DBMS LECTURER NOTE 4th SEM CSE DIPLOMA

COURSE CONTENTS:

1.0 BASIC CONCPETS OF DBMS

1.1 Purpose of database Systems
1.2 Explain Data abstraction
1.3 Database users
1.4 Data definition language
1.5 Data Dictionary

2.0 DATA MODELS

2.1 Data independence
2.2 Entity relationship models
2.3 Entity sets and Relationship sets
2.4 Explain Attributes
2.5 Mapping constraints

2.6 E-R Diagram
2.7 Relational model
2.8 Hierarchical model
2.9 Network model

3.0 RELATIONAL DATABASE

3.1 Relational algebra
3.2 Different operators select, project, join , simple Examples

4.0 NORMALIZATION IN RELATIONAL SYSTEM

4.1 Functional Dependencies
4.2 Lossless join
4.3 Importance of normalization
4.4 Compare First second and third normal forms
4.5 Explain BCNF

5.0 STRUCTURED QUERY LANGUAGE

5.1 Elementary idea of Query language
5.2 Queries in SQL
5.3 Simple queries to create, update, insert in SQL

6.0 TRANSACTION PROCESSING CONCEPTS

6.1 Idea about transaction processing
6.2 Transaction & system concept
6.3 Desirable properties of transaction
6.4 Schedules and recoverability

7.0 CONCURRENCY CONTROL CONCEPTS

7.1 Basic concepts,
7.2 Locks, Live Lock, Dead Lock,
7.3 Serializability (only fundamentals)

8.0 SECURITY AND INTEGRITY

8.1 Authorization and views
8.2 Security constraints
8.3 Integrity Constraints

8.4 Discuss Encryption

DETAILS OF LECTURER NOTE:

1.0 BASIC CONCPETS OF DBMS

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

1.1 Purpose of database Systems

Database Management Systems (DBMS) are software systems used **to store, retrieve, and run queries on data**. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

The purpose of database systems is to make the database user-friendly and do easy operations. Users can easily insert, update, and delete. Actually, the main purpose is to have more control of the data.

The purpose of database systems is to manage the following insecurities:

- data redundancy and inconsistency,
- difficulty in accessing data,
- data isolation,
- atomicity of updates,
- concurrent access,
- security problems, and
- supports multiple views of data.

1.2 Explain Data abstraction

Data Abstraction is a process of hiding unwanted or irrelevant details from the end user. It provides a different view and helps in achieving data independence which is used to enhance the security of data.

The database systems consist of complicated data structures and relations. For users to access the data easily, these complications are kept hidden, and only the relevant part of the database is made accessible to the users through data abstraction.

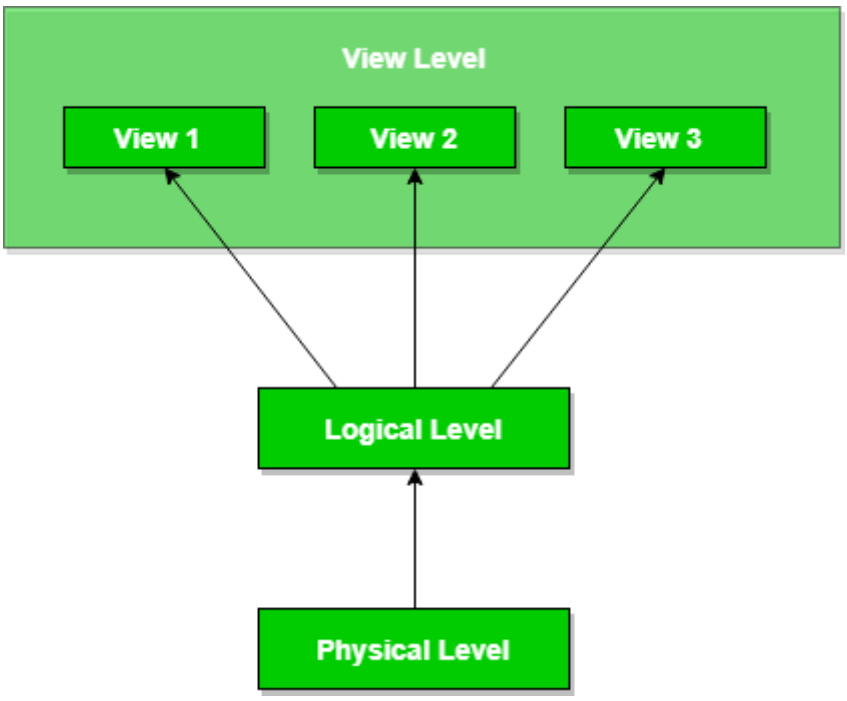
Levels of abstraction for DBMS

Database systems include complex data-structures. In terms of retrieval of data, reduce complexity in terms of usability of users and in order to make the system efficient, developers use levels of abstraction that hide irrelevant details from the users. Levels of abstraction simplify database design.

Mainly there are three levels of abstraction for DBMS, which are as follows −

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level

These levels are shown in the diagram below –



Let us discuss each level in detail.

Physical or Internal Level

It is the lowest level of abstraction for DBMS which defines how the data is actually stored, it defines data-structures to store data and access methods used by the database. Actually, it is decided by developers or database application programmers how to store the data in the database.

So, overall, the entire database is described in this level that is physical or internal level. It is a very complex level to understand. For example, customer's information is stored in tables and data is stored in the form of blocks of storage such as bytes, gigabytes etc.

Logical or Conceptual Level

Logical level is the intermediate level or next higher level. It describes what data is stored in the database and what relationship exists among those data. It tries to describe the entire or whole data because it describes what tables to be created and what are the links among those tables that are created.

It is less complex than the physical level. Logical level is used by developers or database administrators (DBA). So, overall, the logical level contains tables (fields and attributes) and relationships among table attributes.

View or External Level

It is the highest level. In view level, there are different levels of views and every view only defines a part of the entire data. It also simplifies interaction with the user and it provides many views or multiple views of the same database.

View level can be used by all users (all levels' users). This level is the least complex and easy to understand.

For example, a user can interact with a system using GUI that is view level and can enter details at GUI or screen and the user does not know how data is stored and what data is stored, this detail is hidden from the user.

1.3 Database users

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

These are seven types of data base users in DBMS.

1. **Database Administrator (DBA) :**
   Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database.
   The DBA will then create a new account id and password for the user if he/she need to access the data base.
   DBA is also responsible for providing security to the data base and he allows only the authorized users to access/modify the data base.
   - DBA also monitors the recovery and back up and provide technical support.
   - The DBA has a DBA account in the DBMS which called a system or superuser account.
   - DBA repairs damage caused due to hardware and/or software failures.

2. **Naive / Parametric End Users :**
   Parametric End Users are the unsophisticated who don’t have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results.
   For examples, Railway’s ticket booking users are naive users. Clerks in any bank is a naive user because they don’t have any DBMS knowledge but they still use the database and perform their given task.

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

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

5. **Data Base Designers :**
   Data Base Designers are the users who design the structure of data base which

includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items to be related.

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

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

1.4 Data definition language

Data Definition Language

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

Here are some tasks that come under DDL:

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

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

1.5 Data Dictionary

A data dictionary contains metadata i.e data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.

The data dictionary in general contains information about the following −

- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that are visible.

This is a data dictionary describing a table that contains employee details.

Field Name	Data Type	Field Size for display	Description	Example
EmployeeNumber	Integer	10	Unique ID of each employee	1645000001
Name	Text	20	Name of the employee	David Heston
Date of Birth	Date/Time	10	DOB of Employee	08/03/1995
Phone Number	Integer	10	Phone number of employee	6583648648

The different types of data dictionary are −

Active Data Dictionary

If the structure of the database or its specifications change at any point of time, it should be reflected in the data dictionary. This is the responsibility of the database management system in which the data dictionary resides.

So, the data dictionary is automatically updated by the database management system when any changes are made in the database. This is known as an active data dictionary as it is self updating.

Passive Data Dictionary

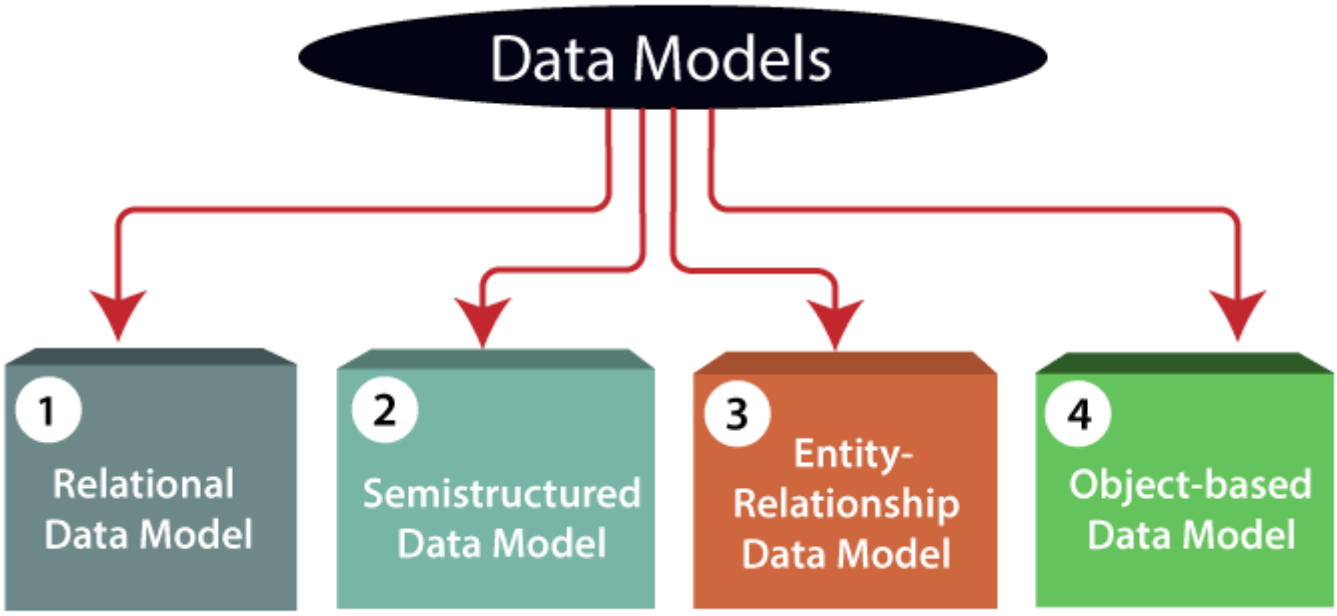
This is not as useful or easy to handle as an active data dictionary. A passive data dictionary is maintained separately to the database whose contents are stored in the dictionary. That means that if the database is modified the database dictionary is not automatically updated as in the case of Active Data Dictionary.

So, the passive data dictionary has to be manually updated to match the database. This needs careful handling or else the database and data dictionary are out of sync.

2.0 DATA MODELS

Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



1) Relational Data Model: This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

2) Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

3) Object-based Data Model: An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

4) Semistructured Data Model: This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

2.1 Data independence

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

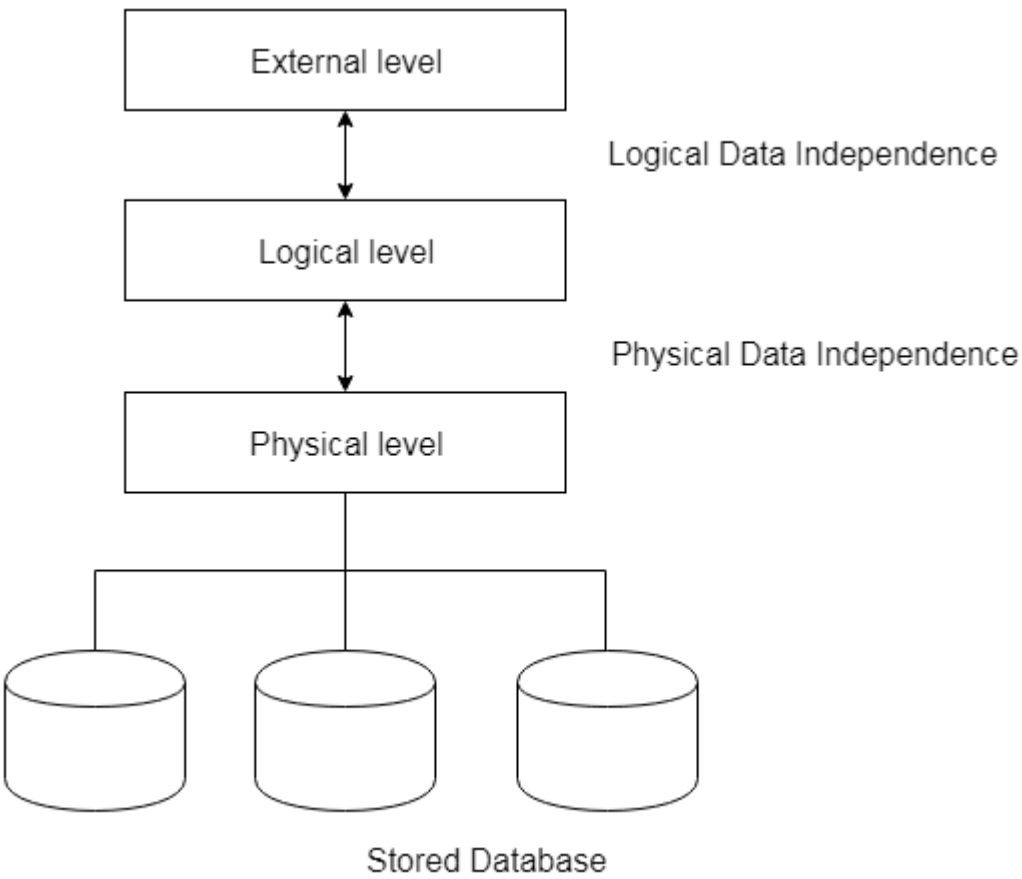
There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.



