**File System vs DBMS – Difference between File System and DBMS**

|  |  |
| --- | --- |
| **File Management System** | **Database Management System** |
| File System is a general, easy-to-use system to store general files which require less security and constraints. | Database management system is used when security constraints are high. |
| Data Redundancy is more in file management system. | Data Redundancy is less in database management system. |
| Data Inconsistency is more in file system. | Data Inconsistency is less in database management system. |
| Centralisation is hard to get when it comes to File Management System. | Centralisation is achieved in Database Management System. |
| User locates the physical address of the files to access data in File Management System. | In Database Management System, user is unaware of physical address where data is stored. |
| Security is low in File Management System. | Security is high in Database Management System. |
| File Management System stores unstructured data as isolated data files/entities. | Database Management System stores structured data which have well defined constraints and interrelation. |
| A file manager is used to store all relationships in directories in file systems. | A database manager (administrator) stores the relationship in form of structural tables |
| Data is scattered in various files and files may be of different format, so data isolation problem exists | The problem of data isolation is not found in database |
| In file system, transactions are not possible | Transactions like insert, delete, view, updating, etc are possible in database |
| Concurrent access and recovery is not possible | Concurrent access and recovery is possible in database |

# Understanding DBMS Architecture

A Database Management system is not always directly available for users and applications to access and store data in it. A Database Management system can be **centralised**(all the data stored at one location), **decentralised**(multiple copies of database at different locations) or **hierarchical**, depending upon its architecture.

**1-tier DBMS** architecture also exist, this is when the database is directly available to the user for using it to store data. Generally such a setup is used for local application development, where programmers communicate directly with the database for quick response.

Database Architecture is logically of two types:

1. 2-tier DBMS architecture
2. 3-tier DBMS architecture

|  |  |
| --- | --- |
| 2-tier DBMS Architecture 2-tier DBMS architecture includes an **Application layer** between the user and the DBMS, which is responsible to communicate the user's request to the database management system and then send the response from the DBMS to the user.  An application interface known as **ODBC**(Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS. | 2-tier dbms architecture |

Such an architecture provides the DBMS extra security as it is not exposed to the End User directly. Also, security can be improved by adding security and authentication checks in the Application layer too.

|  |  |
| --- | --- |
| 3-tier DBMS Architecture 3-tier DBMS architecture is the most commonly used architecture for web applications.  It is an extension of the 2-tier architecture. In the 2-tier architecture, we have an application layer which can be accessed programatically to perform various operations on the DBMS. The application generally understands the Database Access Language and processes end users requests to the DBMS.  In 3-tier architecture, an additional Presentation or GUI Layer is added, which provides a graphical user interface for the End user to interact with the DBMS.  For the end user, the GUI layer is the Database System, and the end user has no idea about the application layer and the DBMS system. | 3-tier dbms architecture |

If you have used **MySQL**, then you must have seen **PHPMyAdmin**, it is the best example of a 3-tier DBMS architecture

**II way** There are following three levels or layers of [DBMS](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) architecture:

• External Level

•Conceptual Level

• Internal Level

## Objective of the Three Level Architecture

The objective of the three level architecture is to separate each user's view of the database from the Way the database is physically represented. There are several reasons why this separation is desirable:

• Each user should be able to access the same data, but have a different customized view of the data. Each user should be able to change the way he or she views the data, and this change should not affect other users.

• Users should not have to deal directly with physical database storage details, such as indexing or hashing. In other words a user's interaction with the database should be independent of storage considerations.

• The [Database Administrator](http://ecomputernotes.com/fundamental/what-is-a-database/what-is-dba) (DBA) should be able to change the database storage structures without affecting the user's views.

. The internal structure of the database should be unaffected by changes to the physical aspects of storage, such as the changeover to a new storage device.

. The DBA should be able to change the conceptual structure of the database without affecting all users.

## External Level or View level

It is the users' view of the database. This level describes that part of the database that is relevant to each user. External level is the one which is closest to the end users. This level deals with the way in which individual users vie\v data. Individual users are given different views according to the user's requirement.

A view involves only those portions of a database which are of concern to a user. Therefore same database can have different views for different users. The external view insulates users from the details of the internal and conceptual levels. External level is also known as the view level. In addition different views may have different representations of the same data. For example, one user may view dates in the form (day, month, year), while another may view dates as (year, month, day).

