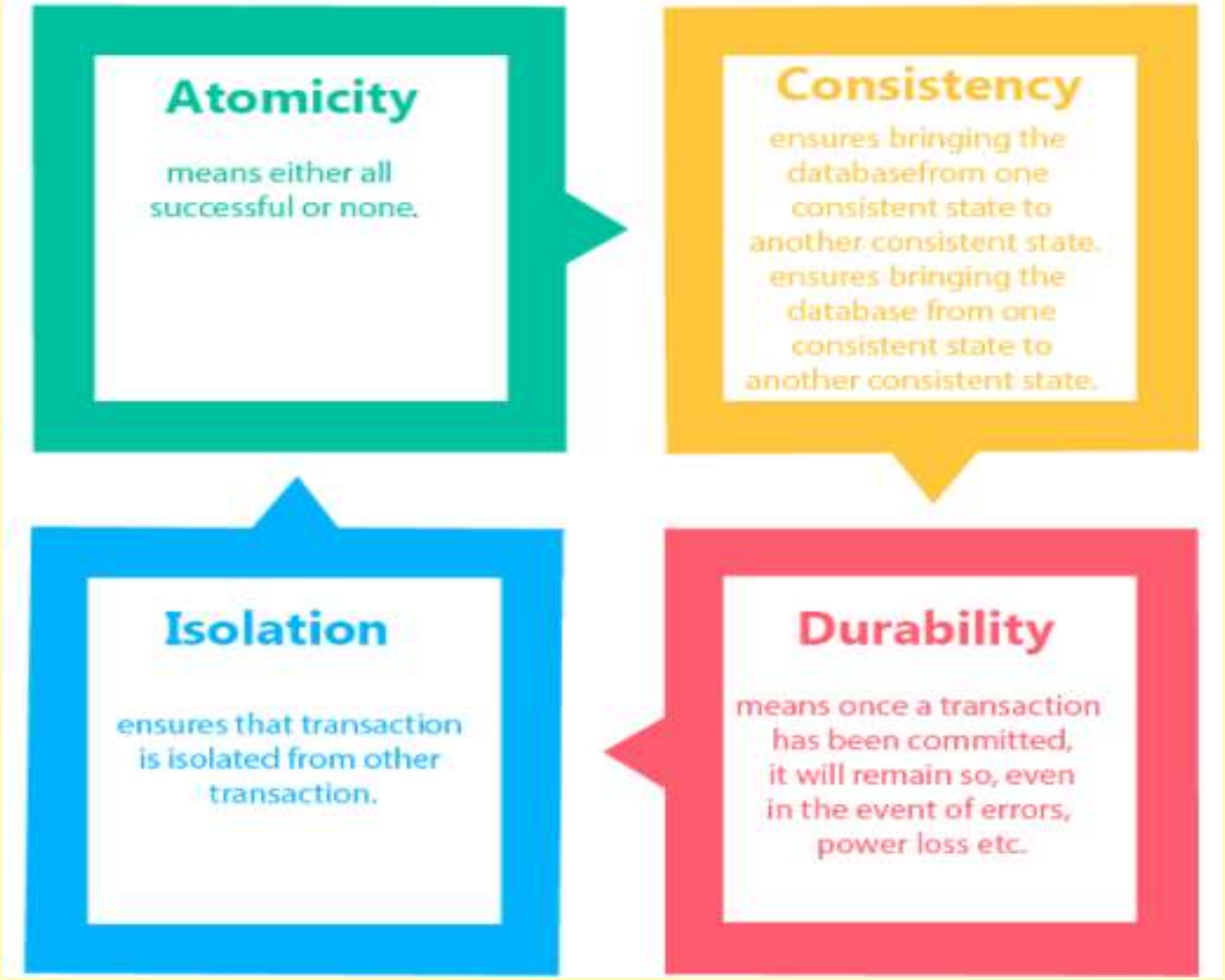
A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of RDBMS-oriented enterprise application to ensure data integrity and consistency. The concept of transactions can be described with the following four key properties described as **ACID** −

Atomicity − A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.

Consistency − This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.

Isolation − There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.

Durability − Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

- Atomicity
- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.
- Atomicity involves the following two operations:

- **Abort:** If a transaction aborts then all the changes made are not visible.

- **Commit:** If a transaction commits then all the changes made are visible.
- **Example:** Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.
- T1 T2

- Read(A)
-

  A:= A-100
-

  Write(A)
- Read(B)
-

  Y:= Y+100

  Write(B)After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

The integrity constraints are maintained so that the database is consistent before and after the transaction.

The execution of a transaction will leave a database in either its prior stable state or a new stable state.

The consistent property of database states that every transaction sees a consistent database instance.

The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.
Total before T occurs = 600+300=900
Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.

In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.

The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.

They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
The recovery subsystem of the DBMS has the responsibility of Durability property.

UNIT-4

Transactions in DBMS :

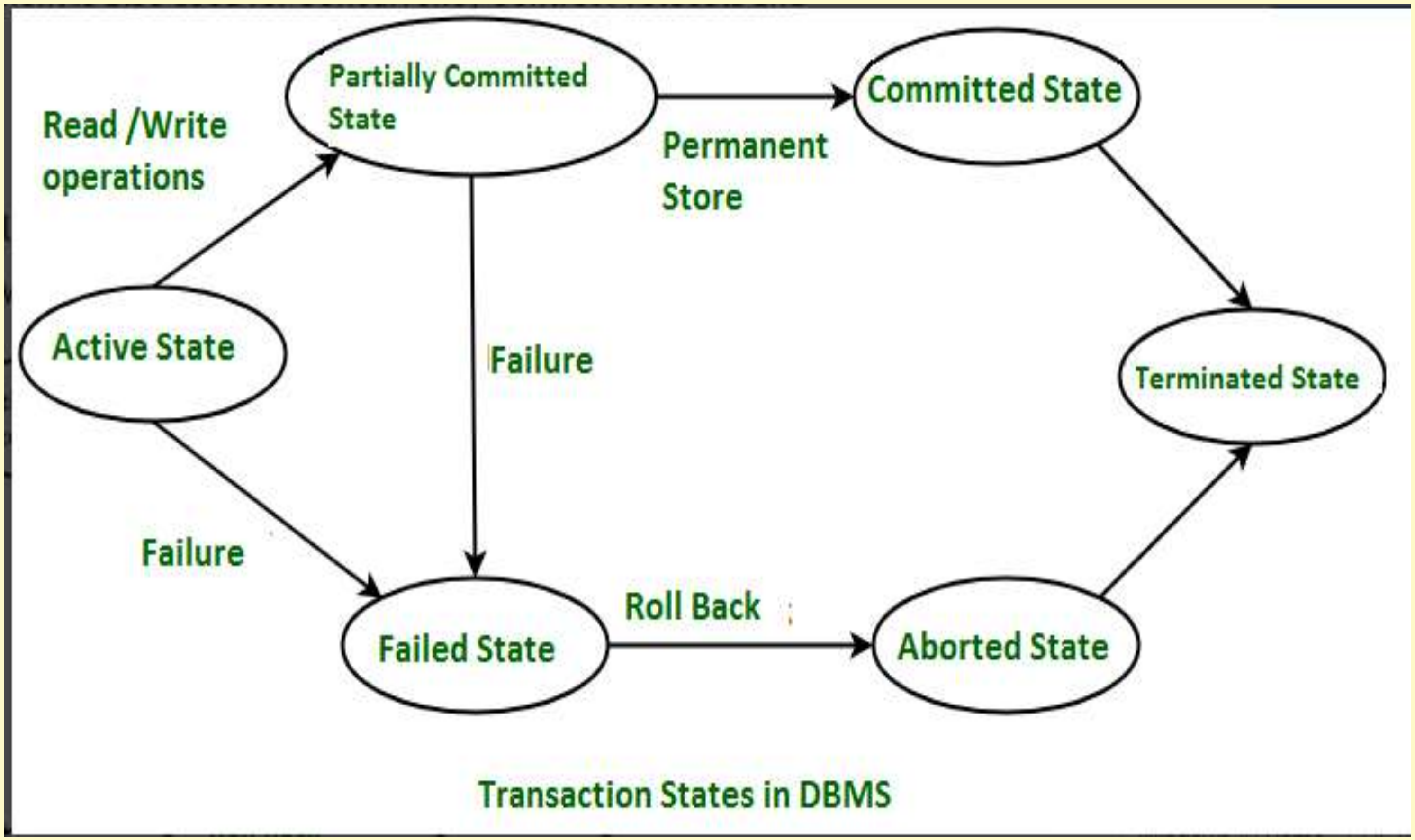
Transactions are a set of operations used to perform a logical set of work. A transaction usually means that the data in the database has changed. One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.

- **Example –**
  Transaction to be performed to withdraw cash from an ATM vestibule.
- **Set of Operations :**
  Consider the following example for transaction operations as follows.
- **Example -ATM transaction steps.**
- Transaction Start.
- Insert your ATM card.
- Select language for your transaction.
- Select Savings Account option.
- Enter the amount you want to withdraw.
- Enter your secret pin.
- Wait for some time for processing.
- Collect your Cash.
- Trasaction Completed.

- Three operations can be performed in a transaction as follows.
-
- Read/Access data (R).
- Write/Change data (W).
- Commit.
- **Example –**
  Transfer of 50₹ from Account A to Account B.
- Initially A= 500₹, B= 800₹.
- This data is brought to RAM from Hard Disk.
- R(A) -- 500 // Accessed from RAM.
- A = A-50 // Deducting 50₹ from A.
- W(A)--450 // Updated in RAM.
- R(B) -- 800 // Accessed from RAM.
- B=B+50 // 50₹ is added to B's Account.
- W(B) --850 // Updated in RAM.
- commit // The data in RAM is taken back to Hard Disk.

- **Note –**
  The updated value of Account A = 450₹ and Account B = 850₹.
- All instructions before commit come under a partially committed state and are stored in RAM. When the commit is read the data is fully accepted and is stored in Hard Disk.
- If the data is failed anywhere before commit we have to go back and start from the beginning. We can't continue from the same state. This is known as Roll Back.
- **Uses of Transaction Management :**
- The DBMS is used to schedule the access of data concurrently. It means that the user can access multiple data from the database without being interfered with each other. Transactions are used to manage concurrency.
- It is also used to satisfy ACID properties.
- It is used to solve Read/Write Conflict.
- It is used to implement Recoverability, Serializability, and Cascading.
- Transaction Management is also used for Concurrency Control Protocols and Locking of data.

- Transaction States :
  Transactions can be implemented using SQL queries and Server. In the below-given diagram, you can see how transaction states works.

- 

Transaction States in DBMS

Active state
The active state is the first state of every transaction. In this state, the transaction is being executed.

For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed
In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.

In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed
A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.

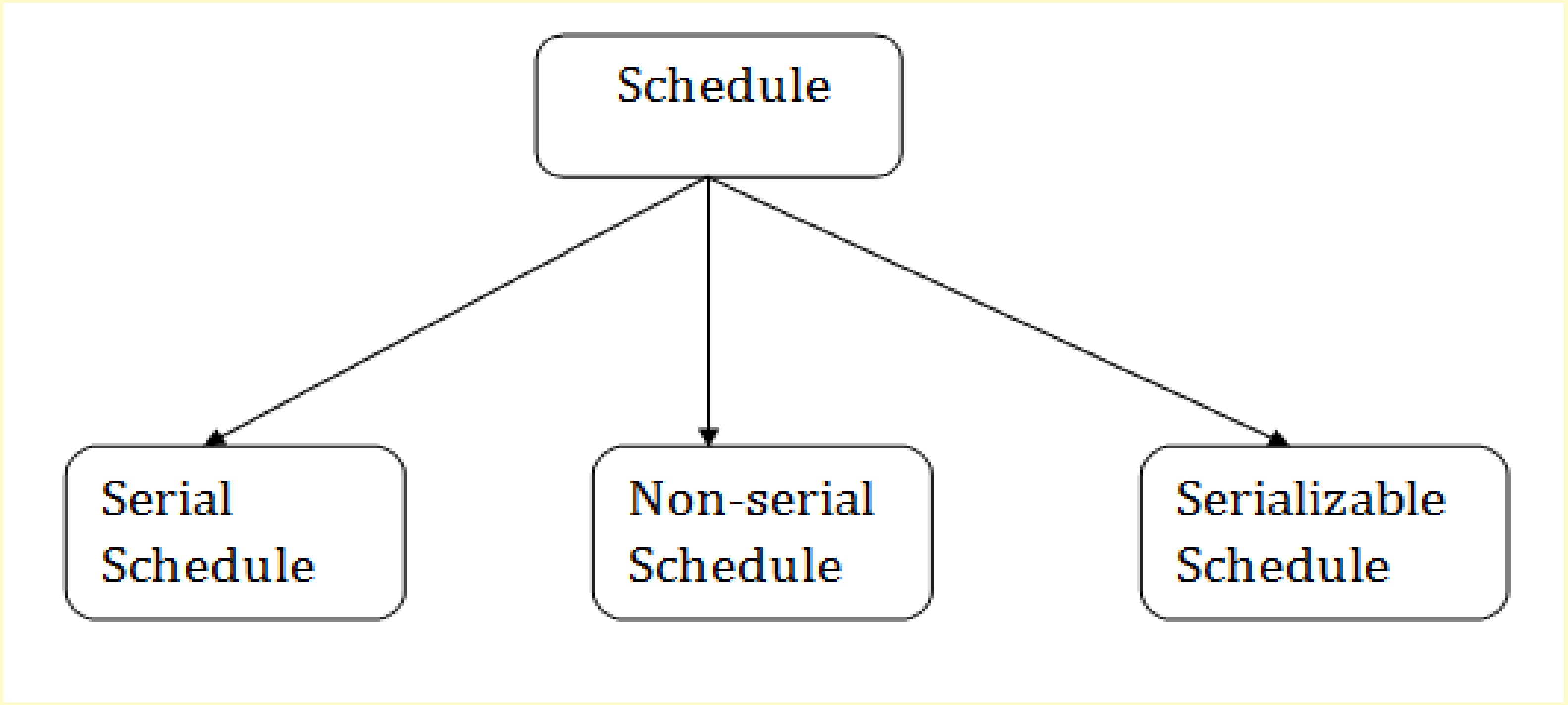
If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.