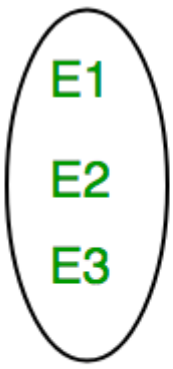
2.2 Entity relationship models

ER Model is used to model the logical view of the system from data perspective which consists of these components:

Entity, Entity Type, Entity Set –
An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:

Entity Type

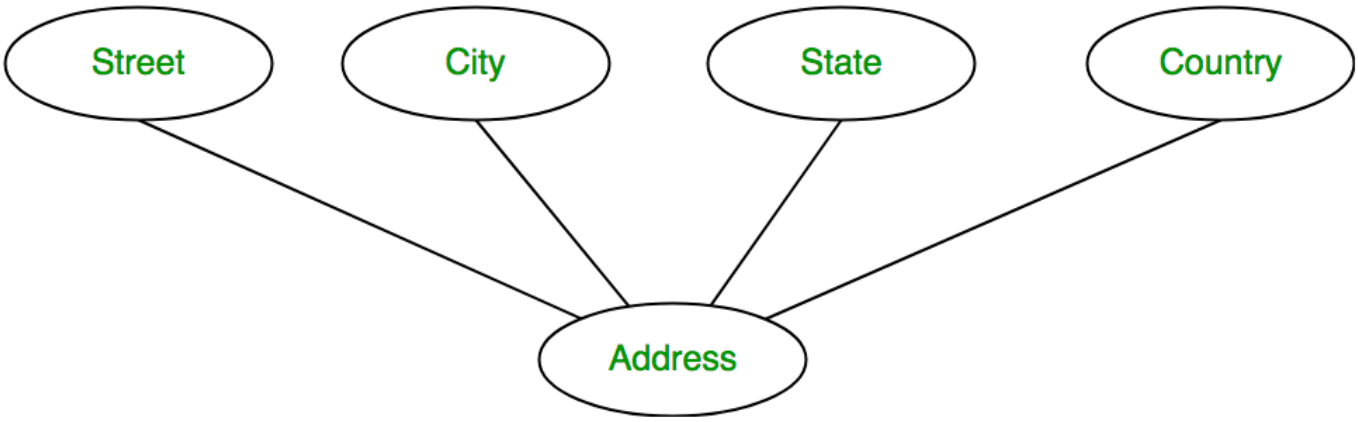


Entity Set

Attribute(s):
Attributes are the **properties which define the entity type**. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

1. Key Attribute –
The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

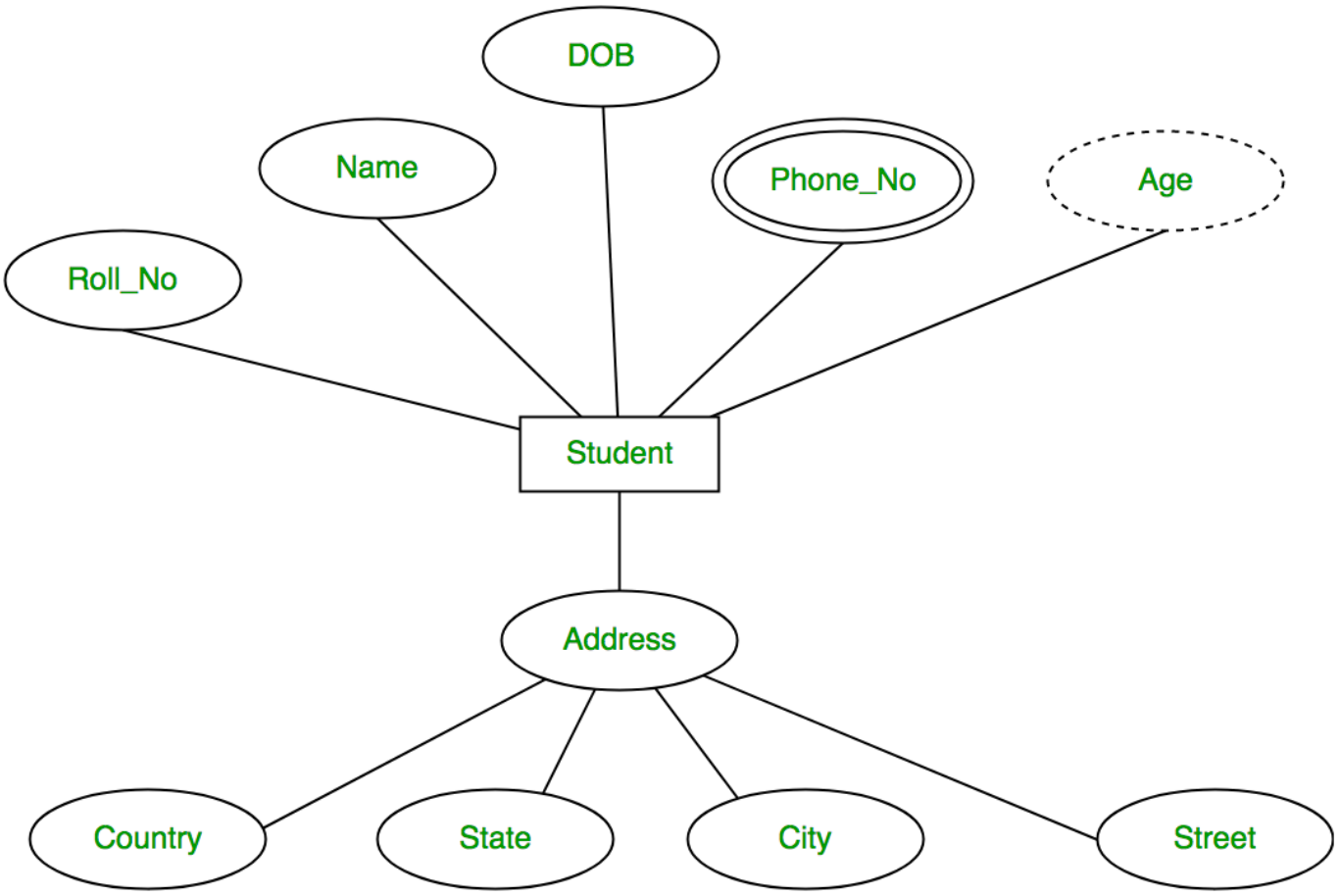
2. Composite Attribute –
An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



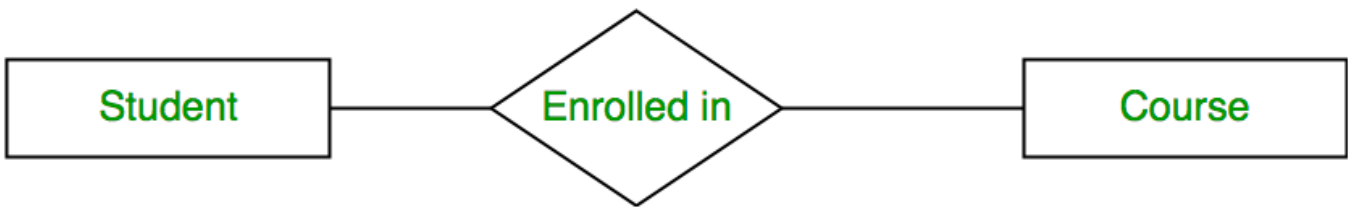
3. Multivalued Attribute –
An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

4. Derived Attribute –
An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

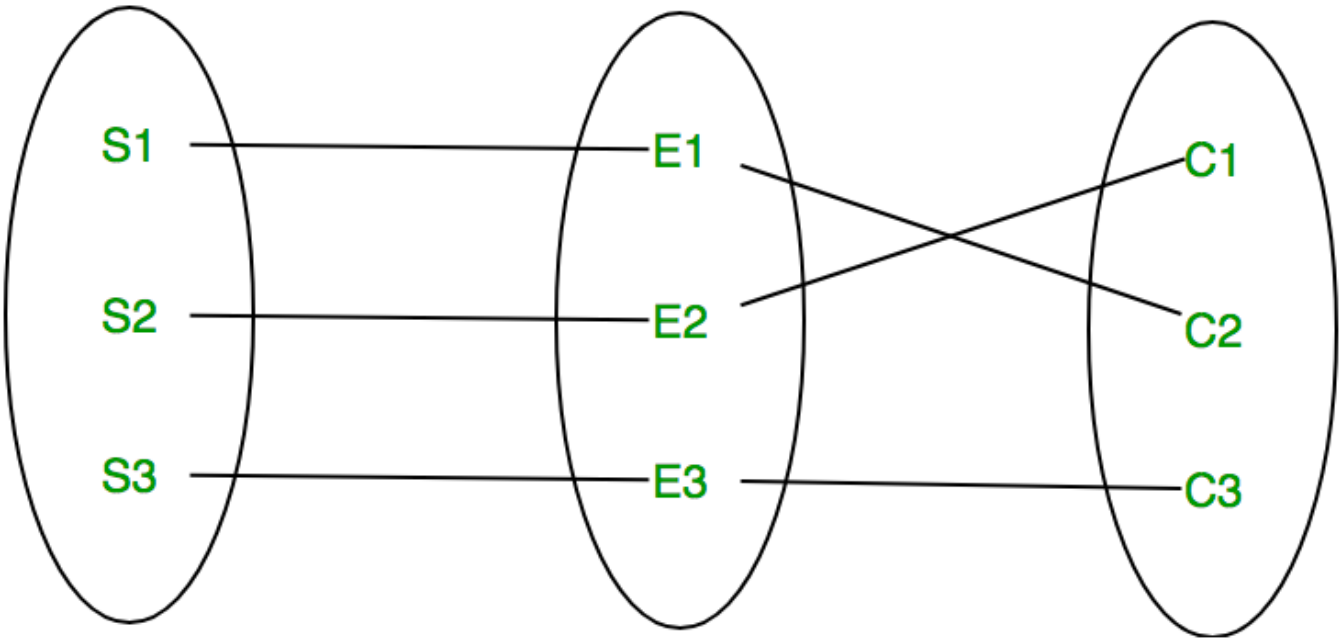
The complete entity type **Student** with its attributes can be represented as:



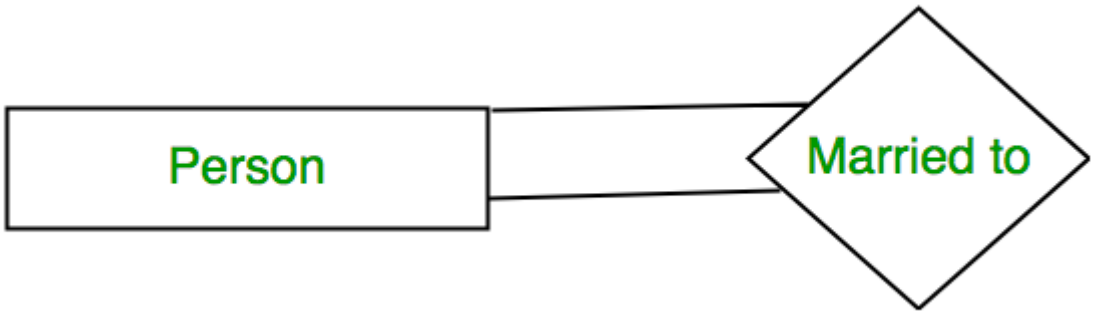
Relationship Type and Relationship Set:
A relationship type represents the **association between entity types**. For example,'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities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



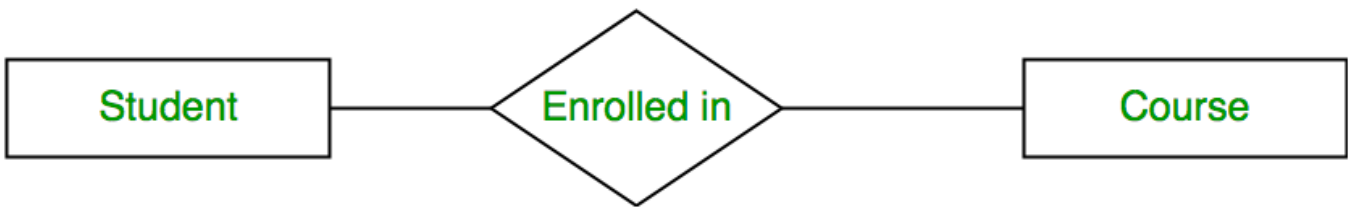
A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



Degree of a relationship set:
The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.
1. 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.

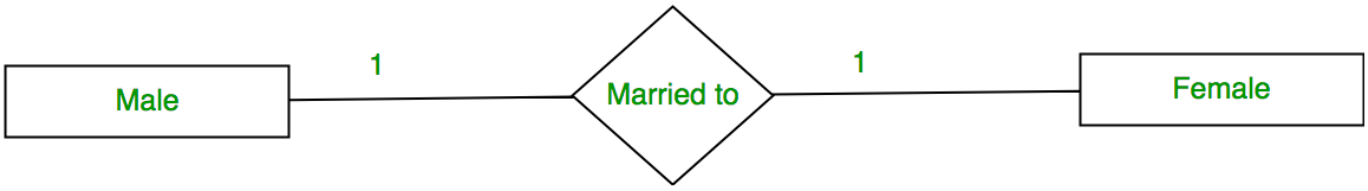


2. Binary Relationship –
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.

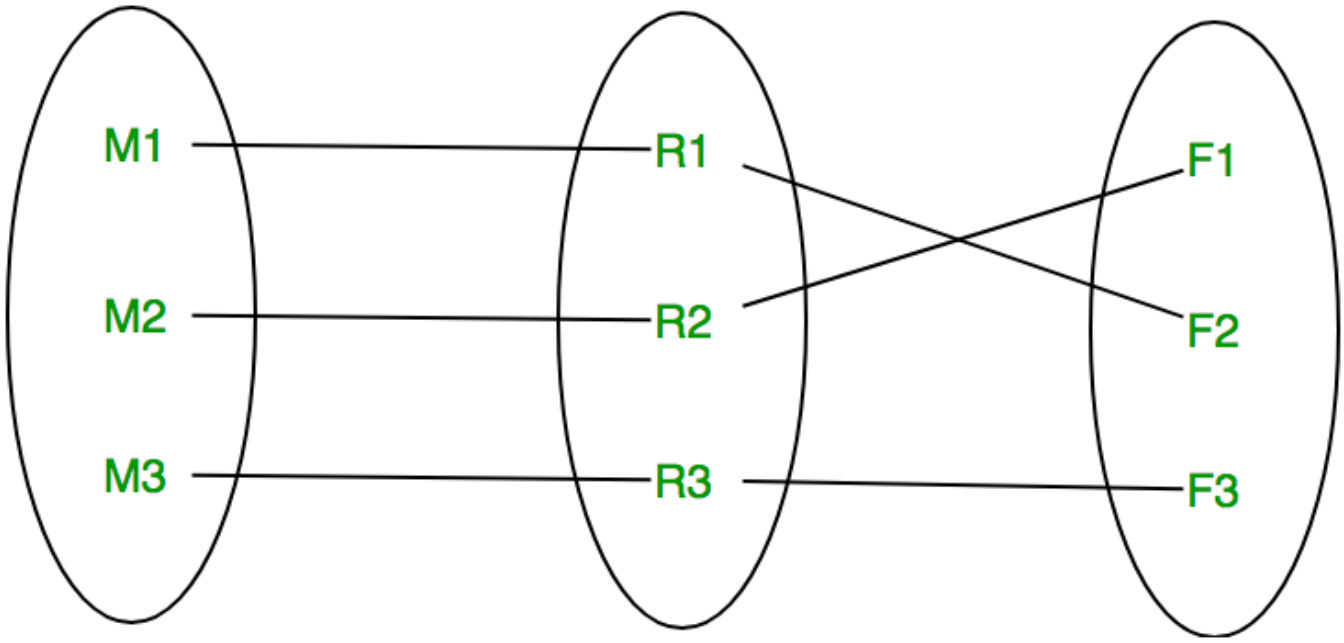


3. n-ary Relationship –
When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

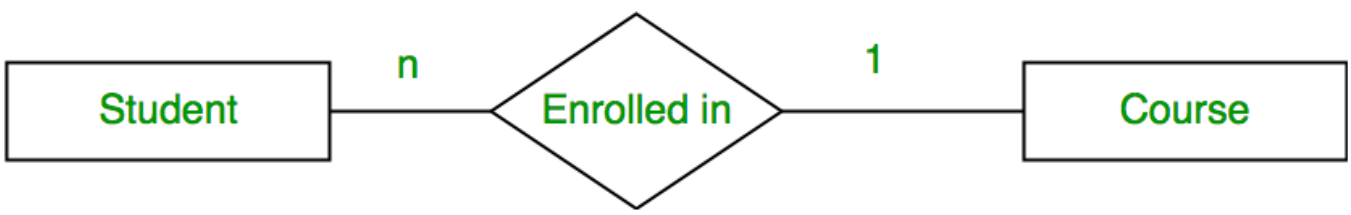
Cardinality:
The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:
1. One to one – When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us 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.