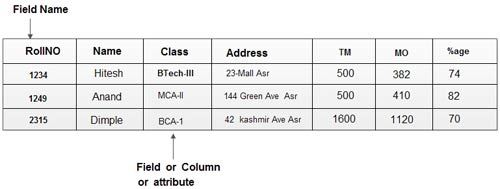
## Conceptual Level or Logical level

It is the community view of the database. This level describes what data is stored in the database and the relationships among the data. The middle level in the three level architecture is the conceptual level. This level contains the logical structure of the entire database as seen by the DBA. It is a complete view of the data requirements of the organization that is independent of any storage considerations. The conceptual level represents:

• All entities, their attributes, and their relationships;

An Entity is an object whose [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is stored in the database. For example, in student database the entity is student. An attribute is a characteristic of interest about an entity.

For example, in case of student database Roll No, Name, Class, Address etc. are attributes of entity student.

[](http://ecomputernotes.com/images/Field%20Name.jpg)

• The constraints on the data;

• Semantic information about the data;

• Security and integrity information.

The conceptual level supports each external view, in that any data available to a user must be contained in, or derivable from, the conceptual level. However, this level must not contain any storage dependent details. For instance, the description of an entity should contain only [data type](http://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java)s of attributes (for example, integer, real, character) and their length (such as the maximum number of digits or characters), but not any stQrage considerations, such as the number of bytes occupied. Conceptual level is also known as the, logical level.

## Internal level or Storage level

It is the physical representation of the database on the [computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer). This level describes how the data is stored in the database. The internal level is the one that concerns the way the data are physically stored on the hardware. The internal level covers the physical\ implementation of the database to achieve optimal runtime performance and storage space utilization. It covers the data structures and file organizations used to store data on [storage devices](http://ecomputernotes.com/fundamental/input-output-and-memory/explain-secondary-storage-devices). It interfaces with the [operating system](http://ecomputernotes.com/fundamental/disk-operating-system/what-is-operating-system) access methods to place the data on the storage devices, build the indexes, retrieve the data, and so· on.

The internal level is concerned with such things as:

• Storage space allocation for data and indexes;

• Record descriptions for storage (with stored sizes for data items);

• Record placement;

• Data compression and data encryption techniques.

There will be only one conceptual view, consisting of the abstract representation of the database in it’s entirely. Similarly there will be only one internal or physical view, representing the total database, as it is physically stored.

## Schema

It is important to note that the data in the database changes frequently, while the plans or schemes remain the same over long periods of time. The database plans consist of types of entities that a database deals with, the relationship among these entities and the ways in which the entities and relationships are expressed from one level of abstraction to the next level for the users' view. The users' view of the data (also called logical organization of data) should be in a form that is most convenient for the users and they should not be concerned about the way data is physically organized. Therefore, a DBMS should do the translation between the logical (users' view) organization and the physical organization of the data in the database.

The plan or scheme of the database is known as Schema. Schema gives the names of the entities and attributes. It specifies the relationship among them. It is a framework into which the values of the data items (or fields) are fitted. The plans or the format of schema remains the same. But the values fitted into this format changes from instance to instance. In other terms, schema means overall plans of all the data item (field) types and record types stored in a database. Schema includes the definition of the database name, the record type and the components that make up those records

### Types of Schema

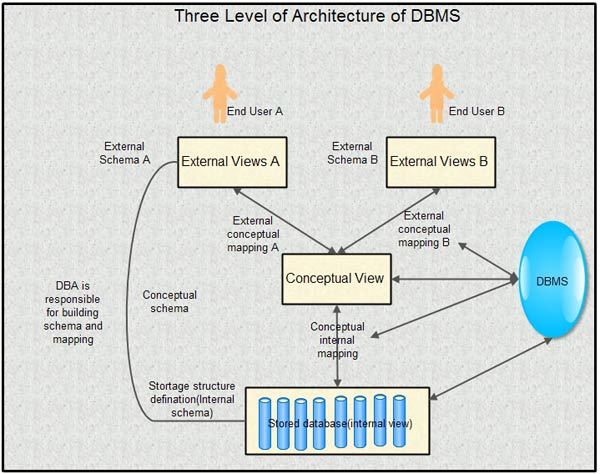
There are three different types of schema in the database corresponding to each data view of database. In other words, the data views at each of three levels are described by schema.