After aborting the transaction, the database recovery module will select one of the two operations:

Re-start the transaction
Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

Execute all the operations of T1 which was followed by all the operations of T2.

In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.

In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

If interleaving of operations is allowed, then there will be non-serial schedule.

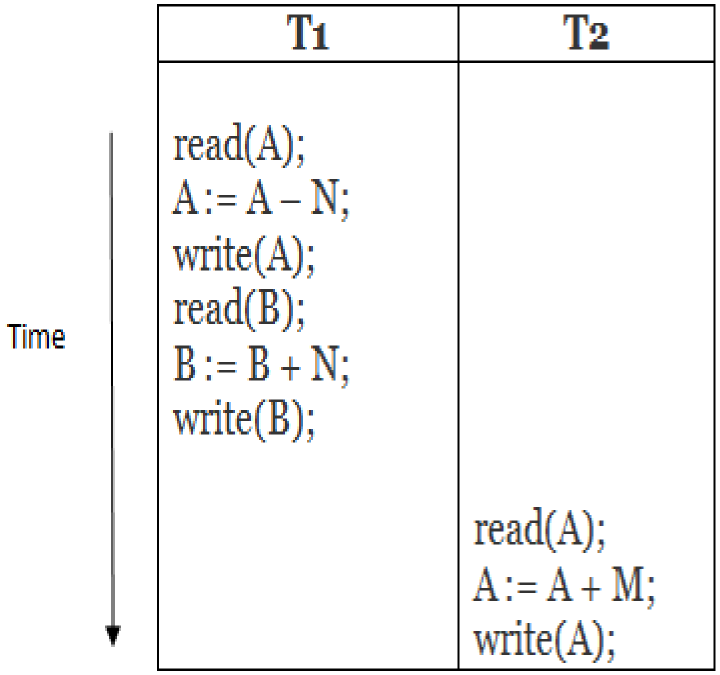
It contains many possible orders in which the system can execute the individual operations of the transactions.

In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. Serializable schedule

The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

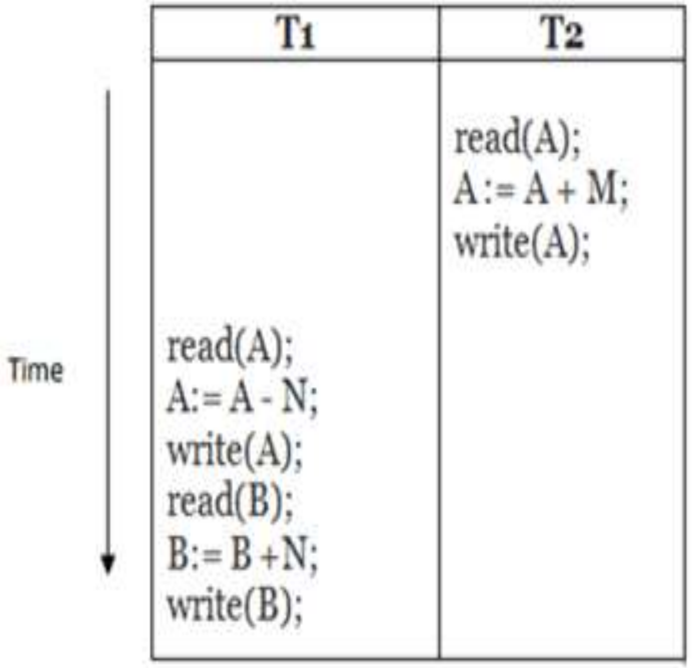
(a)



Time	T1	T2
↓	read(A);	
	A := A – N;	
	write(A);	
	read(B);	
	B := B + N;	
	write(B);	
		read(A);
		A := A + M;
		write(A);

Schedule A

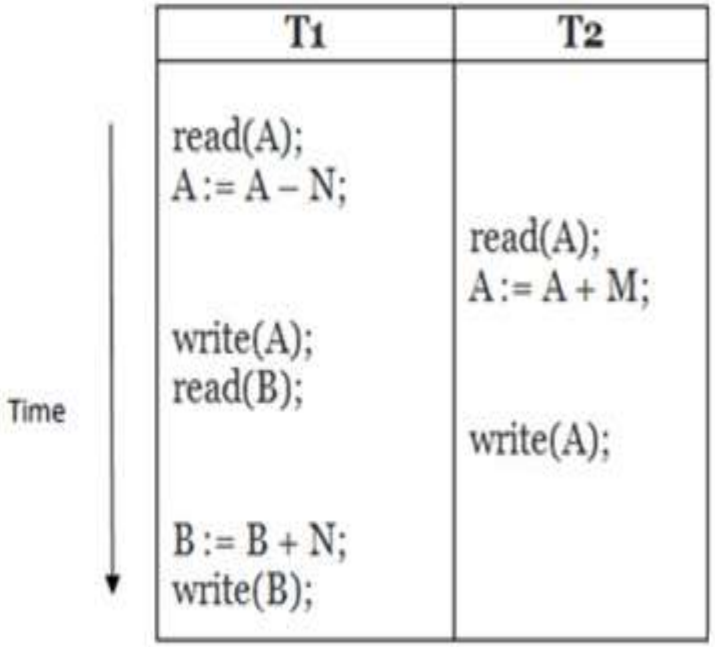
(b)



Time	T1	T2
↓		read(A);
		A:= A + M;
		write(A);
	read(A);	
	A:= A - N;	
	write(A);	
	read(B);	
	B:= B +N;	
	write(B);	

Schedule B

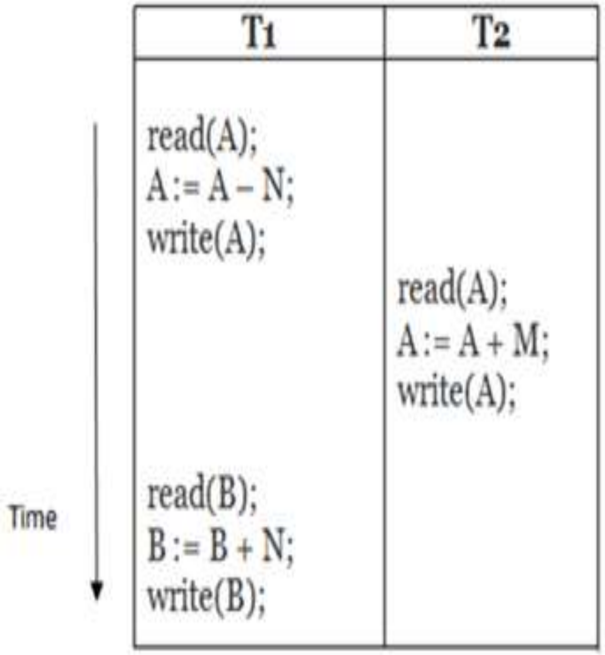
(c)



T1	T2
read(A); A := A − N;	
	read(A); A := A + M;
write(A); read(B);	
	write(A);
B := B + N; write(B);	

Schedule C

(d)



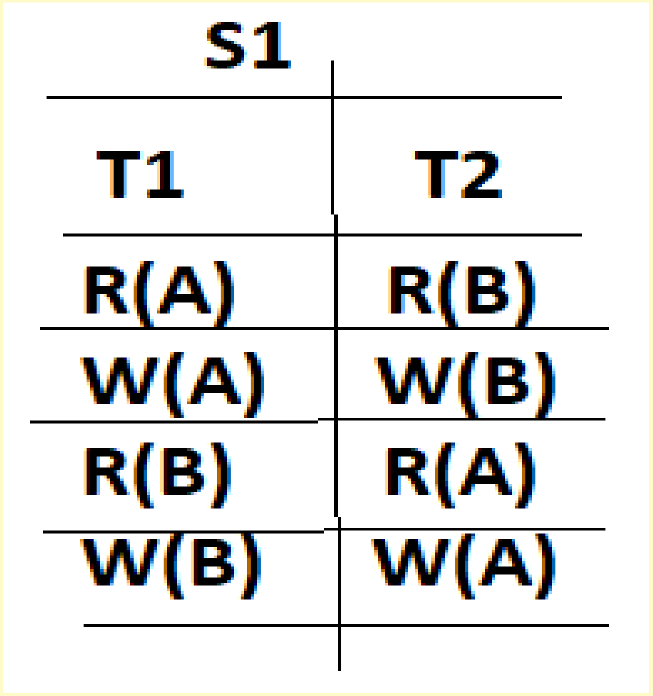
T1	T2
read(A); A := A − N; write(A);	
	read(A); A := A + M; write(A);
read(B); B := B + N; write(B);	

Schedule D

Here,

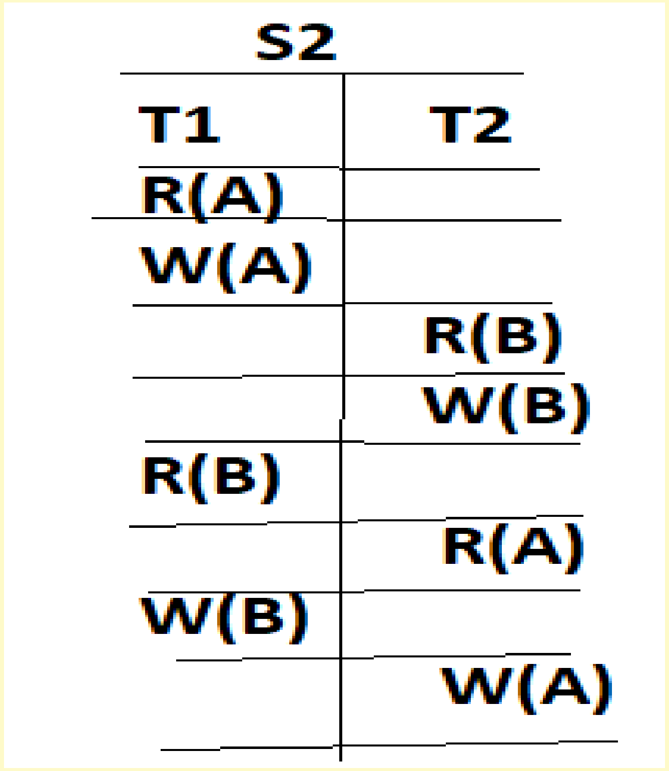
Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.