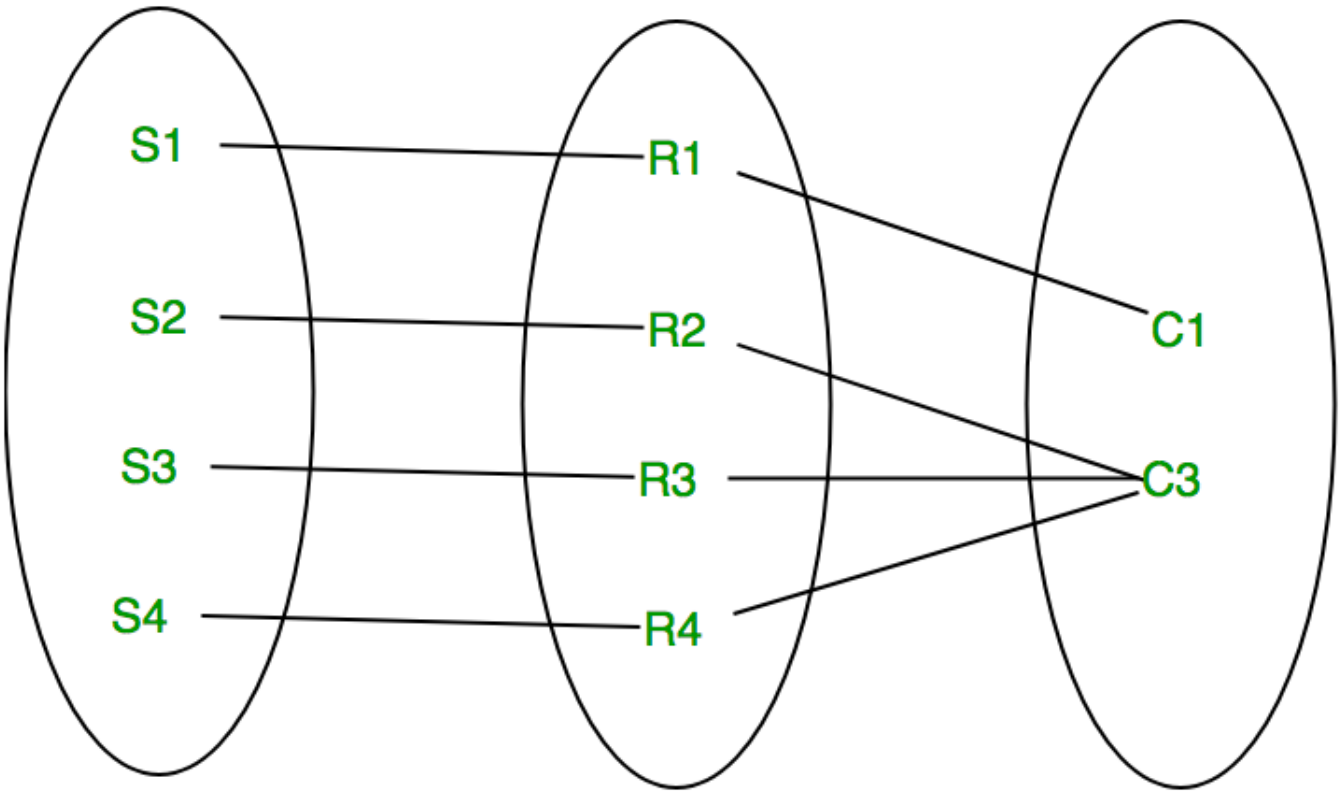
Using Sets, it can be represented as:



2. 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.

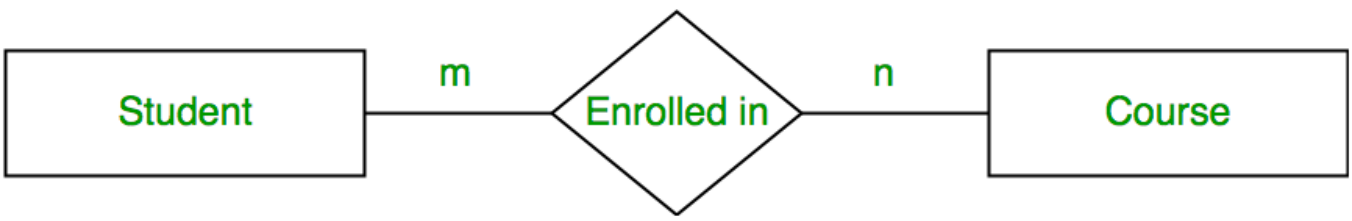


Using Sets, it can be represented as:

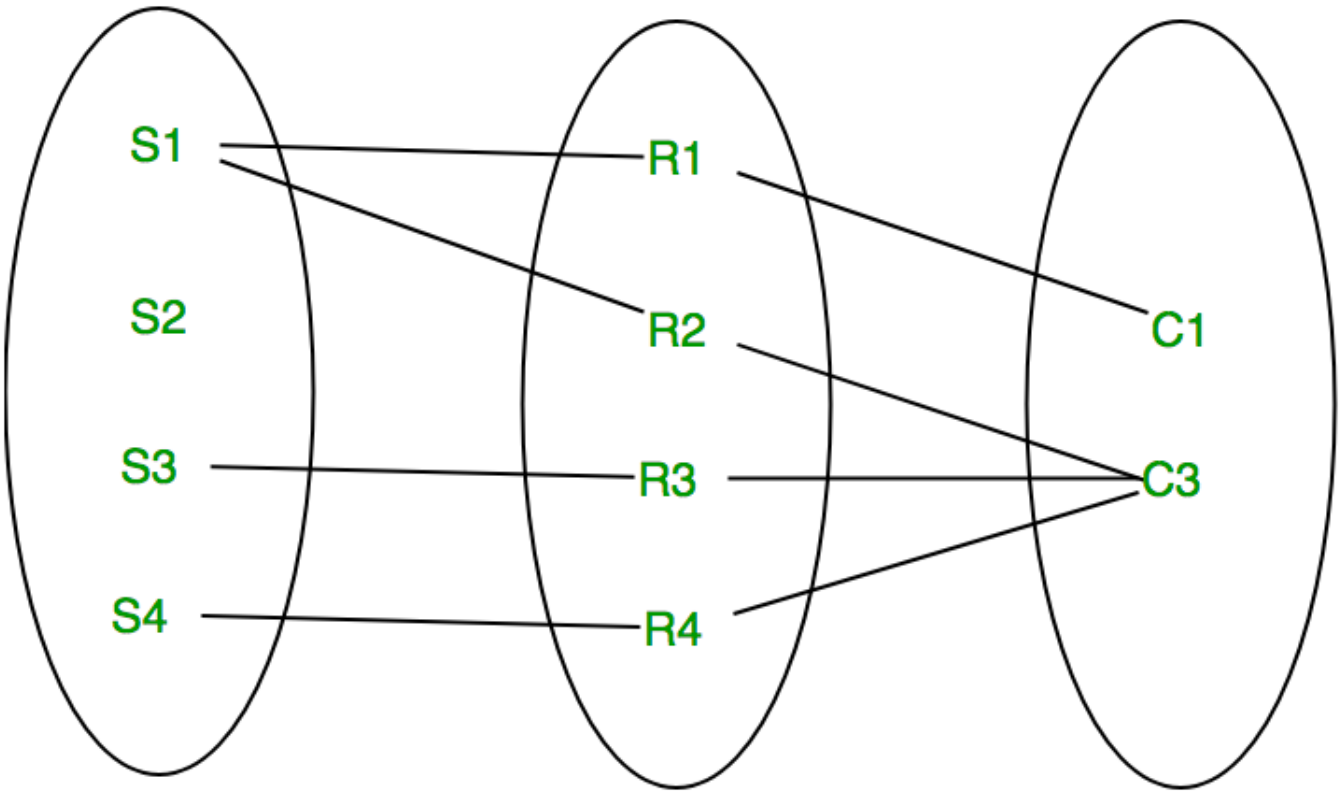


In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. 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.

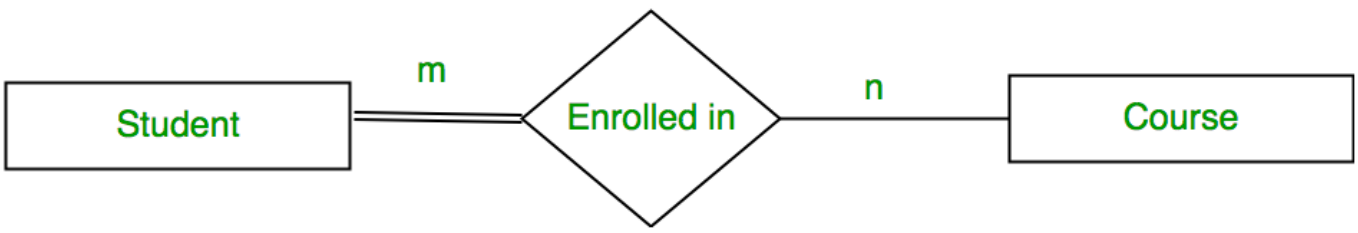


Using sets, it can be represented as:

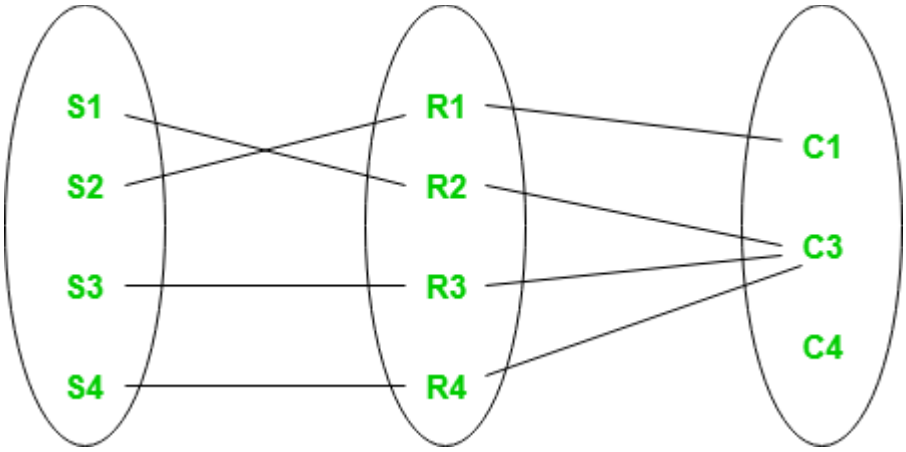


In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

Participation Constraint:
Participation Constraint is applied on the entity participating in the relationship set.
1. 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.
2. 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.



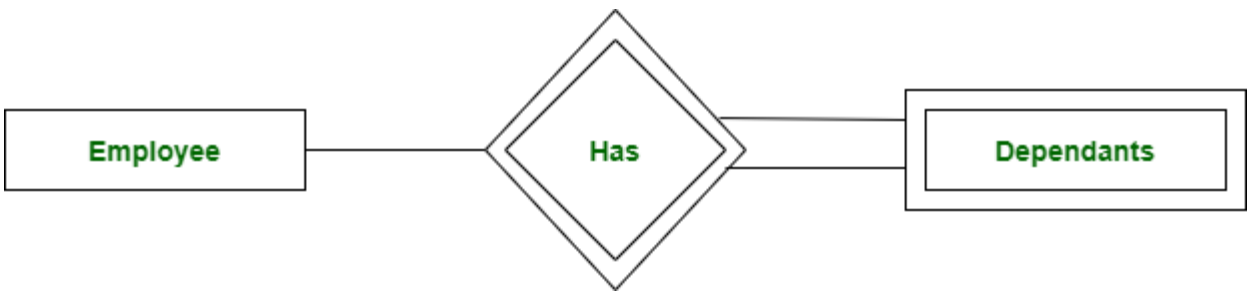
Using set, it can be represented as,



Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

Weak Entity Type and Identifying Relationship:
As discussed before, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called Weak Entity type.
For example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependent.

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond.



2.3 Entity sets and Relationship sets

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like −

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship

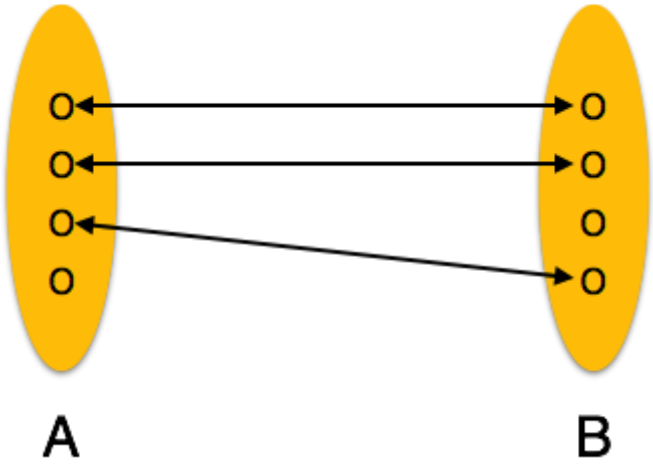
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

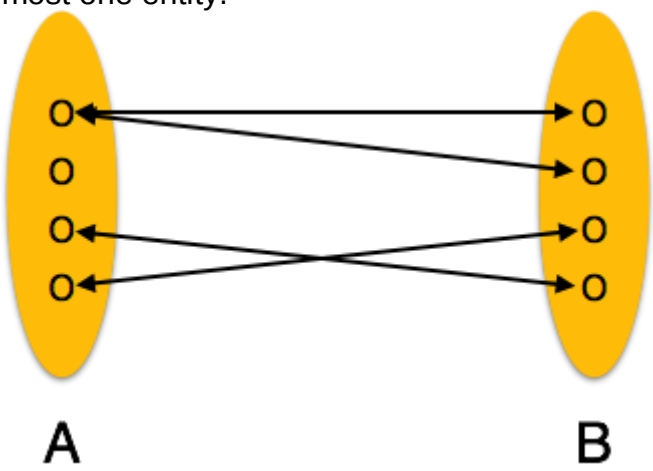
Mapping Cardinalities

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

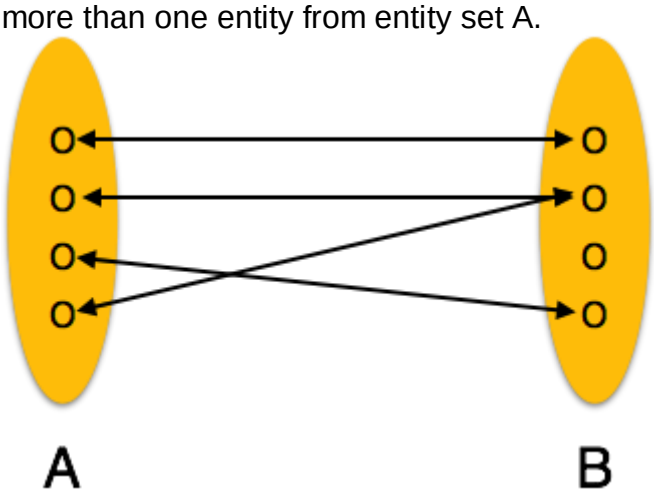
- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



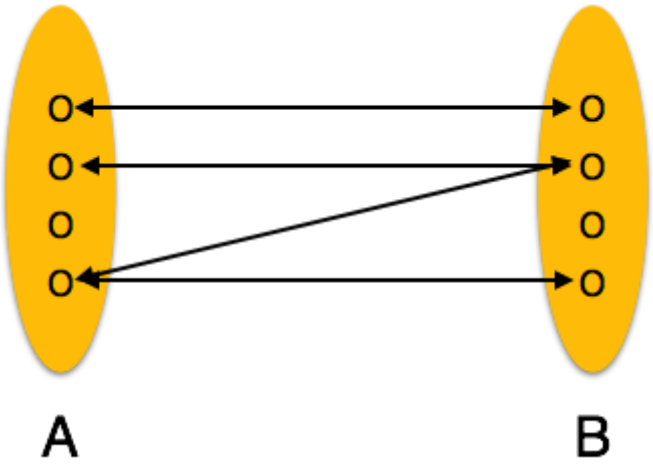
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



2.4 Explain Attributes

An attribute is a property or characteristic of an entity. An entity may contain any number of attributes. One of the attributes is considered as the primary key. In an Entity-Relation model, attributes are represented in an elliptical shape.