A schema is defined as an outline or a plan that describes the records and relationships existing at the particular level. **The External view is described by means of a schema called external schema that correspond to different views of the data. Similarly the Conceptual view is defined by conceptual schema, which describes all the entities, attributes, and relationship together with integrity constraints. Internal View is defined by internal schema, which is a complete description of the internal model, containing definition of stored records, the methods of representation, the data fields, and the indexes used.**

There is only one conceptual schema and one internal schema per database. The schema also describes the way in which data elements at one level can be mapped to the corresponding data elements in the next level.

Thus, we can say that schema establishes correspondence between the records and relationships in the two levels. In a relational database, the schema defines the tables, the fields in each table, and the relationships between fields and tables. Schema are generally stored in a data dictionary.

The data in the database at any particular point in time is called a database instance. Therefore, many database instances can correspond to the same database schema. The schema is sometimes called the intension of the database, while an instance is called an extension (or state) of the database.

[](http://ecomputernotes.com/images/Three-Level-of-Architecture-of-DBMS.jpg)

Example: To understand the difference between the three levels, consider again the database schema that describes College Database system. If User1 is a Library clerk, the external view would contain only the student and book information. If User2 is an account office clerk then he/she may be interested in students detail and fee detail. Shows specific information actually available at each level regarding a particular user.

The external view would depend upon the user who is accessing the database. The conceptual level contain the logical view of the whole database, it represents the data type of each required field. The internal view represents the physical location of each element on the disk of the servers well as how many bytes of storage each element needs.

## Mapping between Views

The DBMS is responsible for mapping between these three types of schema. Two mappings are required in a database system with three different views.

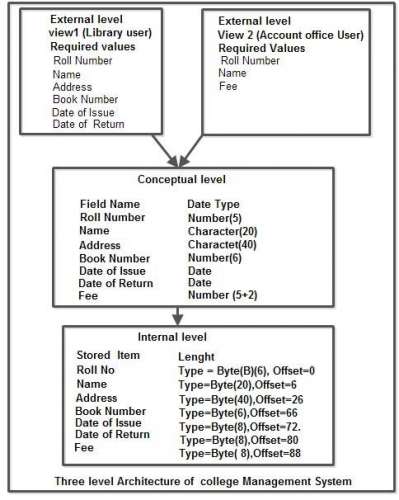
**External/Conceptual Mapping**: Each external schema is related to the conceptual schema by the external/conceptual mapping. A mapping between the external and conceptual views gives the correspondence among the records and the relationships of the external and conceptual views the external view is an abstraction of the conceptual view, which in its turn is an abstraction of the internal view. It describes the contents of the database as perceived by the user or application program of that view. The user of the external view sees and manipulates a record corresponding to the external view. There is a mapping from0 a particular logical record in the external view to one (or more) conceptual record(s) in the conceptual view.

**Differences between External/Conceptual views**

Following could be differences that exist between the two:

Names of the field’s and. records, for instance, may be different. A number of conceptual fields can be combined into a single external field, for example, Last\_Name and First\_Name at the conceptual level but Name at the external level. A given external record could be derived from a number of conceptual .records.

**Conceptual/Internal Mapping:** Conceptual schema is related to the internal schema by the conceptual/internal mapping. This enables the DBMS to find the actual record or combination of records in physical storage that constitute a logical record in conceptual schema. Mapping between the conceptual and the internal levels specifies the method of deriving the conceptual record from the physical database.

[](http://ecomputernotes.com/images/Three-level-Architecture.jpg)

**Database Schema**

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated.

A database schema does not contain any data or information. It formulates all the constraints that are to be applied on the data.

A database schema can be divided broadly into two categories :

1. **Physical Database Schema :** This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
2. **Logical Database Schema :** This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**Database Instance: Instance** : Collection of information stored in the database at a particular (instance of time) moment is called an instance of the database.

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

In a given database state, each schema construct has its own current set of instances.

The DBMS stores the descriptions of the schema constructs and constraints also called metadata in the DMBS catalog so that DBMS software can refer to the schema whenever it needs to.

**Data Abstraction**

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Developers hide the complexity from users through several levels of abstraction to simplify users interactions with the system.

**Schema**: The overall design of the database is called the database schemas.

Analogy to the concept  of data types, variables, and values in programming language :

consider a structure in c.