S1

T1	T2
R(A)	R(B)
W(A)	W(B)
R(B)	R(A)
W(B)	W(A)



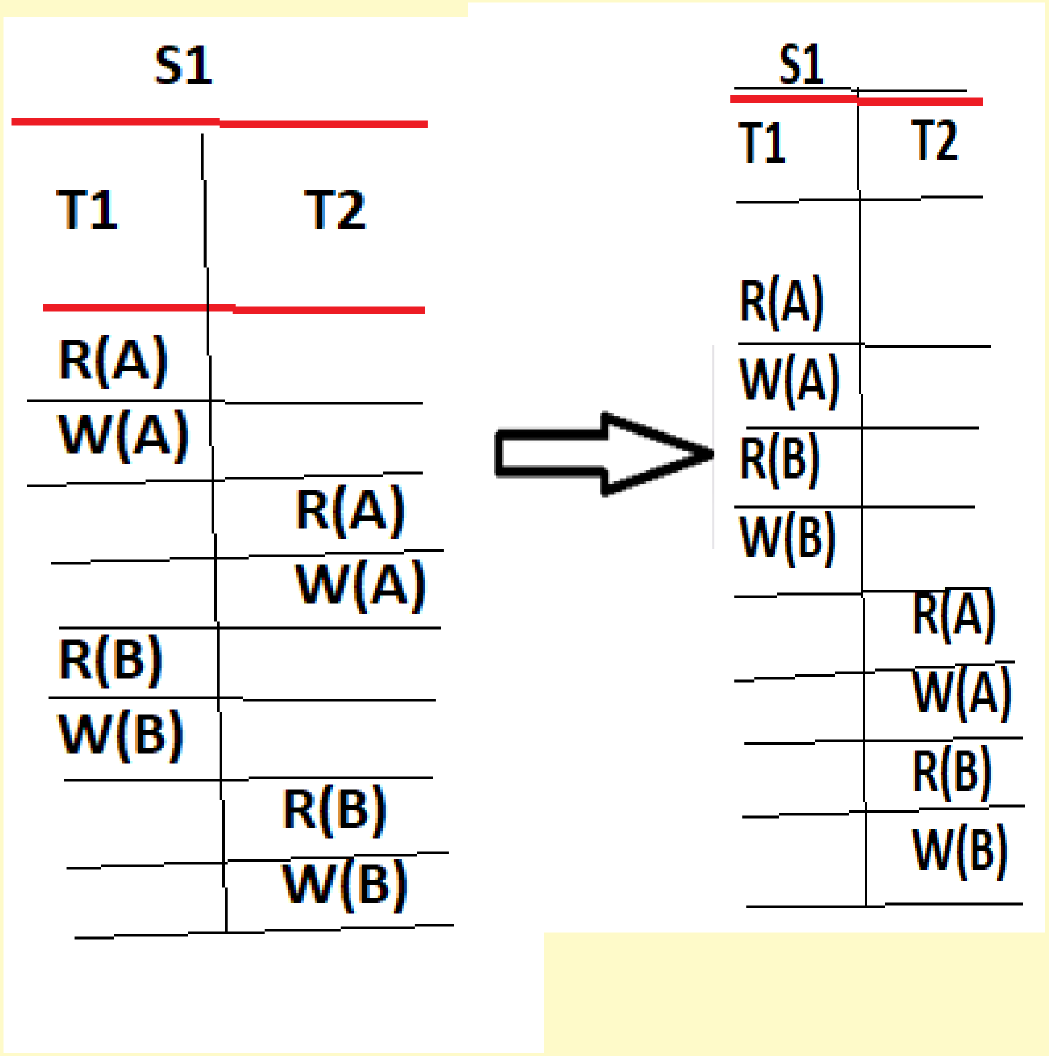
S2

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(B)	
	R(A)
W(B)	
	W(A)

We cannot change sequence of statement execution , but we can go for context switching as shown above. No multiple instructions are not executed, bcos processor do not execute multiple execution at the same time. Does processor do more than one task?

Serial scheduling when no context switching

Non serial scheduling when context switching happens



S1

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

S1

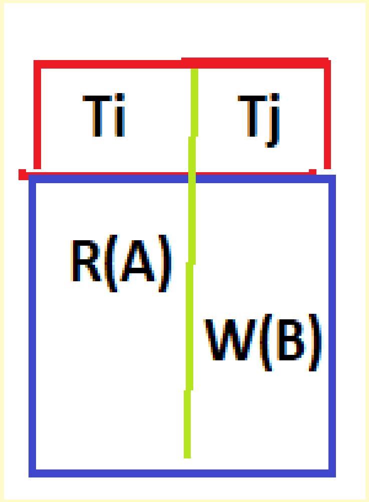
T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

If we can convert a non serial schedule to a serial schedule then we can say it is consistent schedule.
But if we cannot convert a non serial to serial schedule it does not mean it is inconsistent
It is not guarantee that a student not passing EAMCET is poor ….

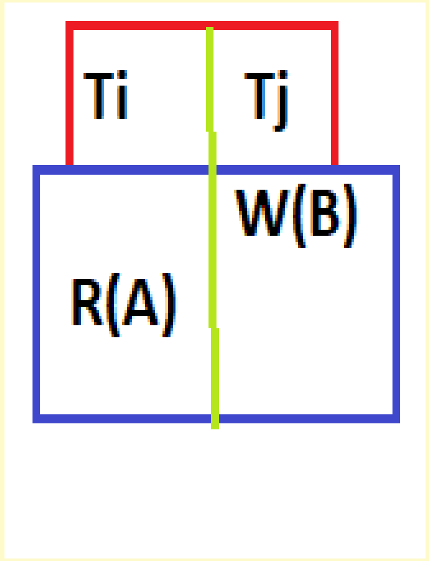
- If we can swap instructions in 2 transactions then we call them as non conflicting instructions otherwise called as conflicting instructions.

- So we can swap instructions of non serial schedule to make it serial schedule by swapping of non conflicting instructions.

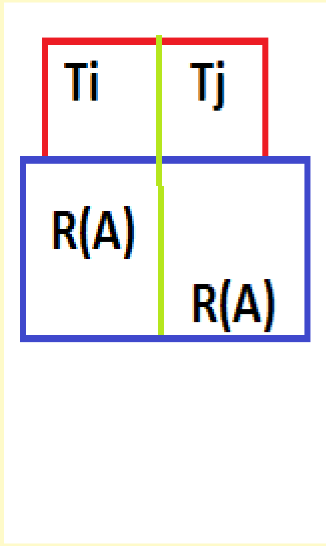
- First we should find out when an instruction is conflicting and when an instruction is non conflicting.



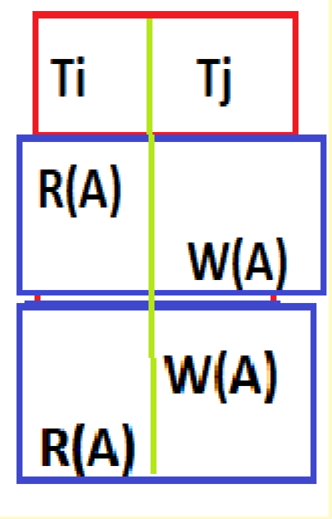
No prob



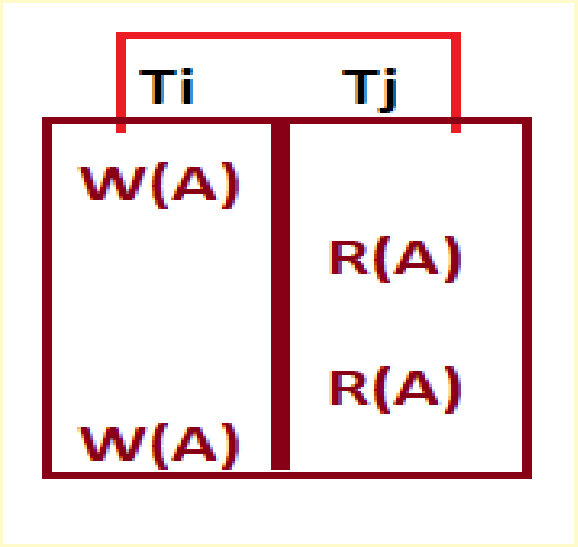
No prob



No Prob

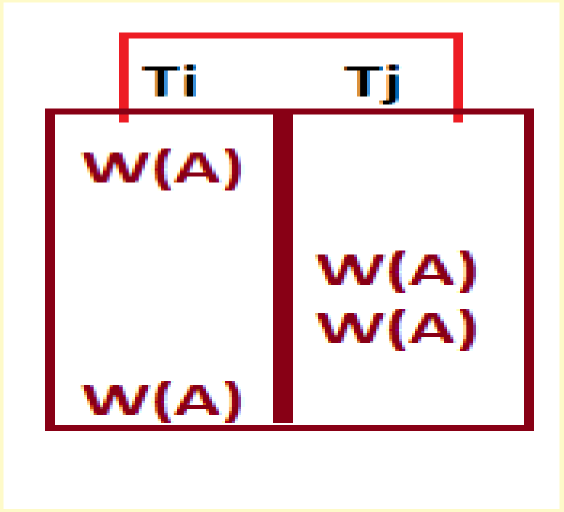


Problem



Problem here also

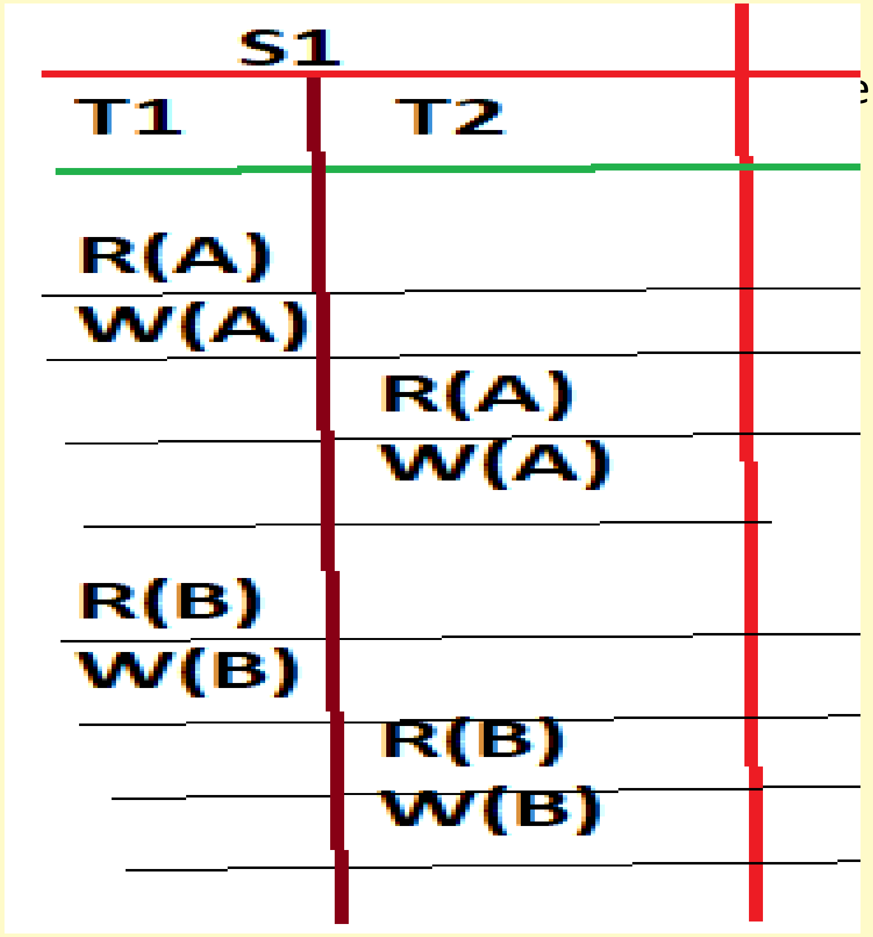
If problem exists means the statements are conflicting so no swapping can happen



If write and write happens and it is called blind write ,because they are not reading. But this is also conflicting bcos the final value written in database is different.

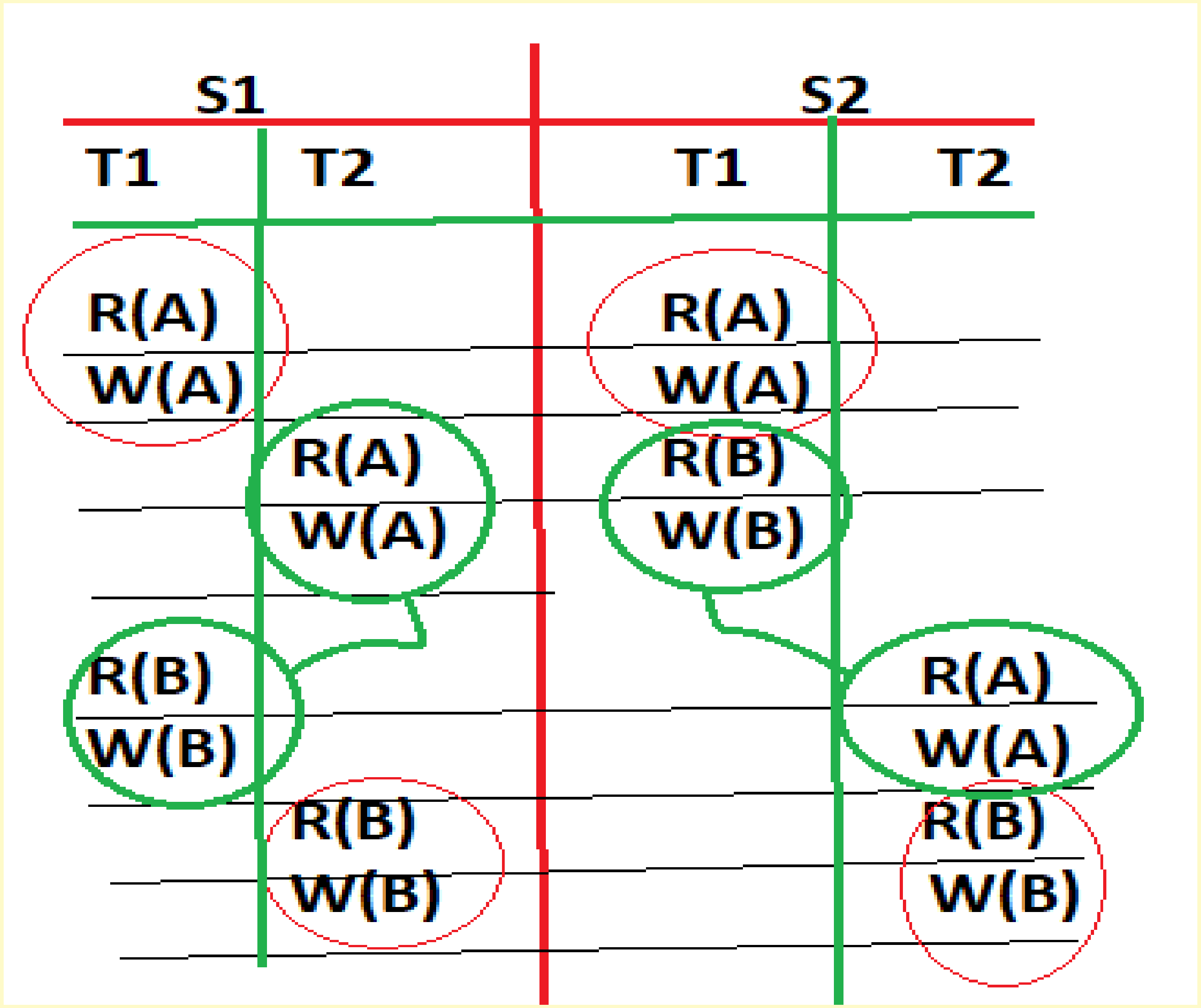
2 instructions are said to be conflicting when they belong to different transactions . But they must operate on same data value and one of the instruction should be write instruction.