Example: Student has attributes like name, age, roll number, and many more. To uniquely identify the student, we use the primary key as a roll number as it is not repeated.
Attributes can also be subdivided into another set of attributes.
There are five such types of attributes: Simple, Composite, Single-valued, Multi-valued, and Derived attribute. One more attribute is their, i.e. Complex Attribute, this is the rarely used attribute.

Simple attribute :

An attribute that cannot be further subdivided into components is a simple attribute.
Example: The roll number of a student, the id number of an employee.

Composite attribute :

An attribute that can be split into components is a composite attribute.

Example: The address can be further split into house number, street number, city, state, country, and pin code, the name can also be split into first name middle name, and last name.

Single-valued attribute :

The attribute which takes up only a single value for each entity instance is a single-valued attribute.

Example: The age of a student.

Multi-valued attribute :

The attribute which takes up more than a single value for each entity instance is a multi-valued attribute.

Example: Phone number of a student: Landline and mobile.

Derived attribute :

An attribute that can be derived from other attributes is derived attributes.

Example: Total and average marks of a student.

Complex attribute :

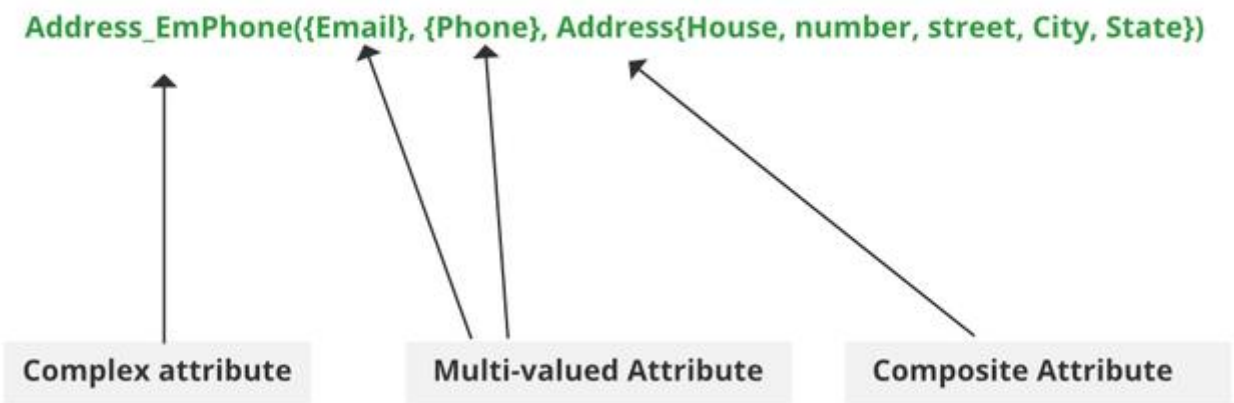
Those attributes, which can be formed by the nesting of composite and multi-valued attributes, are called "*Complex Attributes*". These attributes are rarely used in DBMS(DataBase Management System). That's why they are not so popular.

Representation:

Complex attributes are the nesting of two or more composite and multi-valued attributes. Therefore, these multi-valued and composite attributes are called 'Components' of complex attributes.

These components are grouped between parentheses ‘( )’ and multi-valued attributes between curly braces ‘{ }’, Components are separated by commas ‘, ‘.

For example**:** let us consider a person having multiple phone numbers, emails, and an address.
Here, phone number and email are examples of multi-valued attributes and address is an example of the composite attribute, because it can be divided into house number, street, city, and state.



Complex attributes

Components:
Email, Phone number, Address(All are separated by commas and multi-valued components are represented between curly braces).

Complex Attribute: Address_EmPhone(You can choose any name).
2.5 Mapping constraints

A database must conform to certain defined constraints of the contents.

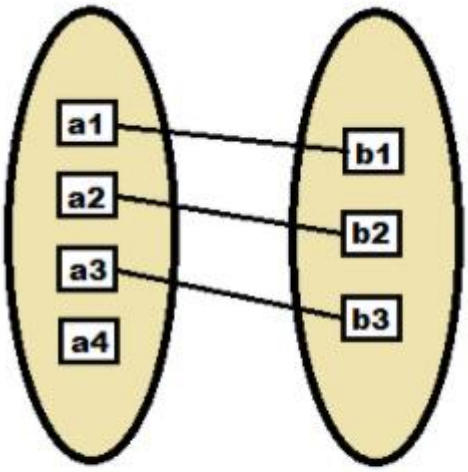
Mapping Cardinalities

Mapping cardinalities is the number of entities to which another entity can be linked through a relationship set.
There are four types of mapping cardinalities.

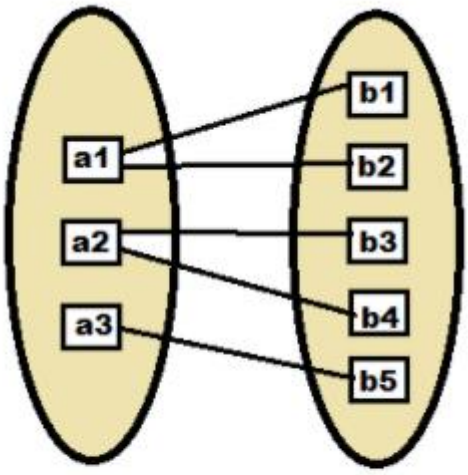
Let A and B be two entity sets.

1. a) **One-to-one** : An entity in A is associated with no more than one entity in B and an entity in B is associated with no more than one entity in A.

1. b) **One-to-many** : An entity in A is associated with many entities in B and an entity in B is associated with at most one entity in A.



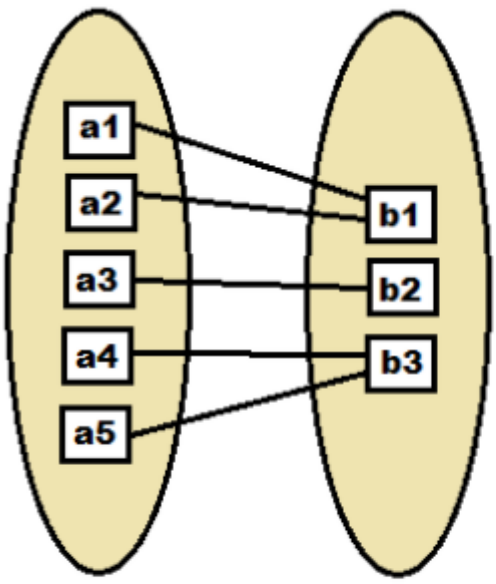
One-to-One



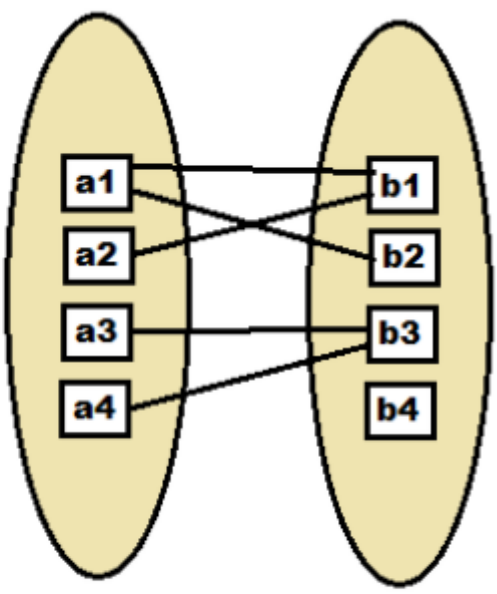
One-to-Many

1. c) **Many-to-one** : Many entities in A can be associated with at most one entity in B and an entity in B can be associated with many entities in A.

1. d) **Many-to-many** : Many entities in A can be associated with many entities in B and many entities in B can be associated with many entities in A.



Many-to-One



Many-to-Many

Participation Constraints :

1. a) **Total** : The participation of an entity set E in a relationship R is considered ‘total’ if every entity in E participates in R.

1. b) **Partial :** If only some entities in E participate in relationship R, then it is called ‘partial’.

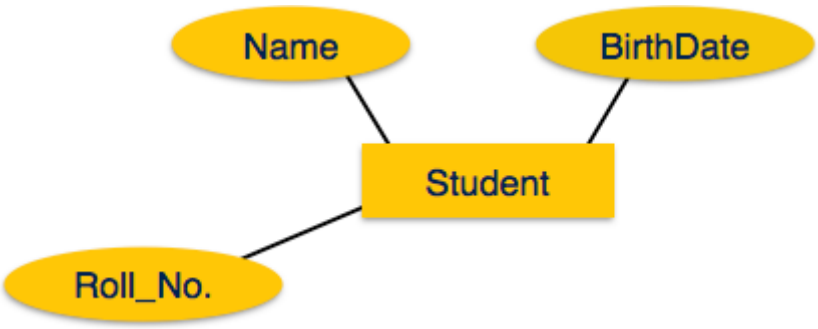
Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

Entity

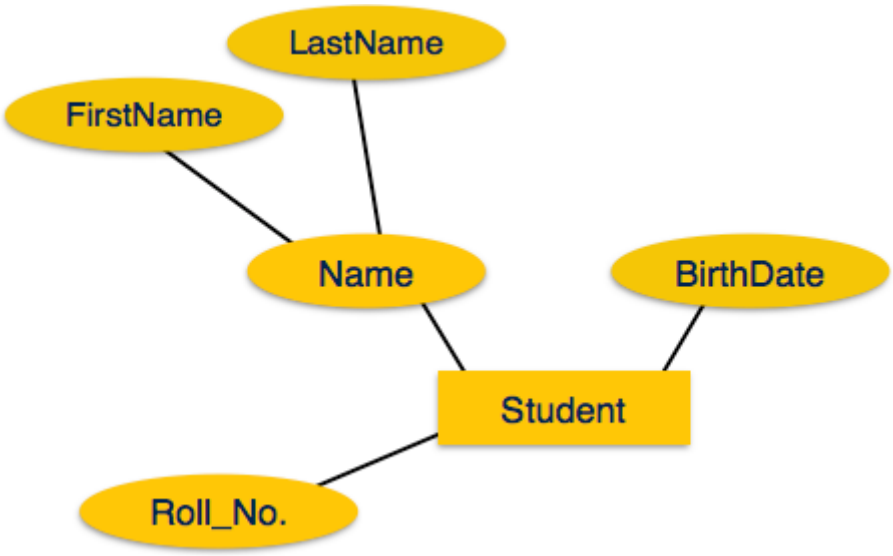
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

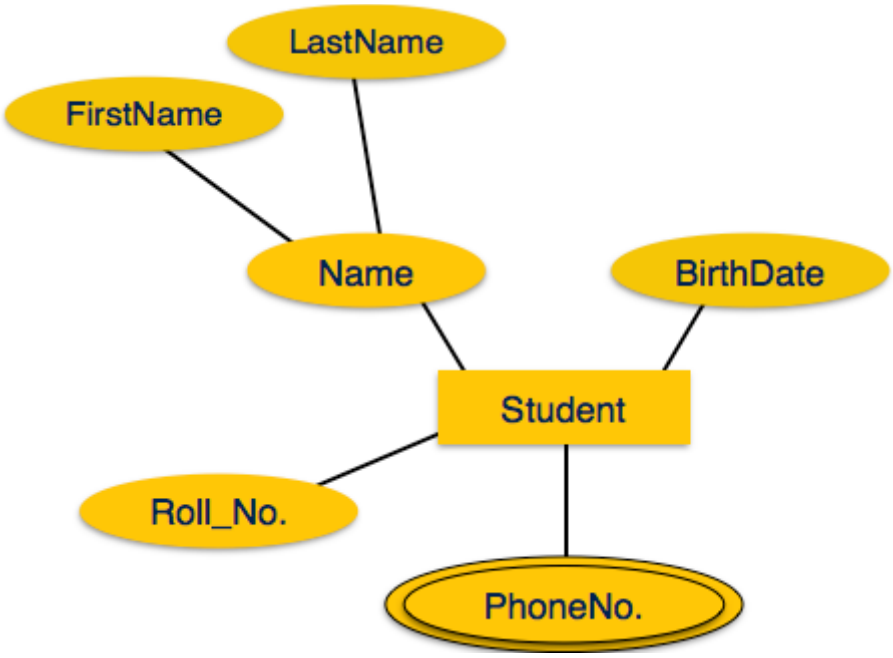
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



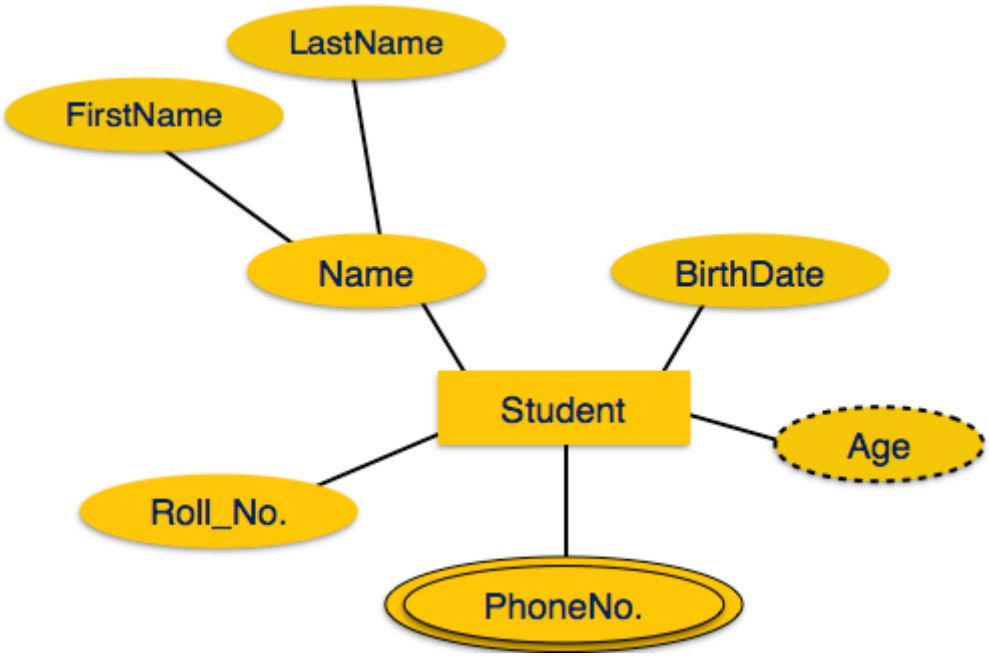
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



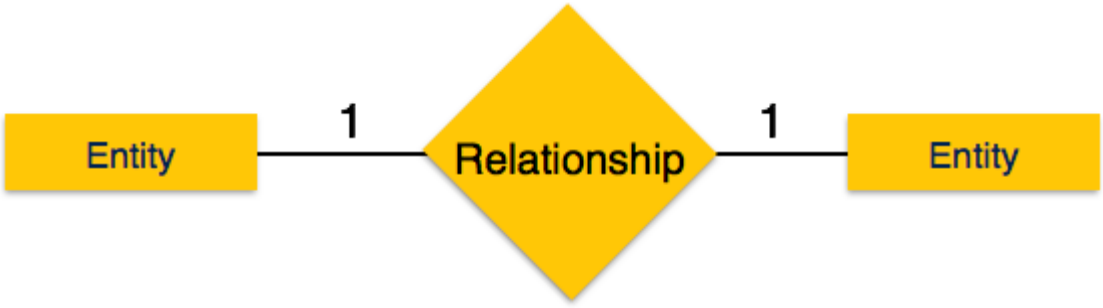
Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