*structure customer  
{  
     char customer\_name[50];  
     char social\_security[50];  
     char customer\_street[50];  
     char customer\_city[50];  
}cust;*

Here, cust is a variable of type customer structure. A database schema corresponds to the programming language type definition.

A variable of a given type has a value at a given instance. The value of the variable in programming languages corresponds to an instance of a database schema.

Database supports three database schemas:

1. **Physical Schema** : It is at the lowest level. i.e. at Physical level.

2. **Logical Schema** : It is the next or intermediate level i.e. at Logical Level.

3.**Sub-Schema** : It is at the highest level i.e. at the view level.

**Data independence**

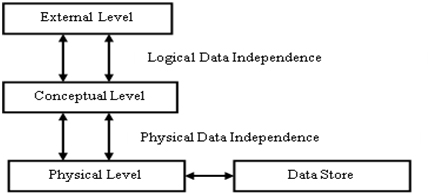
Three levels of the architecture

**External – individual user view**

**Conceptual – community user view**

**Internal – physical or storage view**

The three level database architecture allows a clear separation of the information meaning (conceptual view) from the external data representation and from the physical data structure layout. A database system that is able to separate the three different views of data is likely to be flexible and adaptable. This flexibility and adaptability is data independence



**External View / Level / Schema / User View / Global View :**The highest level of abstraction describes only part of the entire database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many view for the same database.

* Highest or Top level of data abstraction ( No knowledge of DBMS S/W and H/W or physical storage). **This level is concerned with the user.**
* Each external schema describes the part of the database that a particular user is interested in and hides the rest of the database from user.
* There can be n number of external views for database where n is the number of users.
* For example, a accounts department may only be interested in the student fee details. It would not be expected to have any interest in the personal information about students

**Conceptual View** The next-higher level of abstraction describes **what** data are stored in the database, and what relationships exist among those data.

* This level is in between the user level and physical storage view.
* There is only one conceptual view for single database.
* It hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

**Internal View / Physical View** The lowest level of abstraction describes **how** the data are actually stored.

* It is the lowest level of data abstraction. (it has the knowledge about s/w and h/w)
* At this level, it keeps the information about the actua representation of the entire database i.e. the actual storage of the data on the disk in the form of records or blocks.
* It is close to the physical storage method.
* The internal view is the view that tells us what data is stored in the database and how. At least the following aspects are considered at this level: Storage allocation, Access paths etc.
* The internal view does not deal with the physical devices directly. Instead it views a physical device as a collection of physical pages and allocates space in terms of logical pages.

## Data Independence

Data Independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

Data Independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed.

**There are two types of data independence :**

1. **Logical Data Independence**
2. **Physical Data Independence**

**Logical Data Independence**

It is the capacity to change the conceptual schema without having to change external schemas or application program.

We may change the conceptual schema to expand the database, to change constraints, or to reduce the database.

### Physical Data Independence

It is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schema need not to be changed as well.

Changes to the internal schema may be needed because some physical files had to be reorganized.

## Advantages of DBMS

#### Controlling of Redundancy:

Data redundancy refers to the duplication of data (i.e storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

**2. Improved Data Sharing:**

DBMS allows a user to share the data in any number of application programs.

#### Data Integrity:

Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database we can can enforce an integrity that it must accept the customer only from Noida and Meerut city.

#### Security:

Having complete authority over the operational data, enables the DBA in ensuring that the only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

#### Data Consistency :

By eliminating data redundancy, we greatly reduce the opportunities for inconsistency. For example: is a customer address is stored only once, we cannot have disagreement on the stored values. Also updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage that results from redundant data storage.

#### Efficient Data Access :

In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing

#### Enforcements of Standards :

With the centralized of data, DBA can establish and enforce the data standards which may include the naming conventions, data quality standards etc.

#### Data Independence :

Ina database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the meta data obtained by the DBMS is changed but the DBMS is continues to provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

#### Reduced Application Development and Maintenance Time :

DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application.