How to convert this non serial to serial schedule



S1

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)



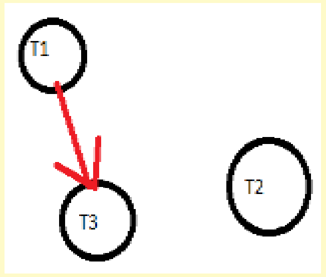
S1		S2	
T1	T2	T1	T2
R(A)		R(A)	
W(A)		W(A)	
	R(A)	R(B)	
	W(A)	W(B)	
R(B)			R(A)
W(B)			W(A)
	R(B)		R(B)
	W(B)		W(B)

- **Conflict serializability def:**

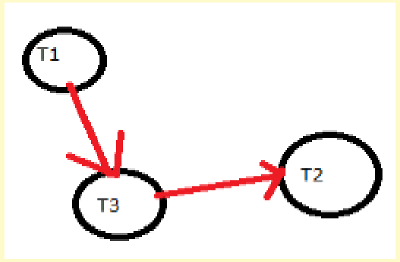
  Non serial schedule is changed to serial schedule by swapping of non conflicting instructions then we say it is conflict seriali-able. So it is consistent .

R(A)	R(A)	NO
R(A)	W(B)	NO
R(A)	W(A)	YES
W(B)	W(A)	NO
W(A)	R(A)	YES

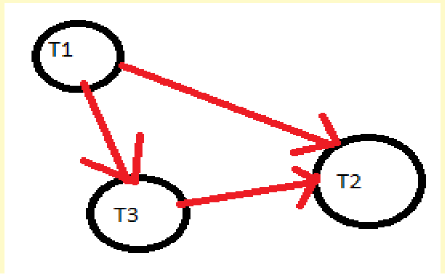
T1	T2	T3
R(X)		
		R(Z)
		W(Z)
	R(Y)	
R(Y)		
	W(Y)	
		W(X)
	W(Z)	
W(X)		



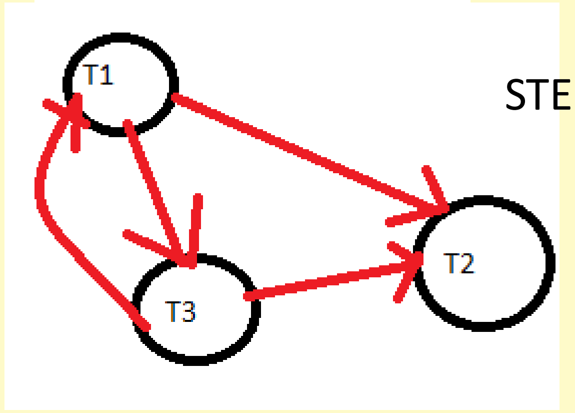
STEP-1



STEP-2



STEP-3



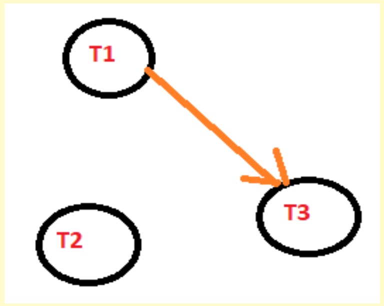
STEP-4

CYCLIC SO NOT A
CONFLICT SERIALIZABLE

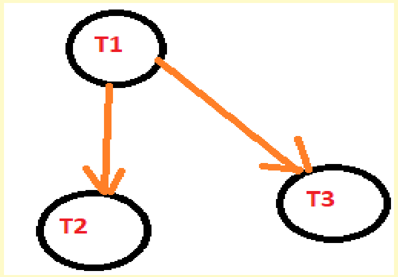
INCONSISTENT

CANNOT CONVERT NON
SERIAL TO SERIAL SCHEDULE

T1	T2	T3
R(X)		
	R(Y)	
		R(Y)
	W(Y)	
W(X)		
		W(X)
	R(X)	
	W(X)	



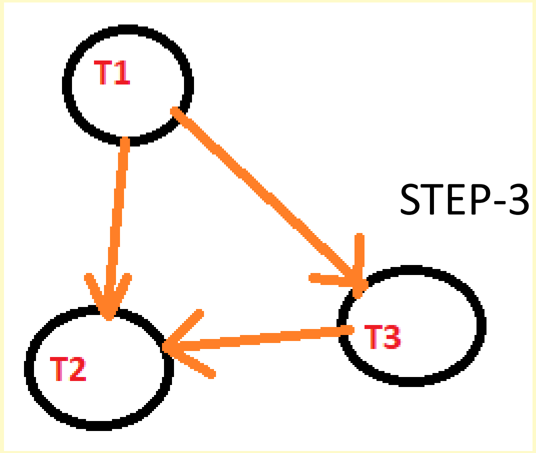
STEP-1



STEP-2

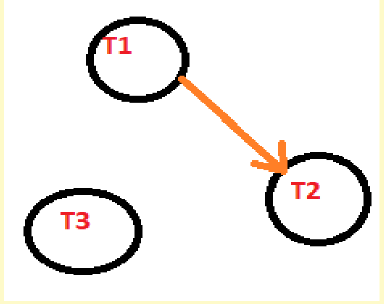
NON CYCLIC
SO CONSISTENT

IT IS CONFLICT SERIALIZABLE

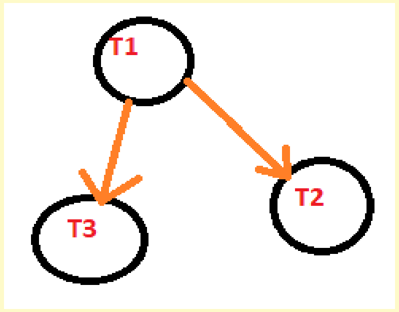


STEP-3

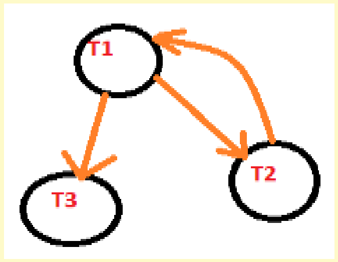
T1	T2	T3
R(B)		
		R(C)
R(A)		
	W(A)	
W(A)		
	W(B)	
		W(A)
W(B)		
		W(B)
		W(C)



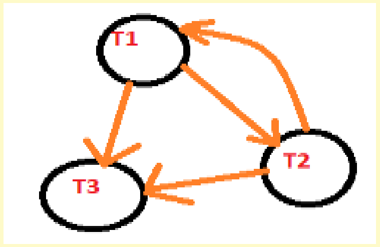
STEP-1



STEP-2



STEP-3



STEP-4

CYCLIC SO
NOT A CONFLICT SERIALIZABLE

INCONSISTENT

T1	T2	T3
R(A)		
		W(A)
	R(A)	
	W(A)	
W(B)		
	W(B)	
		W(A)
	W(C)	
W(A)		
	W(A)	
		W(B)
		W(C)

Ex:1

T1	T2	T3
R(A)		
		W(B)
	R(B)	
W(A)		
	W(B)	
		W(C)
W(B)		
	R(C)	
R(A)		
	W(C)	
		W(A)
R(A)		

EX:2

What is concurrency in DBMS?

- Database **concurrency** is the ability of a database to allow multiple users to affect multiple transactions. This is one of the main properties that separates a database from other forms of data storage, like spreadsheets. ... Other users can read the file, but may not edit data.

What is the need of integrity and concurrency control in DBMS?

- To handle these conflicts we **need concurrency control in DBMS**, which allows transactions to run simultaneously but handles them in such a way so that the **integrity** of data remains intact.

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is **executed** simultaneously on a multi-user system by different users.

- **Reasons for using Concurrency control method is DBMS**: To apply Isolation through mutual exclusion between conflicting transactions. To resolve read-write and write-write conflict issues. To preserve database consistency through constantly preserving execution obstructions.

- **Advantages of concurrency**
- Reduced waiting time response time or turn around time.
- Increased throughput or resource utilization.
- If we run only one transaction at a time than the acid property is sufficient but it is possible that when multiple transactions are executed concurrently than database may become inconsistent.

- **What is concurrency example?**
- **Concurrency** is the tendency for things to happen at the same time in a system. ... Figure 1: **Example** of **concurrency** at work: parallel activities that do not interact have simple **concurrency** issues. It is when parallel activities interact or share the same resources that **concurrency** issues become important.

- Purpose of Concurrency
- 1. To enforce isolation
- 2. To preserve DB consistency
- 3. To resolve R-W and W-W conflict

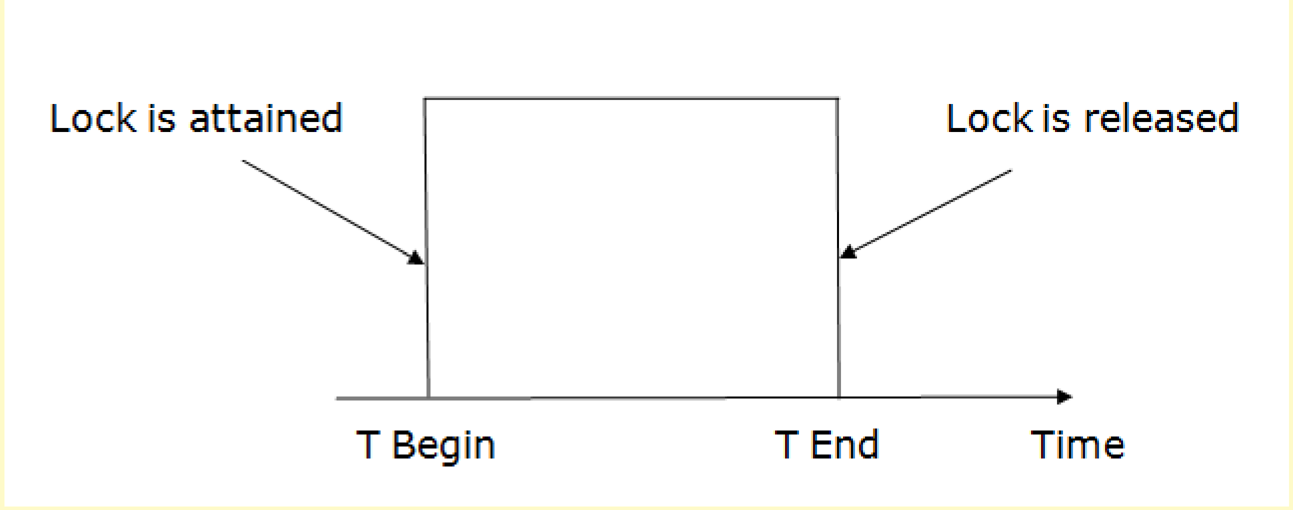
- Concurrency control techniques
- 1.Lock based protocol
- **2.** Two-phase locking Protocol
- **3.** Time stamp ordering Protocol
- **4.** Multi version concurrency control
- **5.** Validation concurrency control

- 1.Lock based protocol
- In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:
-
- **1. Shared lock:**
- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
- **2. Exclusive lock:**
- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.

LOCK COMPATIBILITY

	S	**`X**
S	OK	NO
X	NO	NO



EXCLUSIVE LOCK

T1	T2
LOCK X(B)	
R(B)	
B-50	
W(B)	
UNLOCK (B)	
	LOCK S(B)
	R(B)
	UNLOCK (B)

SHARED LOCK

- Note: Any number of transactions can hold shared lock on one data item

- Exclusive lock can be used by one transaction at a time.