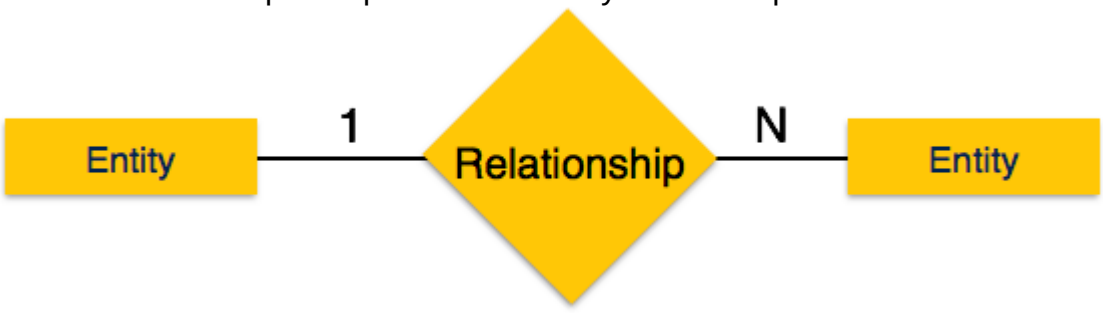
Binary Relationship and Cardinality

A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

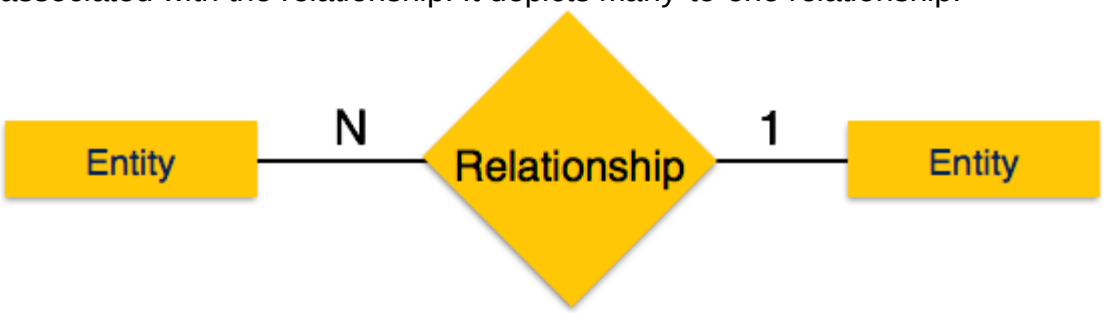
- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



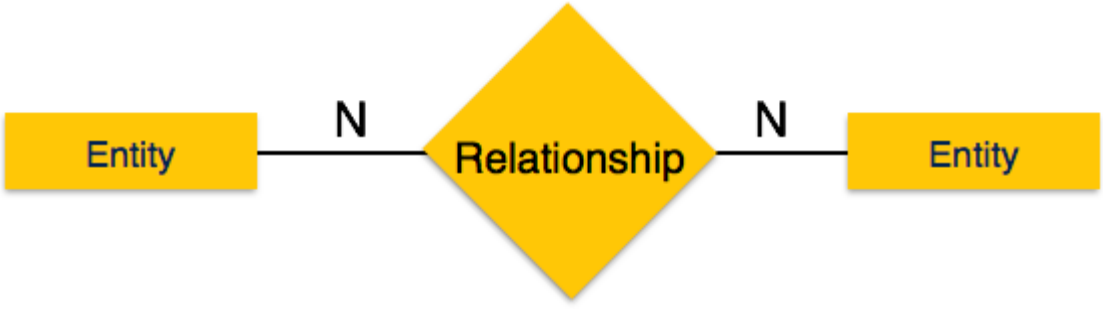
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

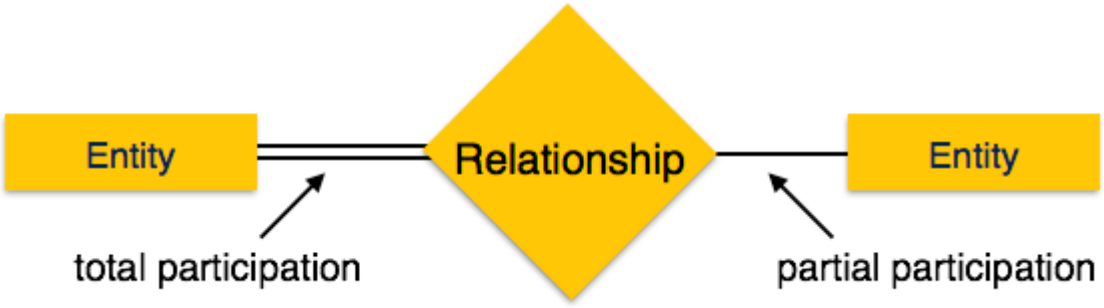


- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Participation Constraints

- **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.



2.7 Relational model

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS
Ram	14795	7305758992	Noida
Shyam	12839	9026288936	Delhi
Laxman	33289	8583287182	Gurugram
Mahesh	27857	7086819134	Ghaziabad
Ganesh	17282	9028 9i3988	Delhi

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

Properties of Relations

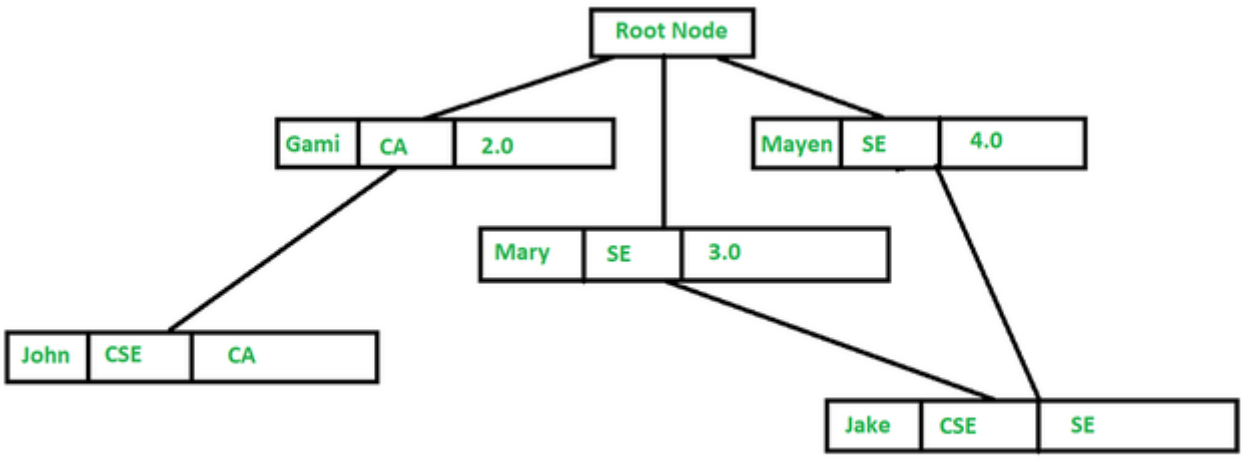
- Name of the relation is distinct from all other relations.

- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Hierarchical Model :
This is one of the oldest models in a data model which was developed by IBM, in the 1950s. In a hierarchical model, data are viewed as a collection of tables, or we can say segments that form a hierarchical relation. In this, the data is organized into a tree-like structure where each record consists of one parent record and many children. Even if the segments are connected as a chain-like structure by logical associations, then the instant structure can be a fan structure with multiple branches. We call the illogical associations as directional associations.
In the hierarchical model, segments pointed to by the logical association are called the **child segment** and the other segment is called the **parent segment**. If there is a segment without a parent is then that will be called the **root** and the segment which has no children are called the **leaves**. The main disadvantage of the hierarchical model is that it can have one-to-one and one-to-many relationships between the nodes.
Applications of hierarchical model :

- Hierarchical models are generally used as semantic models in practice as many real-world occurrences of events are hierarchical in nature like biological structures, political, or social structures.
- Hierarchical models are also commonly used as physical models because of the inherent hierarchical structure of the disk storage system like tracks, cylinders, etc. There are various examples such as Information Management System (IMS) by IBM, NOMAD by NCSS, etc.

Example 1: Consider the below Student database system hierarchical model.



Hierarchical model

In the above-given figure, we have few students and few course-enroll and a course can be assigned to a single student only, but a student can enroll in any number of courses and with this the relationship becomes one-to-many. We can represent the given hierarchical model like the below relational tables:

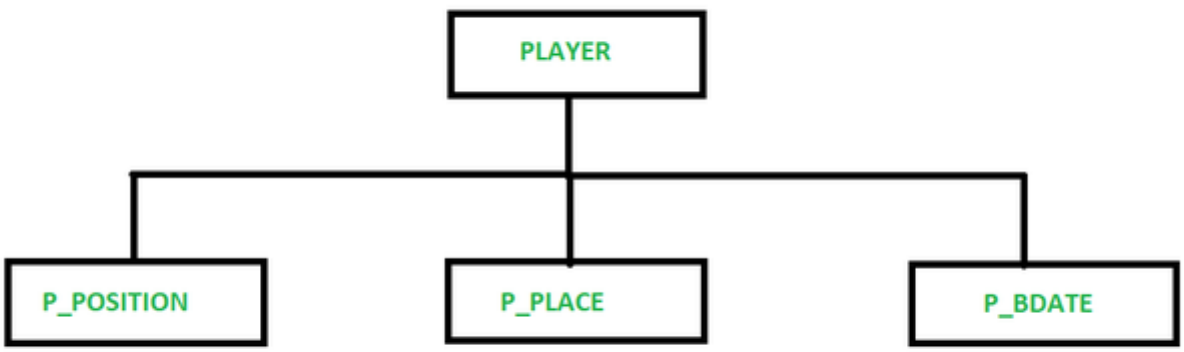
FACULTY Table

Name	Dep	Course-taught
John	CSE	CA
Jake	CSE	SE
Royal	CSE	DBMS

STUDENT Table

Name	Course-enroll	Grade
Gami	CA	2.0
Mary	SE	3.0
Mayen	SE	4.0

Example 2: Consider the below cricket database system hierarchical model scheme.



Hierarchical model

Here, in this example, for each player, there are some set of positions (P_POSITION) he plays, a set of places (P_PLACE), and also a set of birthdates (P_BDATE) of the players. In the above figure, each node represents a logical record type and is displayed by a list of its fields. The child node represents a set of records that are connected to each record of the parent type, which is due to a many-to-many relationship is from child to parent. In the above, figure, the root node PLAYER states that for every player there will be a set of positions, a set of places (only one), and a set of birthdates (which is only one).

Advantages of the hierarchical model :

- As the database is based on this architecture the relationships between various layers are logically simple so, it has a very simple hierarchical database structure.
- It has data sharing as all data are held in a common database data and therefore sharing of data becomes practical.
- It offers data security and this model was the first database model that offered data security.

- There's also data integrity as it is based on the parent-child relationship and also there's always a link between the parents and the child segments.

Disadvantages of the hierarchical model :

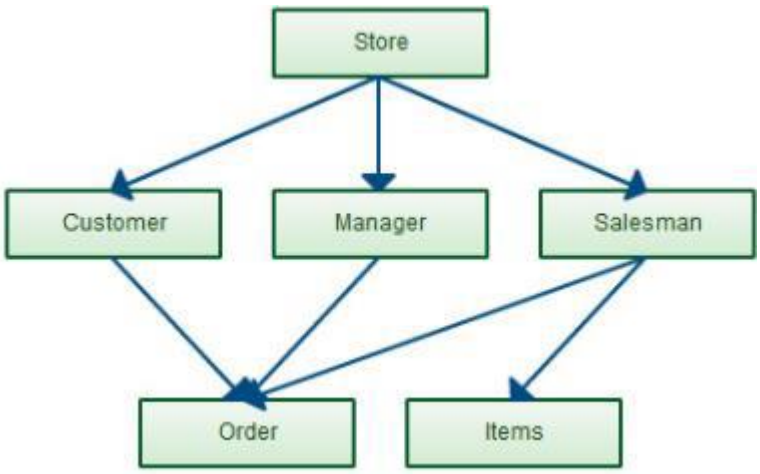
- Even though this model is conceptually simple and easy to design at the same time it is quite complex to implement.
- This model also lacks flexibility as the changes in the new tables or segments often yield very complex system management tasks. Here, a deletion of one segment can lead to the involuntary deletion of all segments under it.
- It has no standards as the implementation of this model does not provide any specific standard.
- It is also limited as many of the common relationships do not conform to the 1 to N format as required by the hierarchical model.

2.9 Network model

The network model was created to represent complex data relationships more effectively when compared to hierarchical models, to improve database performance and standards.

It has entities which are organized in a graphical representation and some entities are accessed through several paths. A User perceives the network model as a collection of records in 1:M relationships.

Given below is the pictorial representation of the network model in DBMS −



Features

The features of a Network Model are as follows −

- **Ability to Merge Relationships** − In this model, because of more relationships the data is more related. It has an ability to manage one-to-one relationships as well as many-to-many relationships.
- **Many paths** − There can be more than one path to the same record because of more relationships. It makes data access fast and simple.
- **Circular Linked List** − The operations in this model are done with the help of the circular linked list. The current position is maintained with the help of a program and navigates through the records based on relationships.

Advantages

The advantages of network model are as follows:

- Network models represent complex data relationships better than the hierarchical models.
- It handles so many relationship types.
- Data access is more flexible than hierarchical models.

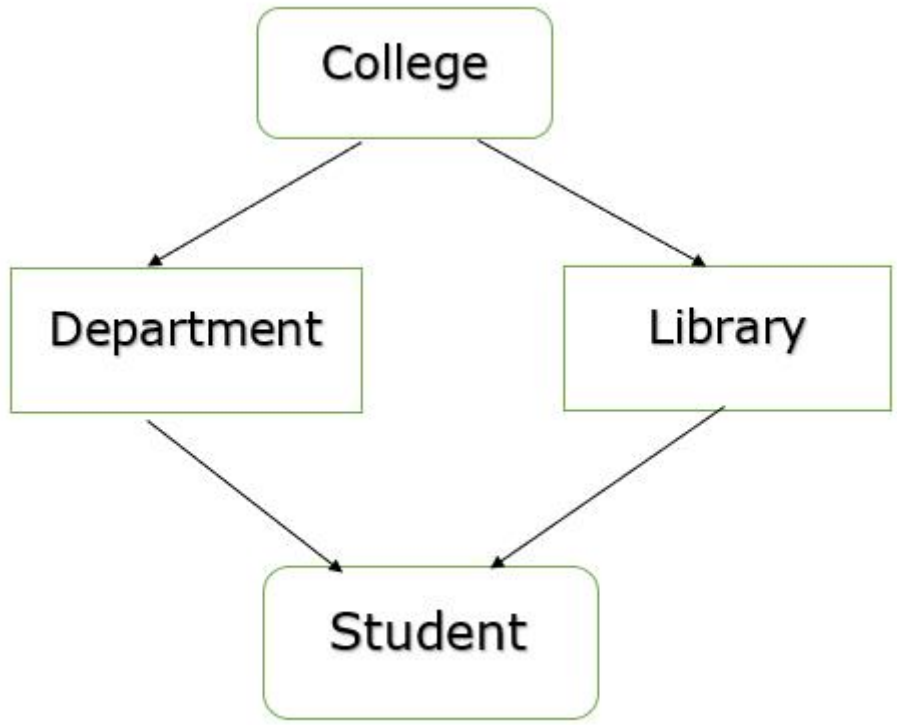
- Improved database performance.
- It includes Data Definition Language (DDL) and Data Manipulation Language (DML) commands.