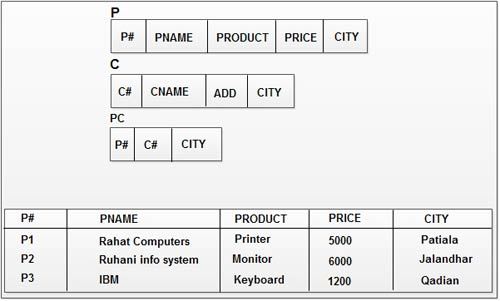
**Disadvantages of DBMS**

1. Increased Complexity
2. Requirement of New and Specialized Manpowers
3. Large Size of DBMS

# [What are INTANCES, SCHEMAS AND SUBSCHEMA in DBMS?](http://ecomputernotes.com/fundamental/what-is-a-database/intances-schemas-and-subschema)

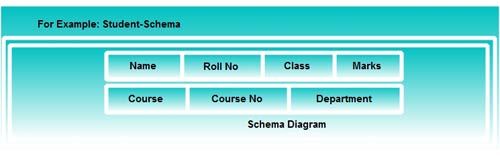
Database changes over time when [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is inserted or deleted. The collection of information stored in the [database](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) at a particular moment is called an instance of the database. The overall design of the database is called the database schema.

A schema diagram, as shown above, displays only names of record types (entities) and names of data items (attributes) and does not show the relationships among the various files.

[](http://ecomputernotes.com/images/Instances.jpg)

The schema will remain the same while the values filled into it change from instant to instant. When the schema framework is filled in with data item values, it is referred as an instance of the schema. The data in the database at a particular moment of time is called a database state or snapshot, which is also called the current set of occurrences or instances in the database

In other words, "the description of a database is called the database schema, which is specified during database design and is not expected to change frequently". A displayed schema is called a schema diagram.

[](http://ecomputernotes.com/images/Schema%20Diagram.jpg)

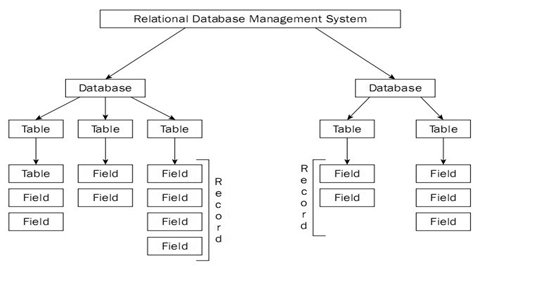
A schema diagram displays only some aspects of a schema, such as the namer. of record types and data items, and some types of constraints. Other aspects are not specified in the schema diagram. It does not specify the [data type](http://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java) of each data item and the relationships among the various files.

## ****Subschema****

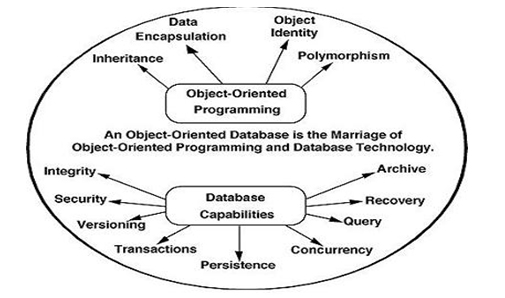
A subschema is a subset of the schema and inherits the same property that a schema has. The plan (or scheme) for a view is often called subschema. Subschema refers to an application programmer's (user's) view of the data item types and record types, which he or she uses. It gives the users a window through which he or she can view only that part of the database, which is of interest to him. Therefore, different application programs can have different view of data.

**Types And Classification Of Database Management System–** As we all know DBMS is an interesting subject and so is its classification. There are several criteria based on which DBMS is classified. The classification and types of **Database Management System**(DBMS) is explained in a detailed manner below based on the different factors. At the end of this article, you will be given a free pdf copy of all these types of DBMS.  
**Based on the data model**

**Relational database** – This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).

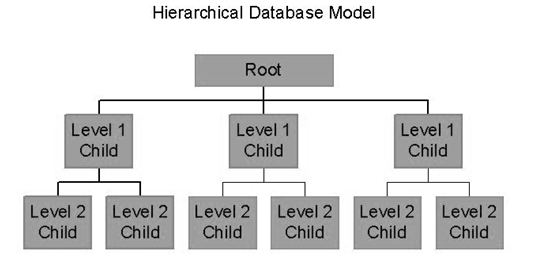
[](http://whatisdbms.com/wp-content/uploads/2017/03/relational-DBMS.jpg)

**Object oriented database** – The information here is in the form of the object as used in object oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain. Examples are ObjectDB (ObjectDB software).

[](http://whatisdbms.com/wp-content/uploads/2017/03/Object-oriented-database.jpg)

**Object relational database** – Relational DBMS are evolving continuously and they have been incorporating many concepts developed in object database leading to a new class called extended relational database or object relational database.