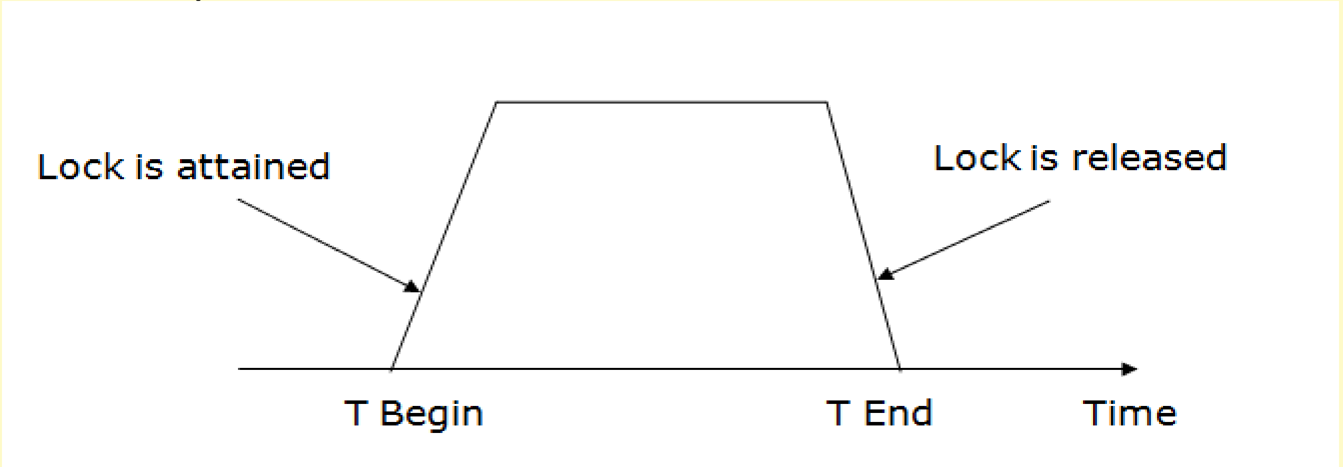
Conversion of Locks

- Upgrading—R-Lock-Write-Lock
- Downgrading-write lock-read lock

Two-phase locking (2PL)

The two-phase locking protocol divides the execution phase of the transaction into three parts.

- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



- There are two phases of 2PL:

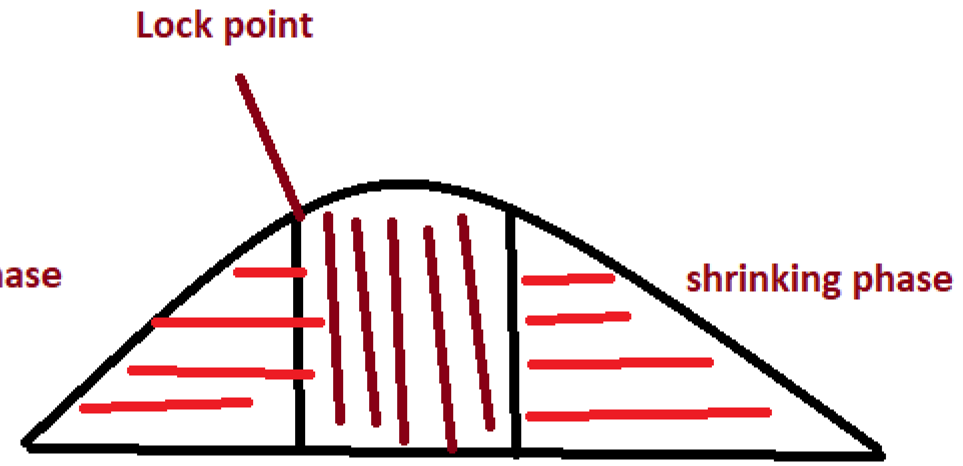
- **Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

- **Shrinking phase:** In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

- In the below example, if lock conversion is allowed then the following phase can happen:

- Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.

- Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.



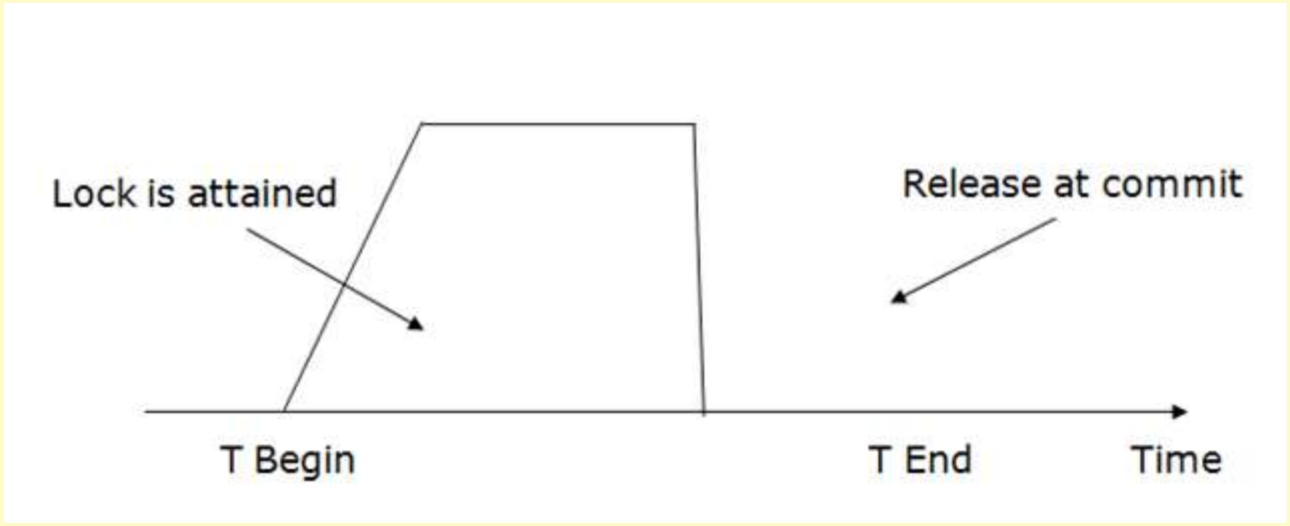
Example:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

- The following way shows how unlocking and locking work with 2-PL.
- **Transaction T1:**
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3
- **Transaction T2:**
- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release
-



Variation of 2PL locking protocol

- Conservative (static)2PL
- Acquires all locks before it starts execution
- Releases all the locks after commit. Deadlock free

- Strict 2PL
- Only exclusive locks cannot be released till commit executes. Deadlock may occur.
-
- Rigorous 2PL: both shared and exclusive locks is not released until transaction is committed, but acquires locks as and when required. Deadlock may occure
-

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 7 o clock times and transaction T2 has entered the system at 9 o clock. T1 has the higher priority, so it executes first as it is entered the system first.

- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

- Time stamp based protocol is to avoid conflict during the execution of the transaction.
- When multiple transactions are handled by the system and if it is overcrowded then we need to think of a mechanism to handle the situation.
- It is like we went to a sweet shop to buy hot jamun and the no of number of jamun prepared are less than the number of people asking for it.
- So what we can do is instead of making every one to ask for the sweet and wait till it is prepared ,where by lot of crowding happens, we can easily maintain the situation by allotting a token with number.

- But if numbers are used there will be a confusion on the following day as we may start with number :1 on next day.

- The same can be applied to temples where people wants to see the god. Like Thirupati. Then it may so happen that the time required will be a day or 2 or 10 days also.
- So instead of making the people to stay and stand in a queue we have assigned with time band.

- This time band is the concept used in DBMS. Where we call it as timestamp.

- The timestamp will be given for the transaction that enters the system for execution
- This time stamp is the time of the clock in the system including date part.
- So when such a TS(timestamp) is given it will be always unique. As date time never repeats.

- The system gives a TS for transaction and also a TS for the data item on which the operation is happening.
- So these are not the same.

- So TS for us is our Data of Birth.

- If the transaction enters the system at 10:00 am on this date then the TS will be accordingly given as 21:05:2021:10:00:00 and if the transaction performs an operation on data item Q at 11:00 then the TS will be 21:05:2021:11:00:00

Ti request for Read(Q)

→ if TS(Ti) < W_TS(Q) means Ti needs to read a value of Q that was already overwritten. Hence request must be rejected and Ti must rollback.

→if TS(Ti) >= W_TS(Q) operation can be allowed .

Ti TS=5	T(x) TS=10
R(Q) has missed this slot	
	W(Q)
Not allowed here ---R(Q)	
	I case

IT IS SWAP OF CONFLICT INSTRN w-r ON Q

if TS(Ti) > W_TS(Q)

Ti TS=12	T(x) TS=10
	W(Q)
R(Q)	
	II case allowed as your request is after write operation

·

if TS(Ti) = W_TS(Q)

Ti TS=12	T(x) TS=10
	W(Q)
	R(Q)

TS(Ti) = W_TS(Q)   WILL BE EQUAL ONLY WHEN THE same transaction has performed Write operation and then Read operation ,but forget the Read operation. So R(Q) is allowed here.

Because no 2 transactions TS will be same . TS will be unique.

Ti request for Write(Q)

→if TS(Ti) < R_TS(Q) means value of Q that Ti is producing was needed previously and the system assumed that the value would never be produced hence reject and rollback

→ If TS(Ti) < W_TS(Q) not allowed

Ti=5	Tx=10
W(Q) requested but not performed	
	R(Q)
W(Q) not allowed	

IT IS SWAP OF CONFLICT INSTRN
w-r ON Q

.

→ If TS(Ti) < W_TS(Q)

Ti=5	Tx=10
W(Q) was suppose to happen here	
	W(Q)
W(Q) not allowed here	

Remaining 2 cases the transaction is allowed.

TS(Ti)>= R_TS(Q)

TS(Ti)>=W_TS(Q)

SO we can say that the Timestamp protocol says that juniors are allowed but not the seniors allowed to do mistakes.

Example

You went to college gate and started doing something and for which other students ignored it

But if the same activity is done by the Professor of the college then others will laugh or say something and so on.

Problems with Concurrency:

1.Dirty Read Problem

2.Unrepeated read problem

3.Phantom read problem

4.Lost update problem/write-write-conflict

Lets understand **Dirty read problem** in database with an example

Imagine there are 2 friends who are going to write exam and a friend X is poor in studies when compared to Y , so X decides to copy the content from X in the examination hall, so during the exam X allows Y to copy the content and then once Y completes writing examination in the sense all the answers to all the questions , he decides to give the paper to the faculty ,thinking everything is done.

So the results are announce Y fails in the examination and X pass the examination with good marks.

So now Y thinks why he got less marks in the exam when he has copied all the answers from Y.
When he asks the question to X , then X explains that after you left the examination hall , and when I was going thru all the answers and modified most of them. So the difference.

So what is the mistake done by Y , he should have submitting the answer sheet after X submits, but it has not happened.

Initial value of A in DB=10

Dirty read problem

A=10

A=A+1=11

T1	T2
R(A)	
W(A)	
	R(A)
	COMMIT
…	
…	
…	
…	
COMMIT	

value of A as 11 which is not a permanent value (uncommitted values)of A.

If T1 rollbacks the transaction then so value of A will be back with 10.

T2 cannot commit bcos he has committed.

Here T1 performs some operations and then commits the transaction

So to avoid the problem of dirty read we need T2 to commit the transaction after T1 commits.

Unrepeated Read Problem

Story /Example:

One day an young guy see that his parents are going out and because of that he feels very happy and in this happiness he tries to do something thinking that he is only present in the house, so he thinks of cooking a maggie and wanted to watch ca`rtoon network.

In this process he takes maggie packet and cooker to cook the same, he puts the cooker on the gas stove and then goes to watch cartoon nw.

When he comes back after 10 minutes and when he takes out the lid of the cooker he observes that the cooker contains Rice with rajma and not maggie.

In the anger he tries to throw the rice bowl out and while doing so he gets a voice of someone sitting on the fan saying it is me and I wants the rice with rajma.

Story ends here.

So what we want to understand here is thinking that he is alone he has done something without knowing that there may be ghost in the house.

In the same way when we run any Transaction, every transaction feels that it is getting executed and no other transactions are in process. And no will be no transaction.

X=10	T1	T2	
x=10	R(x)		
		R(x)	X=10
	W(x)		
X=x+5		R(x)	X=15
	…	….	
	…	…	
	…	…	

T2 is confused because in two read it gets different values as 10 and 15. it will not understand why this is happening as every transaction feels that it is the only transaction working in the system.

Phantom Read Problem

Parents of the boy explains him that there is nothing like ghost and all it is your thought .

As said earlier that the boy again see that his parents are going out and then he tries to do something in the kitchen, and he take the cooker and the pasta packet and cooks the same by placing it on a gas stove, with an ounce of doubt that there is be a ghost in the house. The boy again goes and watch the cartoon nw. when he goes back to the kitchen after sometime he observes that there is nothing in the kitchen.

Phantom Read Problem

.

T1	T2
R(x)	
	R(x)
Delete(x)	
	R(x)

Initial value of X =10

T1 read the value of X which is 10

T2 reads the value of X which is 10

Then T1 deletes X .

T2 now tries to read the value of X and he finds that the particular variable do not exists. This is called phantom read problem.

Reading a value after it is deleted.