Disadvantages

The disadvantages of network model are as follows:

- Database contains a complex array of pointers.
- System complexity limits efficiency.
- Structural changes require changes in all application programs.
- Navigation systems yield complex implementation and management.
- Keep heavy pressure on programmers due to the complex structure.
- Any change like updating, deletion, insertion is very complex.

Let’s have a look on another example on network model, as shown below −



3.0 RELATIONAL DATABASE

A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other. Relationships are a logical connection between different tables, established on the basis of interaction among these tables.

Relational databases are among the most popular types of databases because they allow users to easily understand data points that are related to each other in one or more tables and provide relational operators such as structured query language (SQL) to query and manipulate the data. As a result, they are often used for a well-structured data model or for transactional queries in which the data structure does not change often.

A relational database **includes tables containing rows and columns**. For example, a typical business order entry database would include a table that describes a customer with columns for name, address, phone number and so forth.

3.1 Relational algebra

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

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

3.2 Different operators select, project, join , simple Examples

We will discuss all these operations in the following sections.

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 −

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

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

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

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

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

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 − $\prod_{A_1, A_2, A_n}(r)$

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

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

For example −

$\prod_{subject,\ author}$ (Books)

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

Union Operation (∪)

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

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

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.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (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**.

$\prod_{author}$ (Books) − $\prod_{author}$ (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 −

$r X s = \{ q\ t \mid q \in r \text{ and } t \in s\}$

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

Output − 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)

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

4.0 NORMALIZATION IN RELATIONAL SYSTEM

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

1. It is used to remove the duplicate data and database anomalies from the relational table.
2. Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
3. It is helpful to divide the large database table into smaller tables and link them using relationship.
4. It avoids duplicate data or no repeating groups into a table.
5. 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

4.1 Functional Dependencies

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

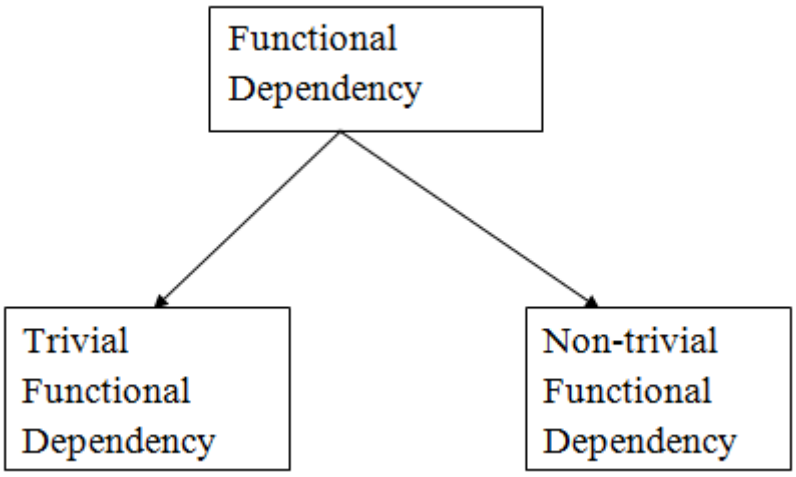
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB

4.2 Lossless join

Lossless-join decomposition is a process in which a relation is decomposed into two or more relations. This property guarantees that the extra or less tuple generation problem does not occur and no information is lost from the original relation during the decomposition. It is also known as non-additive join decomposition.

When the sub relations combine again then the new relation must be the same as the original relation was before decomposition.

Consider a relation R if we decomposed it into sub-parts relation R1 and relation R2.

The decomposition is lossless when it satisfies the following statement −

- If we union the sub Relation R1 and R2 then it must contain all the attributes that are available in the original relation R before decomposition.
- Intersections of R1 and R2 cannot be Null. The sub relation must contain a common attribute. The common attribute must contain unique data.

The common attribute must be a super key of sub relations either R1 or R2.

Here,

R = (A, B, C)

R1 = (A, B)

R2 = (B, C)

The relation R has three attributes A, B, and C. The relation R is decomposed into two relation R1 and R2. . R1 and R2 both have 2-2 attributes.The common attributes are B.

The Value in Column B must be unique. if it contains a duplicate value then the Lossless-join decomposition is not possible.

Draw a table of Relation R with Raw Data −

R (A, B, C)

A	B	C
12	25	34
10	36	09
12	42	30

It decomposes into the two sub relations −

R1 (A, B)

A	B
12	25
10	36
12	42

R2 (B, C)

B	C
25	34
36	09
42	30

Now, we can check the first condition for Lossless-join decomposition.

The union of sub relation R1 and R2 is the same as relation R.

$R_1 U R_2 = R$

We get the following result −

A	B	C
12	25	34
10	36	09
12	42	30

The relation is the same as the original relation R. Hence, the above decomposition is Lossless-join decomposition.

Normalization is a process to eliminate the flaws of a database with bad design. A poorly designed database is inconsistent and create issues while adding, deleting or updating information.

The following makes Database Normalization a crucial step in database design process −

Resolving the database anomalies

The forms of Normalization i.e. 1NF, 2NF, 3NF, BCF, 4NF and 5NF remove all the Insert, Update and Delete anomalies.

Insertion Anomaly occurs when you try to insert data in a record that does not exist.

Deletion Anomaly is when a data is to be deleted and due to the poor deign of database, other record also deletes.

Eliminate Redundancy of Data

Storing same data item multiple times is known as Data Redundancy. A normalized table do not have the issue of redundancy of data.

Data Dependency

The data gets stored in the correct table and ensures normalization.

Isolation of Data

A good designed database states that the changes in one table or field do not affect other. This is achieved through Normalization.

Data Consistency

While updating if a record is left, it can led to inconsistent data, Normalization resolves it and ensures Data Consistency.

4.4 Compare First second and third normal forms

First normal form (1NF)

A relation is said to be in **1NF (first normal form)**, if it doesn't contain any multi-valued attribute. In other words you can say that a relation is in 1NF if each attribute contains only atomic(single) value only.

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Let's say a company wants to store the names and contact details of its employees. It creates a table in the database that looks like this:

Emp_Id	Emp_Name	Emp_Address	Emp_Mobile

101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 , 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123, 8123450987

Two employees (`Jon` & `Lester`) have two mobile numbers that caused the `Emp_Mobile` field to have multiple values for these two employees.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the Emp_Mobile values for employees `Jon` & `Lester` violates that rule.

To make the table complies with 1NF we need to create separate rows for the each mobile number in such a way so that none of the attributes contains multiple values.

Emp_Id	Emp_Name	Emp_Address	Emp_Mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222

103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

To learn more about 1NF refer this article: **1NF**

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any **candidate key** of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Let's say a school wants to store the data of teachers and the subjects they teach. They create a table `Teacher` that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Teacher_Id	Subject	Teacher_Age
111	Maths	38
111	Physics	38
222	Biology	38

333	Physics	40
333	Chemistry	40

Candidate Keys: {Teacher_Id, Subject}
Non prime attribute: Teacher_Age

This table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute Teacher_Age is dependent on Teacher_Id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says **“no non-prime attribute is dependent on the proper subset of any candidate key of the table”**.

To make the table complies with 2NF we can disintegrate it in two tables like this:
Teacher_Details table:

Teacher_Id	Teacher_Age
111	38
222	38
333	40

Teacher_Subject table:

Teacher_Id	Subject
111	Maths

111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables are in Second normal form (2NF). To learn more about 2NF refer this guide: **2NF**

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- **Transitive functional dependency** of non-prime attribute on any super key should be removed.

An attribute that is not part of any **candidate key** is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a **super key** of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Let's say a company wants to store the complete address of each employee, they create a table named `Employee_Details` that looks like this:

Emp_Id	Emp_Name	Emp_Zip	Emp_State	Emp_City	Emp_Distric

1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {Emp_Id}, {Emp_Id, Emp_Name}, {Emp_Id, Emp_Name, Emp_Zip}…so on
Candidate Keys: {Emp_Id}
Non-prime attributes: all attributes except Emp_Id are non-prime as they are not part of any candidate keys.

Here, Emp_State, Emp_City & Emp_District dependent on Emp_Zip. Further Emp_zip is dependent on Emp_Id that makes non-prime attributes (Emp_State, Emp_City & Emp_District) transitively dependent on super key (Emp_Id). This violates the rule of 3NF.

To make this table complies with 3NF we have to disintegrate the table into two tables to remove the transitive dependency:

Employee Table:

Emp_Id	Emp_Name	Emp_Zip
1001	John	282005
1002	Ajeet	222008

1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_Zip table:

Emp_Zip	Emp_State	Emp_City	Emp_District
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

4.5 Explain BCNF

BCNF (Boyce Codd Normal Form) is the advanced version of 3NF. A table is in BCNF if every functional dependency X->Y, X is the super key of the table. For BCNF, the table should be in 3NF, and for every FD. LHS is super key.

Example

Consider a relation R with attributes (student, subject, teacher).

Student	Teacher	Subject
Jhansi	P.Naresh	Database
jhansi	K.Das	C
subbu	P.Naresh	Database
subbu	R.Prasad	C

F: { (student, Teacher) -> subject

(student, subject) -> Teacher

Teacher -> subject}

Candidate keys are (student, teacher) and (student, subject).

The above relation is in 3NF [since there is no transitive dependency]. A relation R is in BCNF if for every non-trivial FD X->Y, X must be a key.

The above relation is not in BCNF, because in the FD (teacher->subject), teacher is not a key. This relation suffers with anomalies −

For example, if we try to delete the student Subbu, we will lose the information that R. Prasad teaches C. These difficulties are caused by the fact the teacher is determinant but not a candidate key.

Decomposition for BCNF

Teacher-> subject violates BCNF [since teacher is not a candidate key].

If X->Y violates BCNF then divide R into R1(X, Y) and R2(R-Y).

So R is divided into two relations R1(Teacher, subject) and R2(student, Teacher).

R1

Teacher	Subject
P.Naresh	database
K.DAS	C
R.Prasad	C

R2

Student	Teacher
Jhansi	P.Naresh
Jhansi	K.Das
Subbu	P.Naresh

Student	Teacher
Subbu	R.Prasad

All the anomalies which were present in R, now removed in the above two relations.

Note

BCNF decomposition does not always satisfy dependency preserving property. After BCNF decomposition if dependency is not preserved then we have to decide whether we want to remain in BCNF or rollback to 3NF. This process of rollback is called denormalization.

5.0 STRUCTURED QUERY LANGUAGE

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDBMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

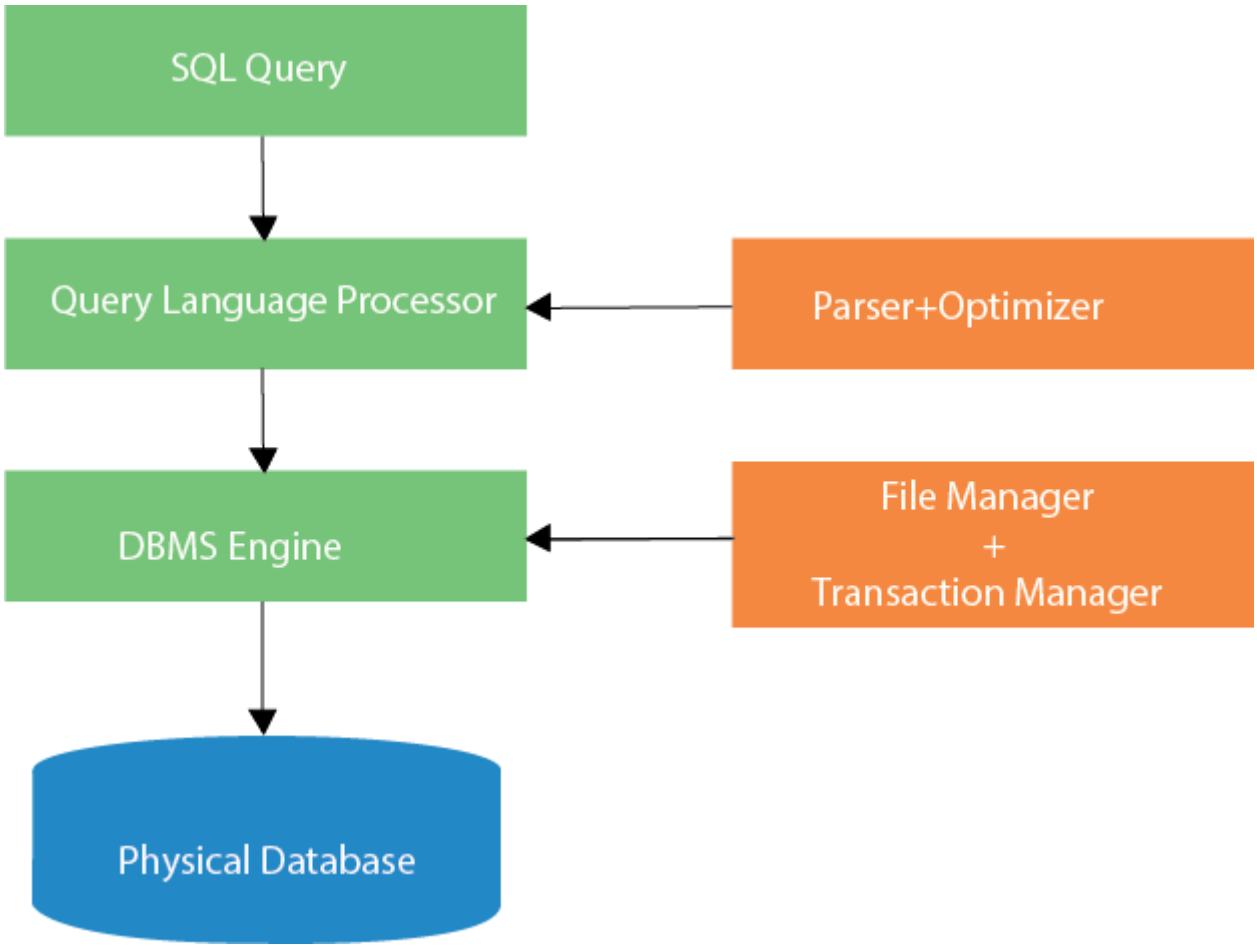
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

Multiple data view

Using the SQL language, the users can make different views of the database structure.

5.1 Elementary idea of Query language

What is SQL?

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS.

You can easily create and manipulate the database, access and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

If you want to get a job in the field of data science, then it is the most important query language to learn. Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back-end.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database management systems.
- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.
- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

History of SQL

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd. They both developed the SQL at the San Jose Research laboratory of IBM Corporation in 1970.

At the end of the 1970s, relational software Inc. developed their own first SQL using the concepts of E.F. Codd, Raymond Boyce, and Donald Chamberlin. This SQL was totally based on RDBMS. Relational Software Inc., which is now known as Oracle Corporation, introduced the Oracle V2 in June 1979, which is the first implementation of SQL language. This Oracle V2 version operates on VAX computers.

Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command
3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

CREATE Command

This command helps in creating the new database, new table, table view, and other objects of the database.

UPDATE Command

This command helps in updating or changing the stored data in the database.

DELETE Command

This command helps in removing or erasing the saved records from the database tables. It erases single or multiple tuples from the tables of the database.

SELECT Command

This command helps in accessing the single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.

DROP Command

This command helps in deleting the entire table, table view, and other objects from the database.

INSERT Command

This command helps in inserting the data or records into the database tables. We can easily insert the records in single as well as multiple rows of the table.

5.3 Simple queries to create, update, insert in SQL

SQL INSERT STATEMENT

SQL INSERT statement is a SQL query. It is used to insert a single or a multiple records in a table.