**Hierarchical database** – In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry(Microsoft).

[](http://whatisdbms.com/wp-content/uploads/2017/03/Hierarchical-database.jpg)

**Network database** – Mainly used on large digital computers. If there are more connections, then this database is efficient. They are similar to hierarchical database, they look like a cobweb or interconnected network of records. Examples are CA-IDMS (COMPUTER associates), IMAGE(HP).

### ****Based on the number of users****

**Single user** – As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

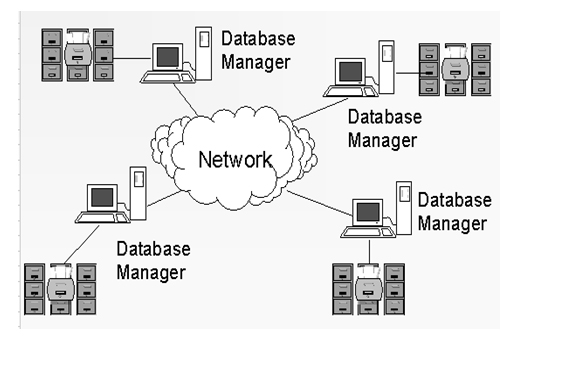
**Multiple users** – It supports multiple users concurrently. Data can be both integrated and shared,a database should be integrated when the same information is not need be recorded in two places. For example a student in the college should have the database containing his information. It must be accessible to all the departments related to him. For example the library department and the fee section department should have information about student’s database. So in such case, we can integrate and even though database resides in only one place both the departments will have the access to it.

### ****Based on the sites over which network is distributed****

|  |  |
| --- | --- |
| **Centralized database system** – The DBMS and database are stored at the single site that is used by several other systems too. We can simply say that data here is maintained on the centralized server. | <http://whatisdbms.com/wp-content/uploads/2017/03/Centralized-database-system.jpg> |

**Parallel network database system** – This system has the advantage of improving processing input and output speeds. Majorly used in the applications that have query to larger database. It holds the multiple central processing units and data storage disks in parallel.

**Distributed database system** – In this data and the DBMS software are distributed over several sites but connected to the single computer.

[](http://whatisdbms.com/wp-content/uploads/2017/03/Distributed-database-system.jpg)

 Further they are classified as

     1**.Homogeneous DBMS** – They use same software but from the multiple sites. Data exchange between the sites can be handled easily. For example, library information systems by the same vendor ,such as Geac Computer corporation, use the same DBMS software that allows the exchanges between various Geac library sites.

     2.**heterogeneous DBMS**– They use different DBMS software for different sites but there is a additional software that helps the exchange of the data between the sites.

**Client-server database system** –  This system has two logical components  namely client and server. Clients are generally the personal computers or workstations whereas servers are the large workstations, mini range computers or a main frame computer system. The applications and tools of the DBMS run on the client platforms and the DBMS software on the server. Both server and client computers are connected over the network. We can relate it to client and server in real life to understand in a much better way. Here the applications and tools act as a client send the requests for its services. The DBMS processes these requests and returns the result to the client. Server handles jobs that are common to many clients say database access and updates.

**Multi-tier client-server database system** – The rise of personal computers in business has increased the reliability of the network hardware leading to evolution of two-tier and three-tier systems which use different software for the client and software.

### ****Based on the cost****

**Low cost DBMS** – The cost of these systems vary from $100 to $3000.

**Medium cost  DBMS** – Cost varies from $10000 to $100000.

**High cost DBMS** –  Cost pf these systems are usually more than $100000.

### ****Based on the access****

This classification simply based on the access to data in the database systems

**Sequential access** – One after the other.

**Direct access**

**Inverted file structures**

### ****Based on the usage****

**Online transaction processing(OLTP) DBMS** – They manage the operational data. Database server must be able to process lots of simple transactions per unit of time. Transactions are initiated in real time, in simultaneous by lots of user and applications hence it must have high volume of short, simple queries.

**Online analytical processing(OLAP) DBMS** – They use the operational data for tactical and strategical decision making. They have limited users deal with huge amount of data,complex queries.

**Big data and analytics DBMS** – To cope with big data new database technologies have been introduced. One such is NoSQL (not only SQL) which abandons the well known relational database scheme.