Lost Update / Write-Write Conflict

.

Value of A=10

Then A is updated as A=A+5. so A=15

T1	T2
R(A)	
W(A)	
	W(A)
	Commit
Commit	

T2 without reading the value of A will modify A as 50 and commits , Without reading the value of A , without reading the value if we write then it is called blind write.

Thinking that A=15 it will perform commit operation ,but what is the value written in database , it is 50 and not 15.

So the value which T1 wants to write is not stored in DB.

This is called W-W conflict or Lost update.

Validation Based Protocol

.

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

Read phase: In this phase, the transaction T is read data from database and executes the operations. It is used to read the value of various data items and stores them in **temporary local variables**. It can perform all the write operations on temporary variables **without an update** to the actual database.

Validation phase: In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

Write phase: If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

This protocol is also called Optimistic protocol because it will continue execution of the statement thinking that every thing will finish without any problem .

This protocol maintains 3 timestamps

1. Timestamp of start of the transaction
2. Timestamp of the validation phase
3. Finish timestamp i.e end of write phase

This leads to starvation in the system as the transaction has to be roll backed every time the validation fails. And once again we have to start a fresh transaction .

There is no chance of deadlock as we are using timestamp to check for serializability of the schedule and so on.

. Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

It can be defined as hierarchically breaking up the database into blocks which can be locked.

The Multiple Granularity protocol enhances concurrency and reduces lock overhead.

It maintains the track of what to lock and how to lock.

It makes easy to decide either to lock a data item or to unlock a data item.

This type of hierarchy can be graphically represented as a tree.

. **For example:** Consider a tree which has four levels of nodes.

The first level or higher level shows the entire database.

The second level represents a node of type area. The higher level database consists of exactly these areas.

The area consists of children nodes which are known as files. No file can be present in more than one area.

Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.

Hence, the levels of the tree starting from the top level are as follows:

Database

Area

File

Record

In this example, the highest level shows the entire database. The levels below are file, record, and fields.

.

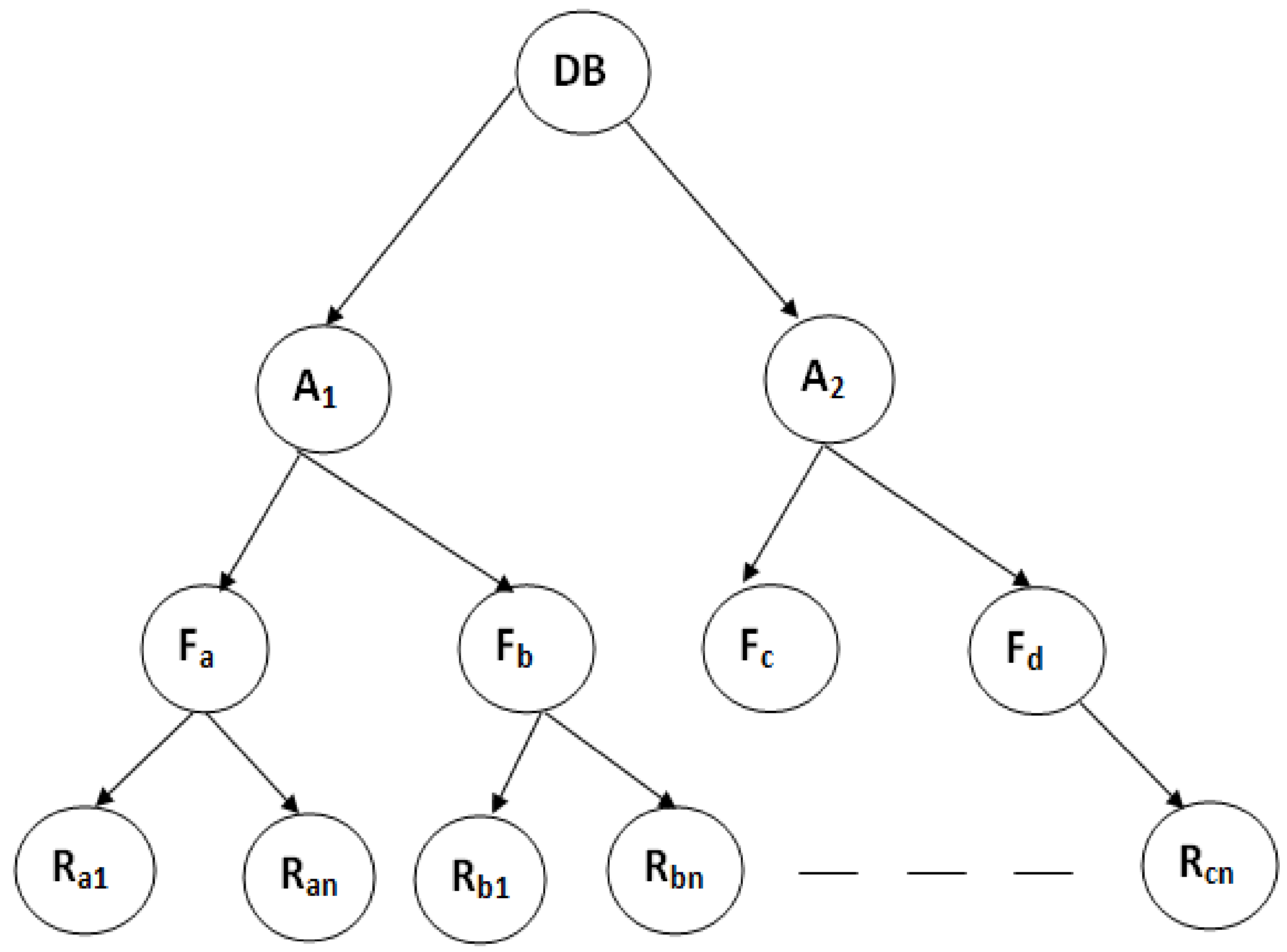


Figure: Multi Granularity tree Hierarchy

What is Data Recovery:-

It is the method of restoring the database to its correct state in the event of a failure at the time of the transaction or after the end of a process. dependability refers to both the flexibility of the DBMS to various kinds of failure and its ability to recover from those failures.

To gain a better understanding of the possible problems you may encounter in providing a consistent system, you will first learn about the need for recovery and its types of failure, which usually occurs in a database environment.

. What is the Need for Recovery of data?

The storage of data usually includes four types of media with an increasing amount of reliability: the main memory, the magnetic disk, the magnetic tape, and the optical disk. Many different forms of failure can affect database processing and/or transaction, and each of them has to be dealt with differently. Some data failures can affect the main memory only, while others involve non-volatile or secondary storage also. Among the sources of failure are:

Due to hardware or software errors, the system crashes, which ultimately resulting in loss of main memory.

Failures of media, such as head crashes or unreadable media that results in the loss of portions of secondary storage.

There can be application software errors, such as logical errors that are accessing the database that can cause one or more transactions to abort or fail.

Natural physical disasters can also occur, such as fires, floods, earthquakes, or power failures.

Carelessness or unintentional destruction of data or directories by operators or users.

Damage or intentional corruption or hampering of data (using malicious software or files) hardware or software facilities.

Whatever the grounds of the failure are, there are two principal things that you have to consider:

. Failure of main memory, including that database buffers.

Failure of the disk copy of that database.

Recovery Facilities

Every DBMS should offer the following facilities to help out with the recovery mechanism:

Backup mechanism makes backup copies at a specific interval for the database.

Logging facilities keep tracing the current state of transactions and any changes made to the database.

Checkpoint facility allows updates to the database for getting the latest patches to be made permanent and keep secure from vulnerability.

Recovery manager allows the database system for restoring the database to a reliable and steady-state after any failure occurs.

Log-Based Recovery

The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.

If any operation is performed on the database, then it will be recorded in the log.

But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.
- <Tn, Start>
- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.
- <Tn, City, 'Noida', 'Bangalore' >
- When the transaction is finished, then it writes another log to indicate the end of the transaction.
- <Tn, Commit>
- There are two approaches to modify the database:
- .

. 1. Deferred database modification:

The deferred modification technique occurs if the transaction does not modify the database until it has committed.

In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

The Immediate modification technique occurs if database modification occurs while the transaction is still active.

In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Checkpoint

The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.

The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.

When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.

The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

.

. Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:

The recovery system reads log files from the end to start. It reads log files from T4 to T1.
Recovery system maintains two lists, a redo-list, and an undo-list.
The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.

- Deadlock in DBMS
- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.
- **For example:** In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.
- Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.
-

- Concurrency control means that multiple transactions can be executed at the same time and then the interleaved logs occur. But there may be changes in transaction results so maintain the order of execution of those transactions.
- During recovery, it would be very difficult for the recovery system to backtrack all the logs and then start recovering.
- Recovery with concurrent transactions can be done in the following four ways.
- Interaction with concurrency control
- Transaction rollback
- Checkpoints
- Restart recovery
- **Interaction with concurrency control :**
- In this scheme, the recovery scheme depends greatly on the concurrency control scheme that is used. So, to rollback a failed transaction, we must undo the updates performed by the transaction.
-

- **Transaction rollback :**
- In this scheme, we rollback a failed transaction by using the log.
- The system scans the log backward a failed transaction, for every log record found in the log the system restores the data item.

- **Checkpoints :**
- Checkpoints is a process of saving a snapshot of the applications state so that it can restart from that point in case of failure.
- Checkpoint is a point of time at which a record is written onto the database form the buffers.

- Checkpoint shortens the recovery process.
- When it reaches the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till the next checkpoint and so on.
- The checkpoint is used to declare the point before which the DBMS was in the consistent state, and all the transactions were committed.

- To ease this situation, ‘Checkpoints‘ Concept is used by the most DBMS.
- In this scheme, we used checkpoints to reduce the number of log records that the system must scan when it recovers from a crash.
- In a concurrent transaction processing system, we require that the checkpoint log record be of the form <checkpoint L>, where **‘L’** is a list of transactions active at the time of the checkpoint.
- A fuzzy checkpoint is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out.

- **Restart recovery :**
- When the system recovers from a crash, it constructs two lists.
- The undo-list consists of transactions to be undone, and the redo-list consists of transaction to be redone.
- The system constructs the two lists as follows: Initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <checkpoint> record.

- UNIT- 5

File Organization

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

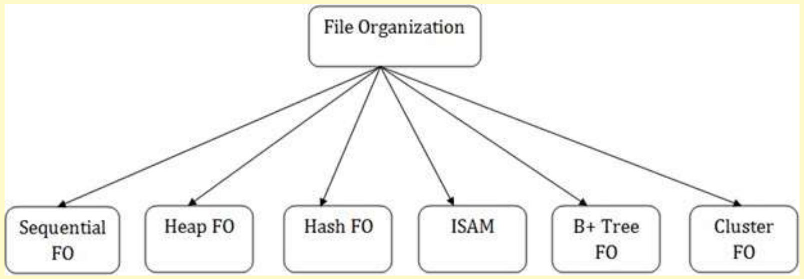
Objective of file organization

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

Types of file organization:

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

Types of file organization are as follows:



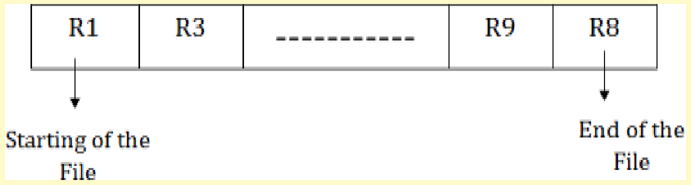
- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

Sequential File Organization

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

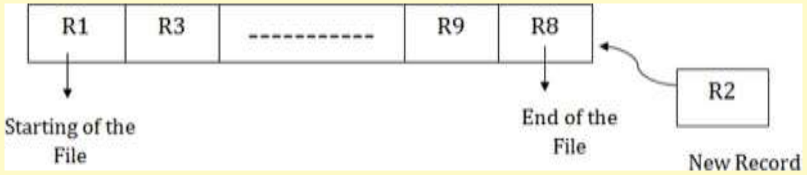
1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



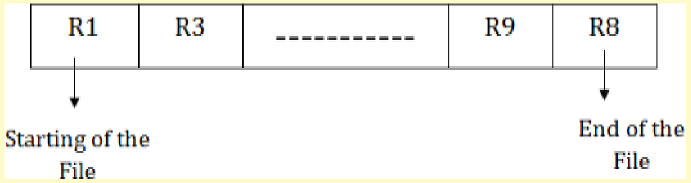
Insertion of the new record:

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



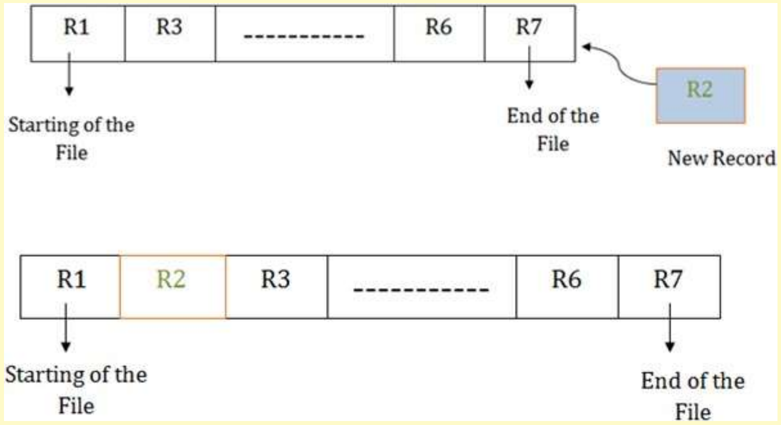
2. Sorted File Method:

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



Insertion of the new record:

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



Pros of sequential file organization

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

Cons of sequential file organization

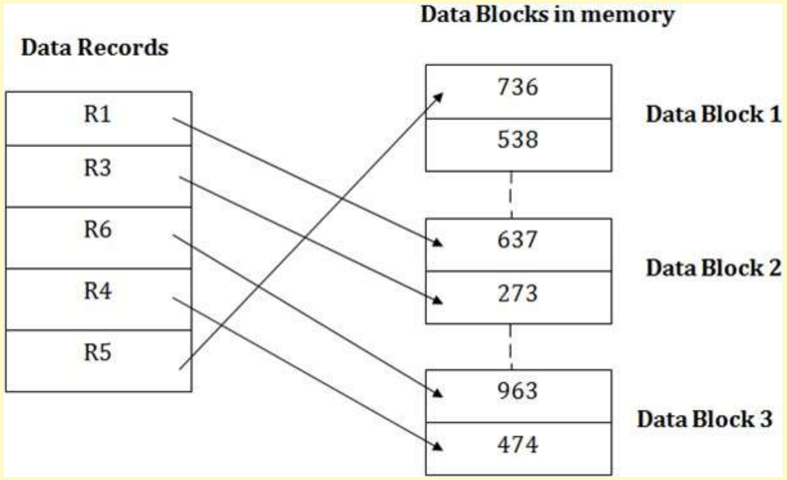
- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but

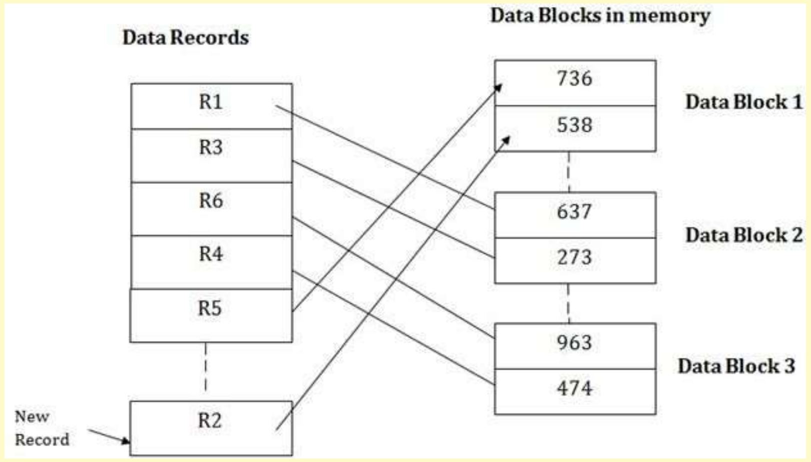
it can select any data block in the memory to store new records. The heap file is also **known as an unordered file**.

- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

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



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

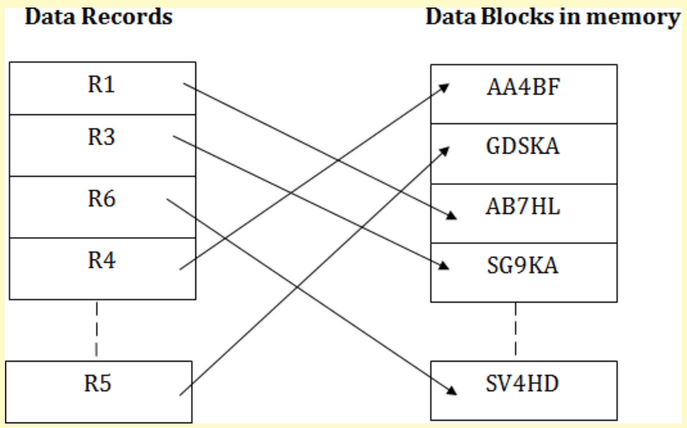
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

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

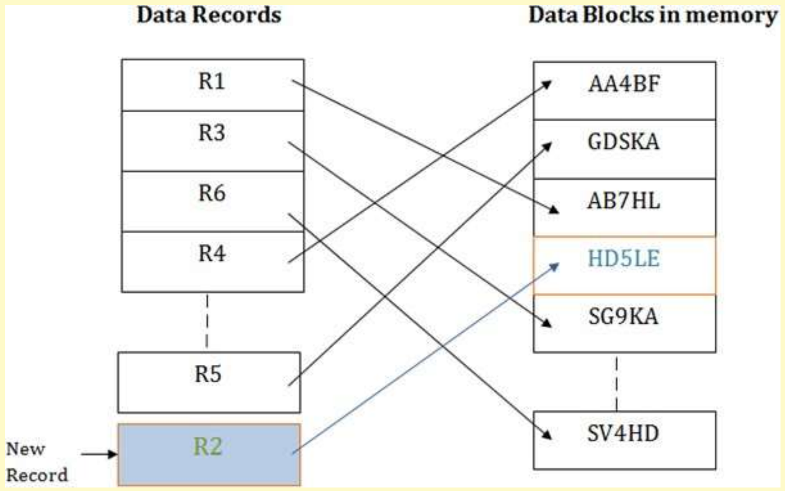
Hash File Organization

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



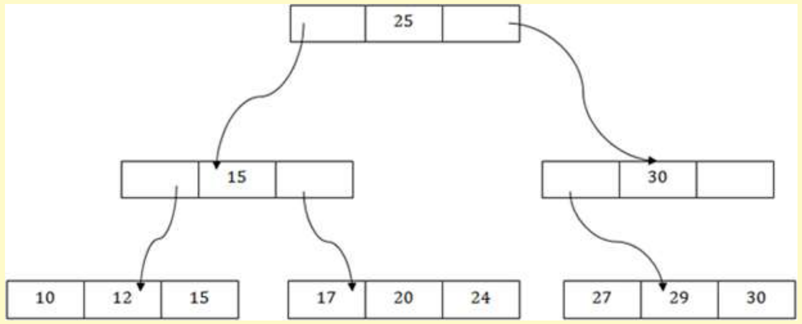
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



B+ File Organization

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



The above B+ tree shows that:

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

Pros of B+ tree file organization

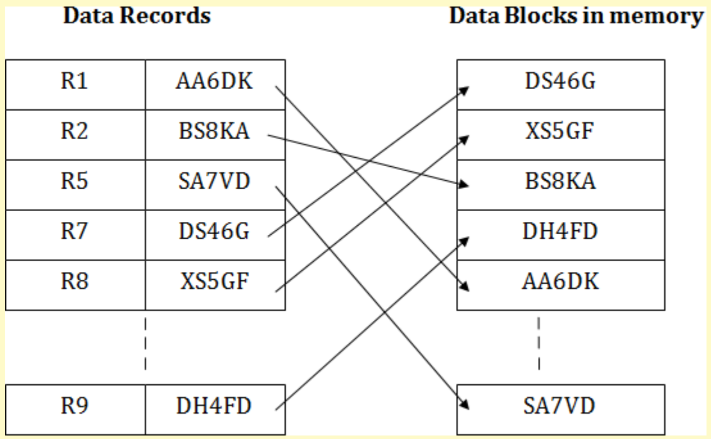
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and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

Cons of B+ tree file organization

- This method is inefficient for the static method.

Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

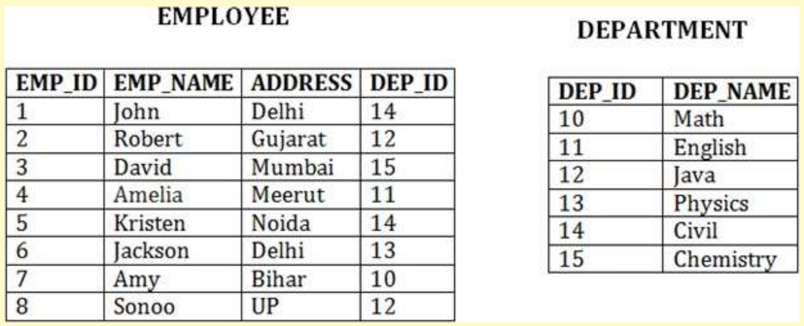
Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.

- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

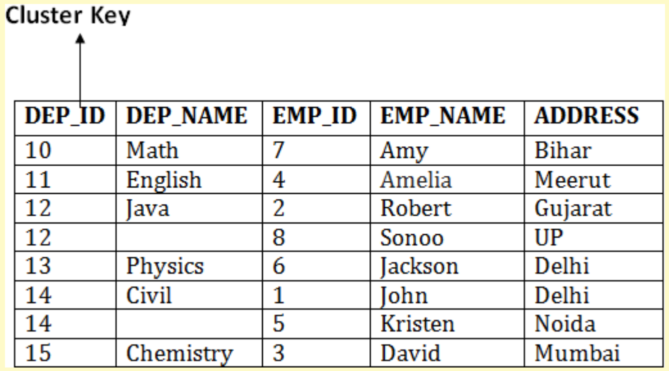


EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry



Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:

In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. Hash Clusters:

It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Pros of Cluster file organization

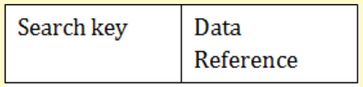
- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.
- Indexing in DBMS
  - Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
  - The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

Indexes can be created using some database columns.

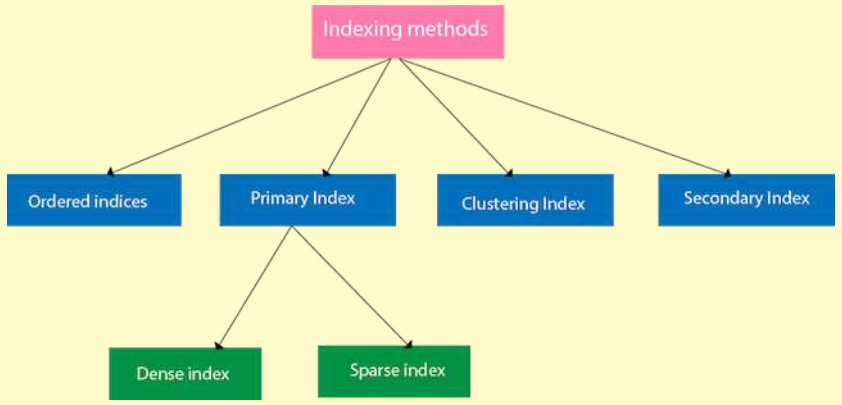


Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

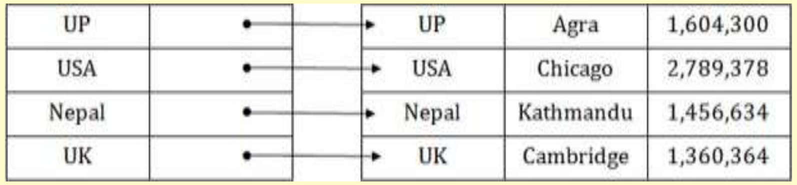
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.