There are two ways to insert data in a table:

1. By SQL insert into statement
    1. By specifying column names
    2. Without specifying column names
2. By SQL insert into select statement

---

1) Inserting data directly into a table

You can insert a row in the table by using SQL INSERT INTO command.

There are two ways to insert values in a table.

In the first method there is no need to specify the column name where the data will be inserted, you need only their values.

```
INSERT INTO table_name
VALUES (value1, value2, value3....);
```

The second method specifies both the column name and values which you want to insert.

```
INSERT INTO table_name (column1, column2, column3....)
VALUES (value1, value2, value3.....);
```

Let's take an example of table which has five records within it.

```
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (1, ABHIRAM, 22, ALLAHABAD);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (2, ALKA, 20, GHAZIABAD);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (3, DISHA, 21, VARANASI);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
VALUES (4, ESHA, 21, DELHI);
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
```

```
VALUES (5, MANMEET, 23, JALANDHAR);
```

It will show the following table as the final result.

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI
4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR

You can create a record in CUSTOMERS table by using this syntax also.

```
INSERT INTO CUSTOMERS
VALUES (6, PRATIK, 24, KANPUR);
```

The following table will be as follow:

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI
4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR
6	PRATIK	24	KANPUR

---

2) Inserting data through SELECT Statement

SQL INSERT INTO SELECT Syntax

```
INSERT INTO table_name
[(column1, column2, .... column)]
SELECT column1, column2, .... Column N
FROM table_name [WHERE condition];
```

Note: when you add a new row, you should make sure that data type of the value and the column should be matched.

If any integrity constraints are defined for the table, you must follow them.

SQL UPDATE

The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.

SQL UPDATE statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.

The UPDATE statement can be written in following form:

```
UPDATE table_name SET [column_name1= value1,... column_nameN = valueN] [WHERE condition]
```

Let's see the Syntax:

```
UPDATE table_name
SET column_name = expression
WHERE conditions
```

Let's take an example: here we are going to update an entry in the source table.

SQL statement:

```
UPDATE students
SET User_Name = 'beinghuman'
WHERE Student_Id = '3'
```

Source Table:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	jonny

See the result after updating value:

Student_Id	FirstName	LastName	User_Nan
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	**beinghuman**

Updating Multiple Fields:

If you are going to update multiple fields, you should separate each field assignment with a comma.

SQL UPDATE statement for multiple fields:

```
UPDATE students
SET User_Name = 'beserious', First_Name = 'Johnny'
WHERE Student_Id = '3'
```

Result of the table is given below:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	**Johnny**	Walker	**beserious**

MYSQL SYNTAX FOR UPDATING TABLE:

```
UPDATE table_name
SET field1 = new-value1, field2 = new-value2,
[WHERE CLAUSE]
```

SQL UPDATE SELECT:

SQL UPDATE WITH SELECT QUERY:

We can use SELECT statement to update records through UPDATE statement.

SYNTAX:

```
UPDATE tableDestination
SET tableDestination.col = value
WHERE EXISTS (
SELECT col2.value
FROM  tblSource
WHERE tblSource.join_col = tblDestination. Join_col
AND  tblSource.Constraint = value)
```

You can also try this one -

```
UPDATE
Table
SET
Table.column1 = othertable.column 1,
Table.column2 = othertable.column 2
FROM
Table
INNER JOIN
Other_table
ON
```

```
Table.id = other_table.id
```

My SQL SYNTAX:

If you want to UPDATE with SELECT in My SQL, you can use this syntax:

Let's take an example having two tables. Here,

First table contains -

Cat_id, cat_name,

And the second table contains -

Rel_cat_id, rel_cat_name

SQL UPDATE COLUMN:

We can update a single or multiple columns in SQL with SQL UPDATE query.

SQL UPDATE EXAMPLE WITH UPDATING SINGLE COLUMN:

```
UPDATE students
SET student_id = 001
WHERE student_name = 'AJEET';
```

This SQL UPDATE example would update the student_id to '001' in the student table where student_name is 'AJEET'.

SQL UPDATE EXAMPLE WITH UPDATING MULTIPLE COLUMNS:

To update more than one column with a single update statement:

```
UPDATE students
SET student_name = 'AJEET',
Religion = 'HINDU'
WHERE student_name = 'RAJU';
```

This SQL UPDATE statement will change the student name to 'AJEET' and religion to 'HINDU' where the student name is 'RAJU'.

SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

```
create table "tablename"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

The data type of the columns may vary from one database to another. For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

Let us take an example to create a STUDENTS table with ID as primary key and NOT NULL are the constraint showing that these fields cannot be NULL while creating records in the table.

```
SQL> CREATE TABLE STUDENTS (
ID INT              NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT             NOT NULL,
ADDRESS CHAR (25),
PRIMARY KEY (ID)
);
```

You can verify it, if you have created the table successfully by looking at the message displayed by the SQL Server, else you can use DESC command as follows:

SQL> DESC STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT
ID	Int(11)	NO	PRI	
NAME	Varchar(20)	NO		
AGE	Int(11)	NO		
ADDRESS	Varchar(25)	YES		NULL

4 rows in set (0.00 sec)

Now you have the STUDENTS table available in your database and you can use to store required information related to students.

SQL CREATE TABLE Example in MySQL

Let's see the command to create a table in MySQL database.

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(255),
LastName varchar(255),
Email varchar(255),
AddressLine varchar(255),
City varchar(255)
);
```

SQL CREATE TABLE Example in Oracle

Let's see the command to create a table in Oracle database.

```
CREATE TABLE Employee
(
EmployeeID number(10),
FirstName varchar2(255),
LastName varchar2(255),
Email varchar2(255),
AddressLine varchar2(255),
City varchar2(255)
);
```

SQL CREATE TABLE Example in Microsoft SQLServer

Let's see the command to create a table in SQLServer database. It is same as MySQL and Oracle.

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(255),
LastName varchar(255),
Email varchar(255),
AddressLine varchar(255),
City varchar(255)
);
```

Create a Table using another table

We can create a copy of an existing table using the create table command. The new table gets the same column signature as the old table. We can select all columns or some specific columns.

If we create a new table using an old table, the new table will be filled with the existing value from the old table.

The basic syntax for creating a table with the other table is:

```
CREATE TABLE table_name  AS
SELECT column1, column2,...
FROM old_table_name WHERE ..... ;
The following SQL creates a copy of the employee table.
CREATE TABLE EmployeeCopy AS
SELECT EmployeeID, FirstName, Email
FROM Employee;
```

SQL Primary Key with CREATE TABLE Statement

The following query creates a PRIMARY KEY on the "D" column when the "Employee" table is created.

MySQL

```
CREATE TABLE Employee(
EmployeeID NOT NULL,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255),
PRIMARY KEY (EmployeeID)
);
```

SQL Server / Oracle / MS Access

```
CREATE TABLE Employee(
EmployeeID NOT NULL PRIMARY KEY,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255)
);
```

Use the following query to define a PRIMARY KEY constraints on multiple columns, and to allow naming of a PRIMARY KEY constraints.

For MySQL / SQL Server /Oracle / MS Access