**XML DBMS** – two types

1. Native XML DBMS – Use the logical,intrinsic structure of XML document.

     2 .Enabled XML DBMS – Existing DBMS with facilities to store XML data and structured data in integrated way.

**Multimedia DBMS** – Stores data such as text, images, audio, video and 3D games which are usually stored in binary large object.

**GIS DBMS** – Stores and queries the spatial data.

**Sensor DBMS** – Allows to manage sensor data, bio-metric and telematics data.

**Mobile DBMS** – Runs on the smartphones, tablets. It Handles the local queries. Supports self management( no DBA).

**Open source DBMS** – Code is publicly available and can be extended by anyone, popular for small business applications.

# DBMS Database Models

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

* Hierarchical Model
* Network Model
* Entity-relationship Model
* Relational Model

|  |  |
| --- | --- |
| Hierarchical Model This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node | Hierarchical Model of database |

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

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

|  |  |
| --- | --- |
| Network Model This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node. In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships | Network Model of database |

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

## Entity-relationship Model

|  |  |
| --- | --- |
| .In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.  Different entities are related using relationships.  E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand. | E-R Model of database |

This model is good to design a database, which can then be turned into tables in relational model(explained below).

Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

Relationships can also be of different types. To learn about [E-R Diagrams](https://www.studytonight.com/dbms/er-diagram.php) in details, click on the link.

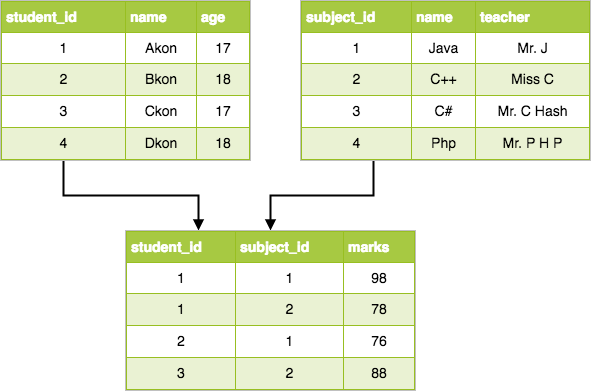
## Relational Model

In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world. The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as **relations** in relational model.

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



#### Reducing Entity-Relationship Schema to Tables

**To represent the data and relationships in an E-R database in a collection of tables.**

##### **Reducing Entity Sets to Tables**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The table corresponding to an entity set has a column for each attribute of the entity set and a row for each entity in the entity set. The table has the same name as the entity set.  Example: the table for [entity set fly](https://turing.cs.hbg.psu.edu/courses/comp419.taw.s97/er.html#es%20fly).  Discussion: does the order of rows matter? | |  |  |  |  | | --- | --- | --- | --- | | fly | | | | | **pattern** | **size** | **color** | **fly-stock-num** | | woolly bugger | 10 | olive | 5 | | gold-ribbed hare's ear | 14 | natural | 2 | | woolly bugger | 2 | black | 4 | | Dahlberg Diver | 2 | natural | 1 | |

##### **Reducing Relationship Sets to Tables**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The table for a relationship set will have a column for each attribute of the [superkey](https://turing.cs.hbg.psu.edu/courses/comp419.taw.s97/er.html" \l "primary%20keys%20of%20relationship%20sets)of that relationship set (i.e. one for each attribute of the primary key of each entity set involved in the relationship, and one for each attribute of the relationship set). These attributes are renamed if necessary to avoid name clashes. The table will have one row for each relationship, and will have the same name as the relationship set.  Example: [relationship set Requested-By](https://turing.cs.hbg.psu.edu/courses/comp419.taw.s97/er.html#rs%20requested-by) becomes: | |  |  | | --- | --- | | Requested-By | | | **cust-num** | **fly-stock-num** | | 1 | 2 | | 3 | 5 | | 2 | 4 | | 2 | 1 | |

Example: [relationship set Purchased](https://turing.cs.hbg.psu.edu/courses/comp419.taw.s97/er.html#rs%20requested-by) (including the date attribute) becomes:

|  |  |  |
| --- | --- | --- |
| Purchased | | |
| **cust-num** | **flyrod-stock-num** | **date** |
| 1 | 3 | 9/4/1994 |
| 3 | 3 | 9/3/1994 |
| 2 | 2 | 8/8/1992 |
| 3 | 2 | 9/2/1994 |
| 3 | 1 | 9/1/1994 |
|  |  |  |

# ER Model to Relational Model

As we all know that ER Model can be represented using ER Diagrams which is a great way of designing and representing the database design in more of a flow chart form.