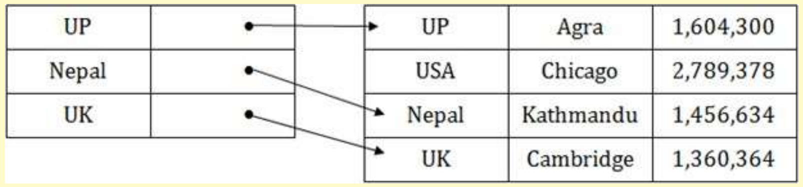


UP	●
USA	●
Nepal	●
UK	●

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

Sparse index

- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



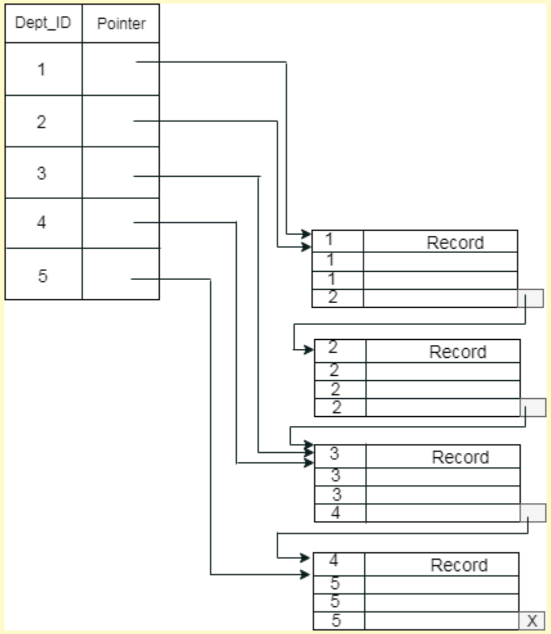
UP	●
Nepal	●
UK	●

UP	Agra	1,604,300
USA	Chicago	2,789,378
Nepal	Kathmandu	1,456,634
UK	Cambridge	1,360,364

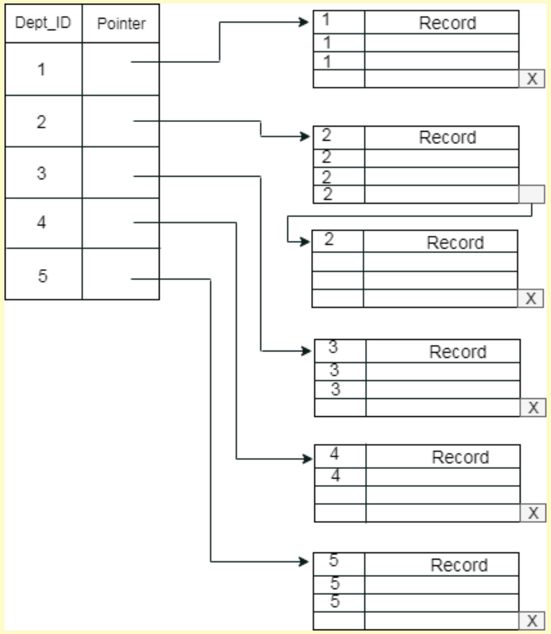
Clustering Index

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



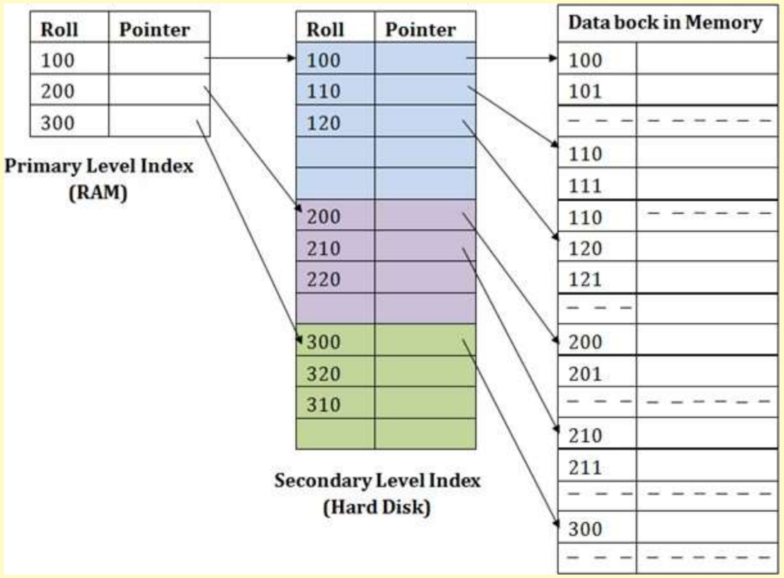
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



For example:

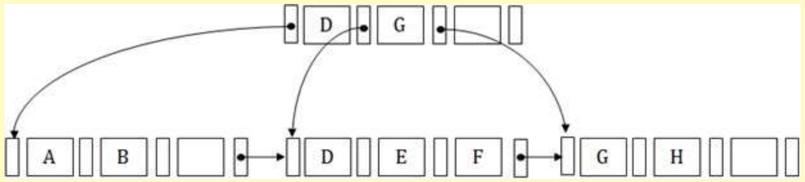
- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

Structure of B+ Tree

- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.



Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

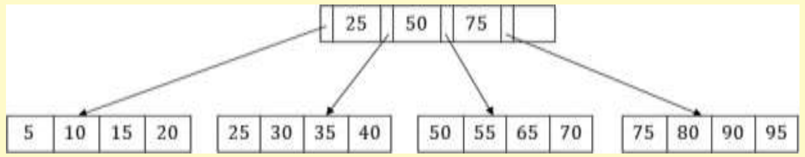
Leaf node

- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

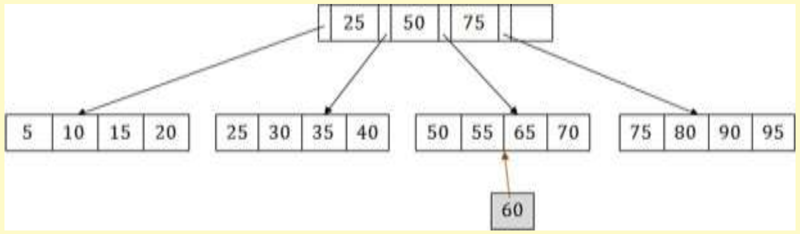
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

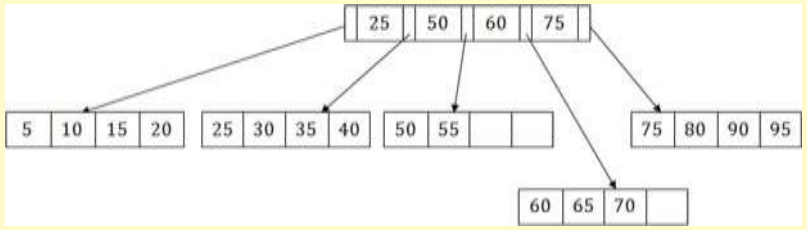
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

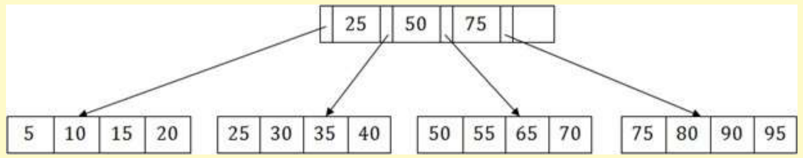


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:

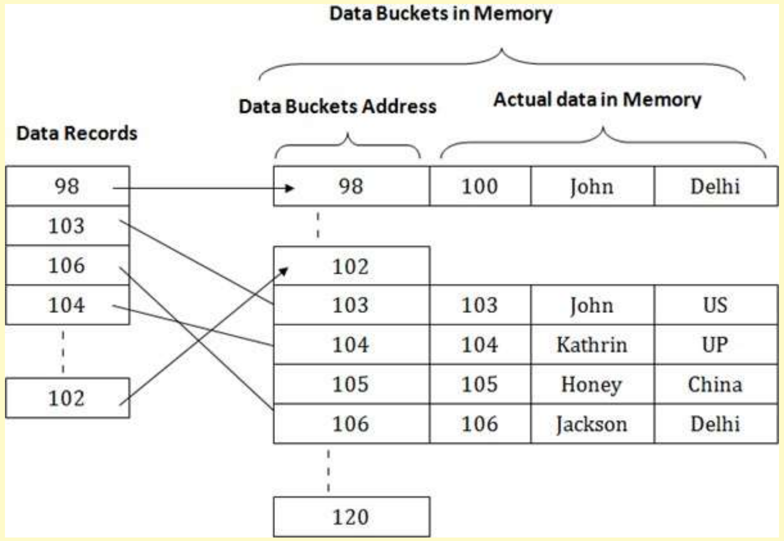


Hashing

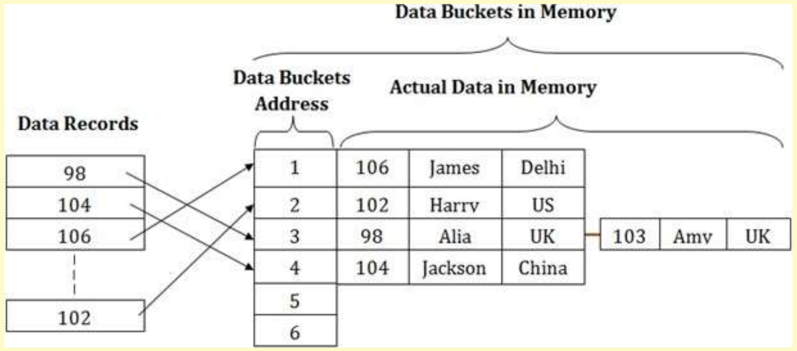
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

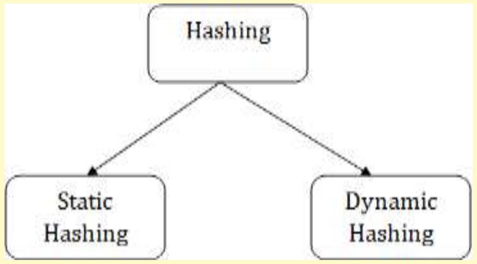
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



Types of Hashing:

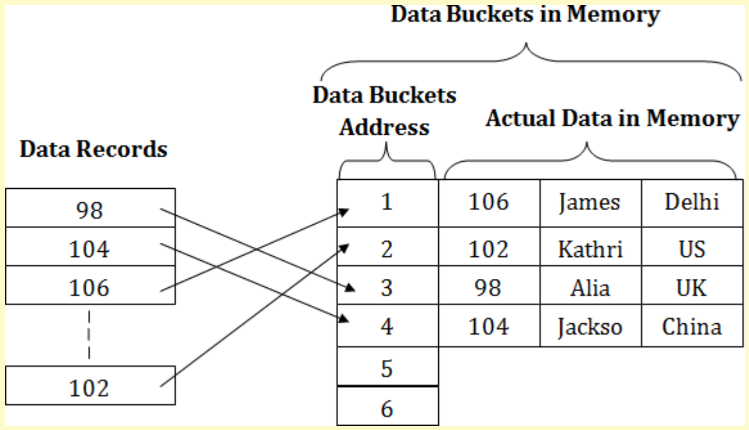


- Static Hashing
- Dynamic Hashing

Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



Operations of Static Hashing

- **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

- **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

- **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

- **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

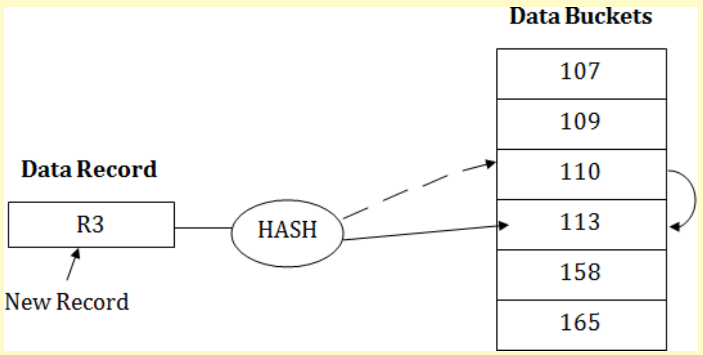
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

1. Open Hashing

When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.

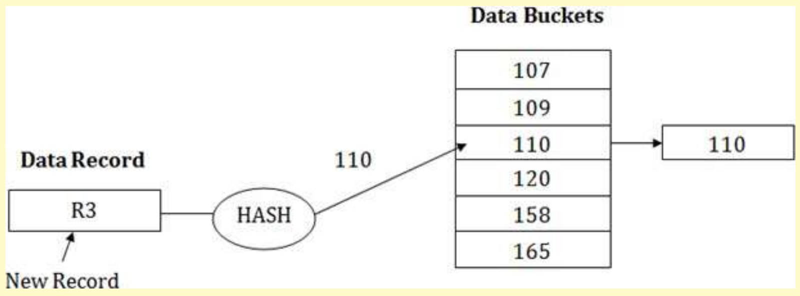
For example: suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



2. Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

For example: Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.

- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

How to search a key

- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

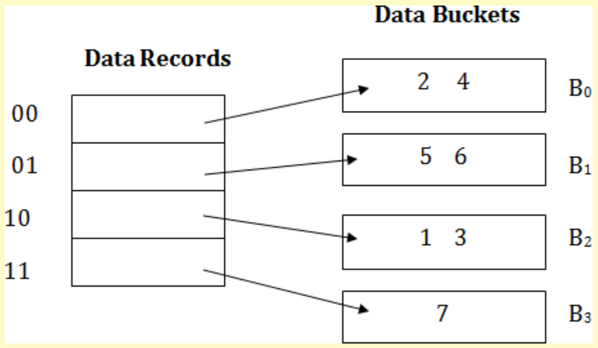
- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

For example:

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

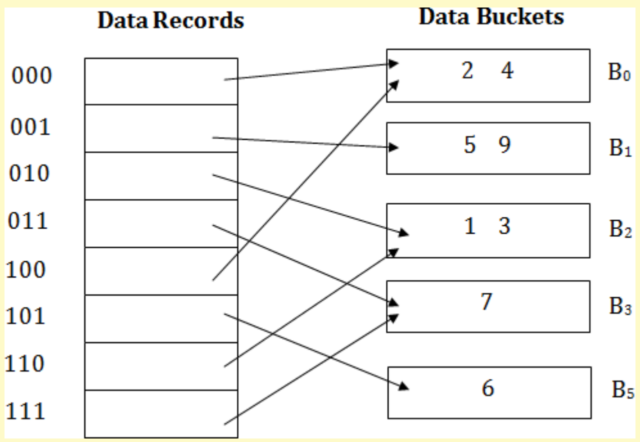
Key	**Hash address**
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

Duplicate tuples