```
CREATE TABLE Employee(
EmployeeID NOT NULL,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255),
CONSTRAINT     PK_Employee PRIMARY KEY (EmployeeID, FirstName)
);
```

6.0 TRANSACTION PROCESSING CONCEPTS

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

6.1 Idea about transaction processing

A transaction is a program including a collection of database operations, executed as a logical unit of data processing. The operations performed in a transaction include one or more of database operations like insert, delete, update or retrieve data. It is an atomic process that is either performed into completion entirely or is not performed at all. A transaction involving only data retrieval without any data update is called read-only transaction.

Each high level operation can be divided into a number of low level tasks or operations. For example, a data update operation can be divided into three tasks −

- **read_item()** − reads data item from storage to main memory.
- **modify_item()** − change value of item in the main memory.
- **write_item()** − write the modified value from main memory to storage.

Database access is restricted to read_item() and write_item() operations. Likewise, for all transactions, read and write forms the basic database operations.

Transaction Operations

The low level operations performed in a transaction are −

- **begin_transaction** − A marker that specifies start of transaction execution.
- **read_item or write_item** − Database operations that may be interleaved with main memory operations as a part of transaction.
- **end_transaction** − A marker that specifies end of transaction.
- **commit** − A signal to specify that the transaction has been successfully completed in its entirety and will not be undone.
- **rollback** − A signal to specify that the transaction has been unsuccessful and so all temporary changes in the database are undone. A committed transaction cannot be rolled back.

6.2 Transaction & system concept

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

- **Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then

the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Equivalence Schedules

An equivalence schedule can be of the following types −

Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example −

- If T reads the initial data in S1, then it also reads the initial data in S2.
- If T reads the value written by J in S1, then it also reads the value written by J in S2.
- If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Conflict Equivalence

Two schedules would be conflicting if they have the following properties −

- Both belong to separate transactions.
- Both accesses the same data item.
- At least one of them is "write" operation.

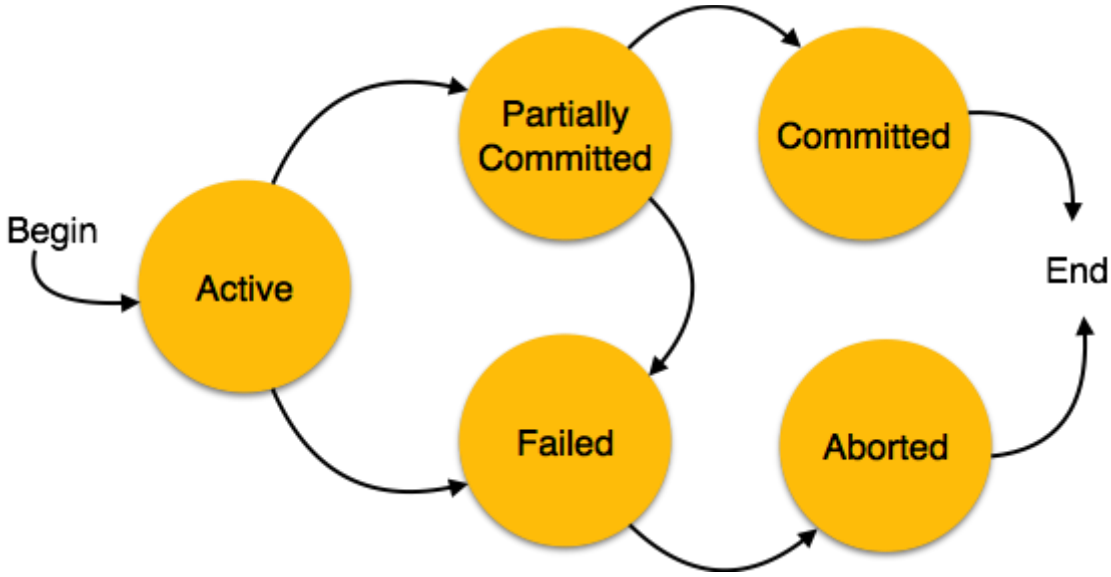
Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

States of Transactions

A transaction in a database can be in one of the following states −



- **Active** − In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
    - Re-start the transaction
    - Kill the transaction
- **Committed** − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

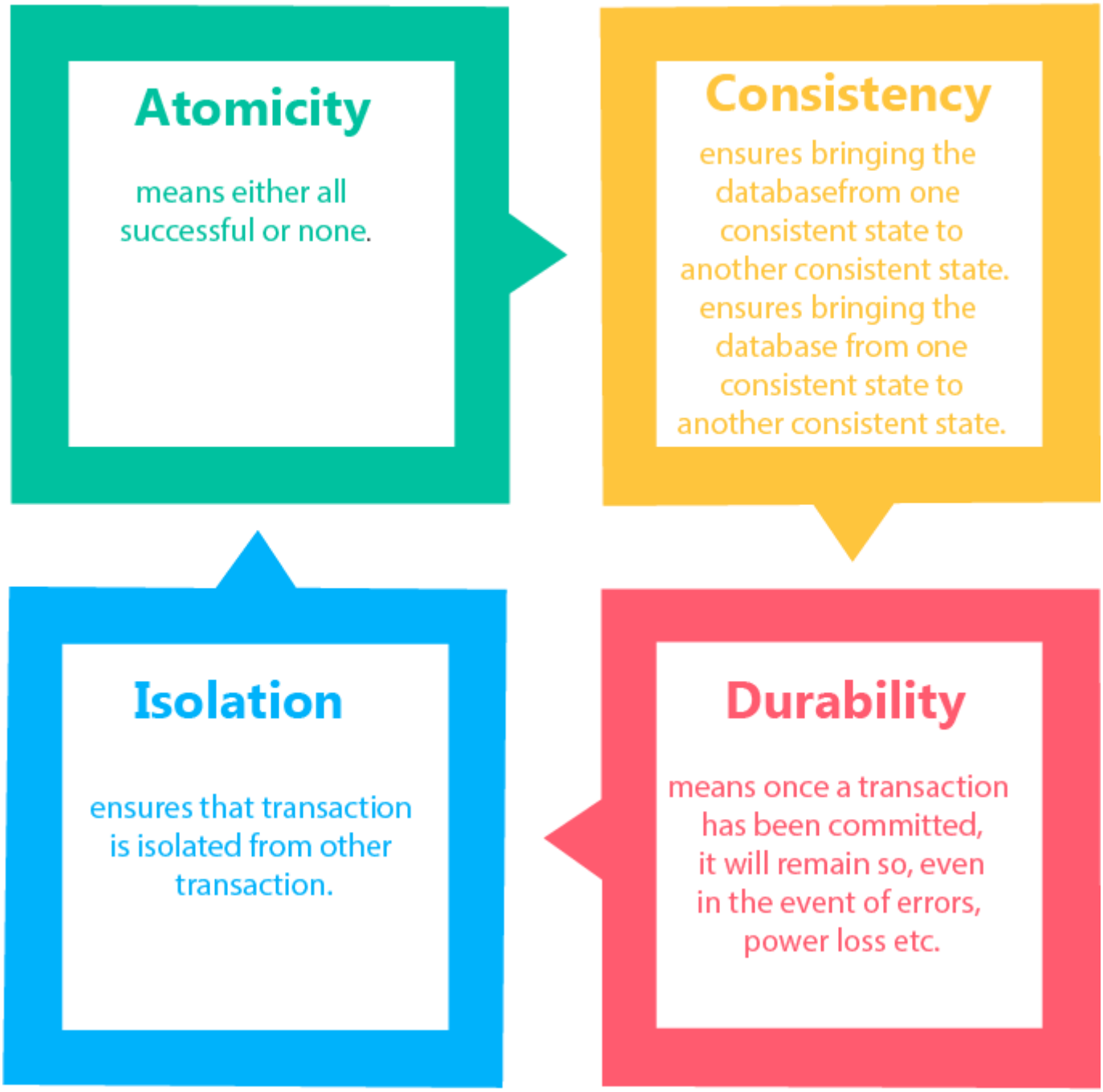
6.3 Desirable properties of transaction

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability



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:= Write(A)	Read(B) Y:= Write(B)

A-100

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.

1. Total before T occurs = 600+300=900
2. 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.

6.4 Schedules and recoverability
Recoverability in DBMS

As discussed, a transaction may not execute completely due to hardware failure, system crash or software issues. In that case, we have to roll back the failed transaction. But some other transaction may also have used values produced by the failed transaction. So we have to roll back those transactions as well.

Recoverable Schedules:

- Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.
  Example 1:
- S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y),
  W2(x), W1(y), **C1**, **C2**;
  Given schedule follows order of **Ti->Tj => C1->C2**. Transaction T1 is executed before T2 hence there is no chances of conflict occur. R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of first transaction performed first update on data item x, hence given schedule is recoverable.
  Example 2: Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
W(A)	
	W(A)
	R(A)
commit	
	commit

This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.

Irrecoverable Schedule:

- The table below shows a schedule with two transactions, T1 reads and writes A and that value is read and written by T2. T2 commits. But later on, T1 fails. So we have to rollback T1. Since T2 has read the value written by T1, it should also be rollbacked. But we have already committed that. So this schedule is irrecoverable schedule. When Tj is reading the value updated by Ti and Tj is

committed before committing of Ti, the schedule will be irrecoverable.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		
Failure Point				
Commit;				

Recoverable with Cascading Rollback:

- The table below shows a schedule with two transactions, T1 reads and writes A and that value is read and written by T2. But later on, T1 fails. So we have to rollback T1. Since T2 has read the value written by T1, it should also be rollbacked. As it has not committed, we can rollback T2 as well. So it is recoverable with cascading rollback. Therefore, if Tj is reading value updated by Ti and commit of Tj is delayed till commit of Ti, the schedule is called recoverable with cascading rollback.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
Failure Point				
Commit;				
		Commit;		

Cascadeless Recoverable Rollback:

- The table below shows a schedule with two transactions, T1 reads and writes A and commits and that value is read by T2. But if T1 fails before commit, no other transaction has read its value, so there is no need to rollback other transaction. So this is a Cascadeless recoverable schedule. So, if Tj reads value updated by Ti only after Ti is committed, the schedule will be cascadeless recoverable.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
Commit;				
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		

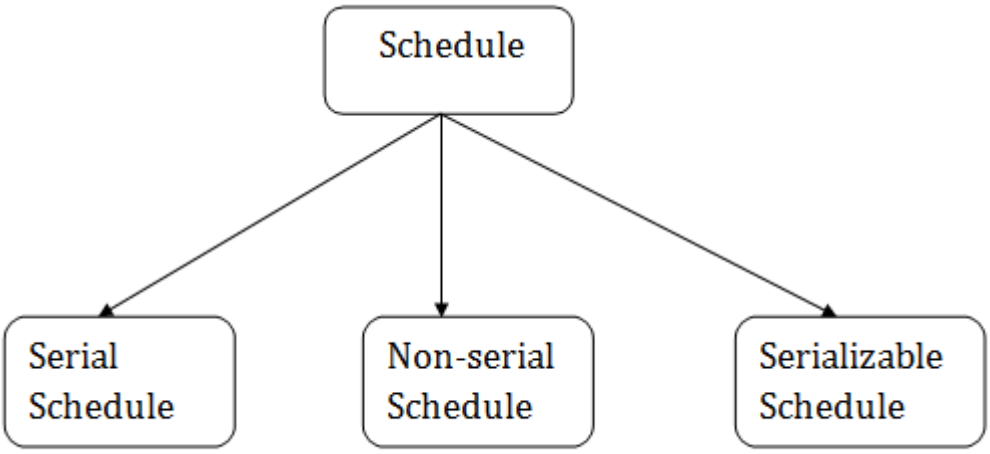
Question: Which of the following scenarios may lead to an irrecoverable error in a database system?

1. A transaction writes a data item after it is read by an uncommitted transaction.
2. A transaction reads a data item after it is read by an uncommitted transaction.
3. A transaction reads a data item after it is written by a committed transaction.
4. A transaction reads a data item after it is written by an uncommitted transaction.

Answer: See the example discussed in Table 1, a transaction is reading a data item after it is written by an uncommitted transaction, the schedule will be irrecoverable.

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:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. 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

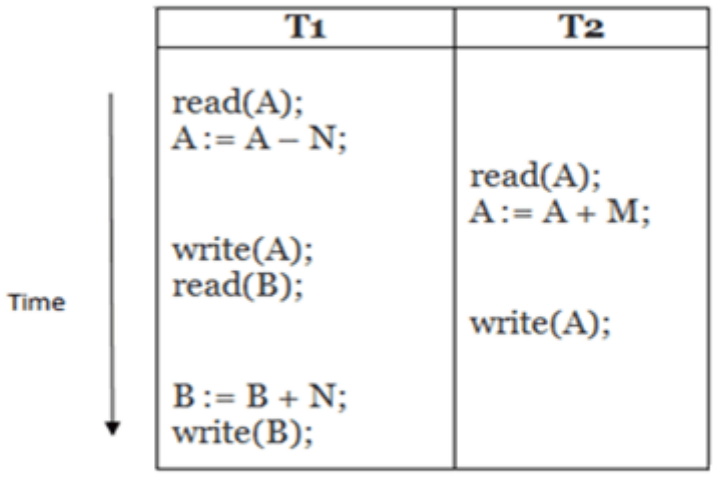
T1	T2

	read(A);
	A := A + M;
	write(A);

read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);	

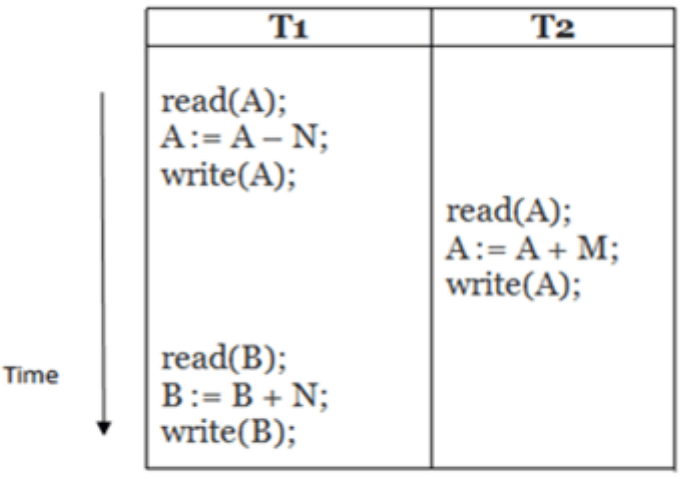
Schedule B

(c)



Schedule C

(d)



Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

7.0 CONCURRENCY CONTROL CONCEPTS

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- 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.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.

- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

7.1 Basic concepts,

Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems.

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Advantages

The advantages of concurrency control are as follows −

- Waiting time will be decreased.
- Response time will decrease.
- Resource utilization will increase.
- System performance & Efficiency is increased.

Control concurrency

The simultaneous execution of transactions over shared databases can create several data integrity and consistency problems.

For example, if too many people are logging in the ATM machines, serial updates and synchronization in the bank servers should happen whenever the transaction is done, if not it gives wrong information and wrong data in the database.

Main problems in using Concurrency

The problems which arise while using concurrency are as follows −

- **Updates will be lost** − One transaction does some changes and another transaction deletes that change. One transaction nullifies the updates of another transaction.
- **Uncommitted Dependency or dirty read problem** − On variable has updated in one transaction, at the same time another transaction has started and deleted the value of the variable there the variable is not getting updated or committed that has been done on the first transaction this gives us false values or the previous values of the variables this is a major problem.

- **Inconsistent retrievals** − One transaction is updating multiple different variables, another transaction is in a process to update those variables, and the problem occurs is inconsistency of the same variable in different instances.

Concurrency control techniques

The concurrency control techniques are as follows −

Locking

Lock guaranties exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock.

Types of Locks

The types of locks are as follows −

- Shared Lock [Transaction can read only the data item values]
- Exclusive Lock [Used for both read and write data item values]

Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

Optimistic

It is based on the assumption that conflict is rare and it is more efficient to allow transactions to proceed without imposing delays to ensure serializability.

7.2 Locks, Live Lock, Dead Lock,

LOCKS

Locks are **an integral part to maintain concurrency control in DBMS**. A transaction in any system implementing lock based concurrency control cannot read or write a statement until it has obtained the required locks. There are two types of locks in Lock based protocols.

1. Binary Locks - These can only be in one of two states, locked or unlocked.
2. Shared/Exclusive Locks - Shared locks are acquired when only read operation is to be performed. Shared locks can be shared between multiple transactions as there is no data being altered.

LIVE LOCK

Livelock is **a deadlock-like situation in which processes block each other with a repeated state change yet make no progress**. Starvation is the outcome of a deadlock, livelock, or as a result of continuous resource denial to a process. Authors Bottom

DeadLock

In a database, a deadlock is **an unwanted situation in which 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 it brings the whole system to a Halt.

7.3 Serializability (only fundamentals)

A schedule is serialized if it is equivalent to a serial schedule. A concurrent schedule must ensure it is the same as if executed serially means one after another. It refers to the sequence of actions such as read, write, abort, commit are performed in a serial manner.

Example

Let’s take two transactions T1 and T2,

If both transactions are performed without interfering each other then it is called as serial schedule, it can be represented as follows −

T1	T2
READ1(A)	
WRITE1(A)	
READ1(B)	
C1	
	READ2(B)
	WRITE2(B)
	READ2(B)
	C2

Non serial schedule − When a transaction is overlapped between the transaction T1 and T2.

Example

Consider the following example −

T1	T2
READ1(A)	
WRITE1(A)	
	READ2(B)
	WRITE2(B)
READ1(B)	
WRITE1(B)	
READ1(B)	

Types of serializability

There are two types of serializability −

View serializability

A schedule is view-serializability if it is viewed equivalent to a serial schedule.

The rules it follows are as follows −

- T1 is reading the initial value of A, then T2 also reads the initial value of A.
- T1 is the reading value written by T2, then T2 also reads the value written by T1.
- T1 is writing the final value, and then T2 also has the write operation as the final value.

Conflict serializability

It orders any conflicting operations in the same way as some serial execution. A pair of operations is said to conflict if they operate on the same data item and one of them is a write operation.

That means

- Readi(x) readj(x) - non conflict read-read operation
- Readi(x) writej(x) - conflict read-write operation.
- Writei(x) readj(x) - conflict write-read operation.
- Writei(x) writej(x) - conflict write-write operation.

8.0 SECURITY AND INTEGRITY

Database security refers to **the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability**.

The term 'integrity' means **guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity**

8.1 Authorization and views
Authorization is a privilege provided by the Database Administer. **Users of the database can only view the contents they are authorized to view**. The rest of the database is out of bounds to them.

Authorization is **the process where the database manager gets information about the authenticated user**. Part of that information is determining which database operations the user can perform and which data objects a user can access.

Views can be an important security mechanism for a DBMS. Views are **virtual tables derived (directly or indirectly) from one or more stored ' base" tables, using the relational operators (restrict, project, join) and/or statistical summary**. In R* the semantics of a view are given by an SQL SELECT statement.
Views are **some kind of virtual tables created by original tables from the database**. Views actually do not hold the actual data and just have the definition of the original data. Views act as a proxy or virtual table created from the original table. The view has two primary purposes: Simplifying complex SQL queries.

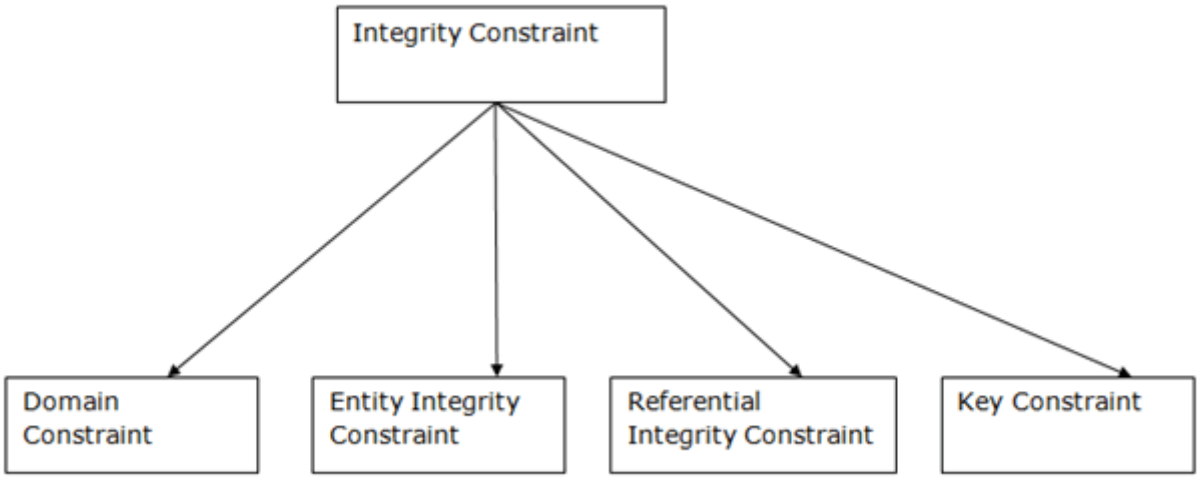
8.2 Security constraints
Security constraints are **rules which assign security levels to the data**. They can be used either as integrity rules, derivation rules or as schema rules (such as data dependencies). If they are used as integrity rules, then they must be satisfied by the data in the multilevel database.

8.3 Integrity Constraints

Integrity Constraints

- 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

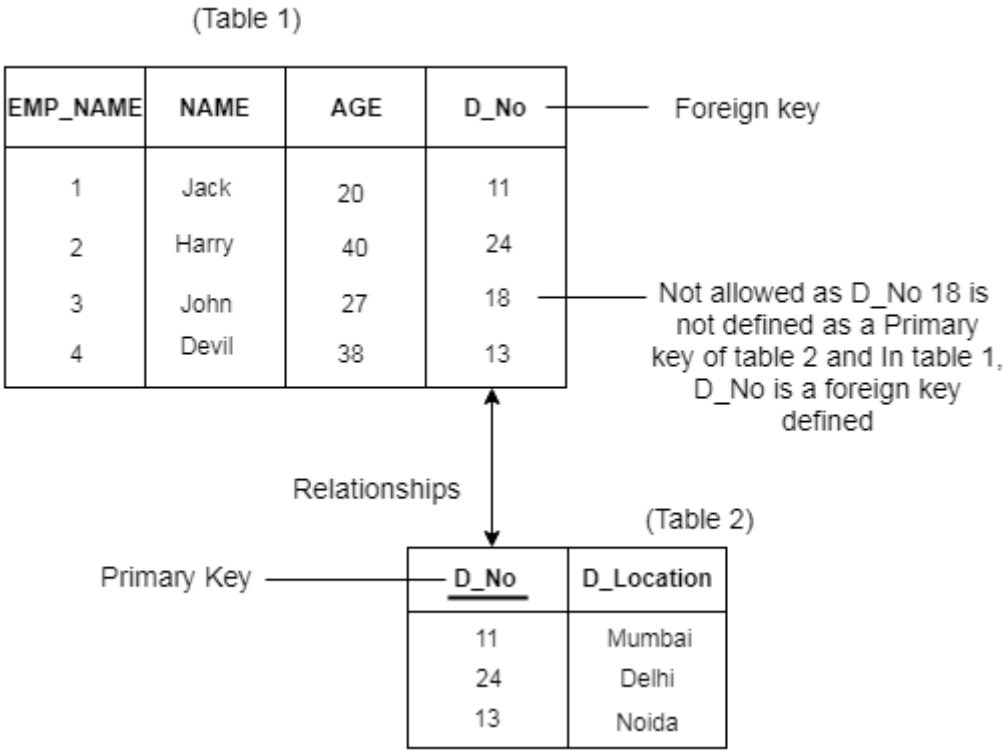
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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:



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 and null 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

8.4 Discuss Encryption

Database encryption is **the process of converting data, within a database, in plain text format into a meaningless cipher text by means of a suitable algorithm**. Database decryption is converting the meaningless cipher text into the original information using keys generated by the encryption algorithms.

Encryption is **an important way for individuals and companies to protect sensitive information from hacking**. For example, websites that transmit credit card and bank account numbers should always encrypt this information to prevent identity theft and fraud.