It is very convenient to design the database using the ER Model by creating an ER diagram and later on converting it into relational model to design your tables.

Not all the ER Model constraints and components can be directly transformed into relational model, but an approximate schema can be derived.

So let's take a few examples of ER diagrams and convert it into relational model schema, hence creating tables in RDBMS.

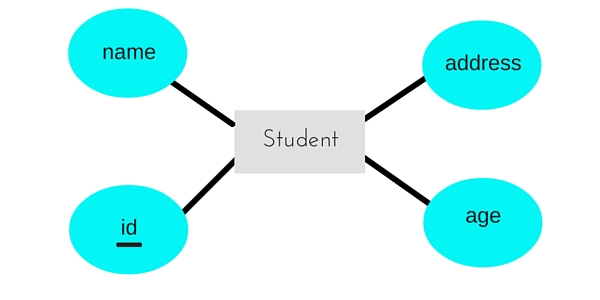
## Entity becomes Table

Entity in ER Model is changed into tables, or we can say for every Entity in ER model, a table is created in Relational Model.

And the **attributes** of the Entity gets converted to columns of the table.

And the primary key specified for the entity in the ER model, will become the primary key for the table in relational model.

For example, for the below ER Diagram in ER Model,

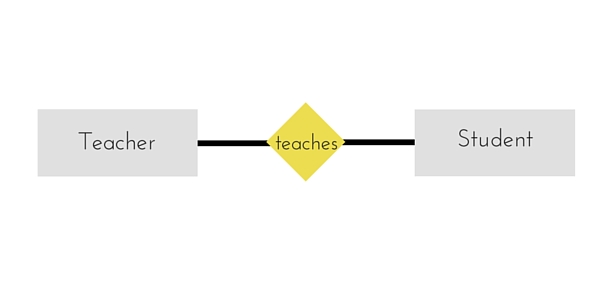


A table with name **Student** will be created in relational model, which will have 4 columns, id, name, age, address and id will be the primary key for this table.

## Relationship becomes a Relationship Table

In ER diagram, we use diamond/rhombus to reprsent a relationship between two entities. In Relational model we create a relationship table for ER Model relationships too.

In the ER diagram below, we have two entities **Teacher** and **Student** with a relationship between them.



As discussd above, entity gets mapped to table, hence we will create table for **Teacher** and a table for **Student** with all the attributes converted into columns.

Now, an additional table will be created for the relationship, for example **StudentTeacher** or give it any name you like. This table will hold the primary key for both Student and Teacher, in a tuple to describe the relationship, which teacher teaches which student.

If there are additional attributes related to this relationship, then they become the columns for this table, like subject name.

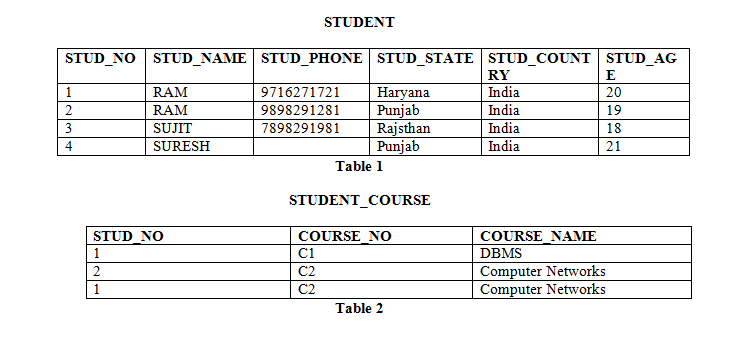
Also proper foreign key constraints must be set for all the tables.

### Points to Remember

Similarly we can generate relational database schema using the ER diagram. Following are some key points to keep in mind while doing so:

1. Entity gets converted into Table, with all the attributes becoming fields(columns) in the table.
2. Relationship between entities is also converted into table with primary keys of the related entities also stored in it as foreign keys.
3. Primary Keys should be properly set.
4. For any relationship of Weak Entity, if primary key of any other entity is included in a table, foreign key constraint must be defined.

**Different Types of Keys in Relational Model**

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/image7.png)

**Candidate Key:** The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD\_NO in STUDENT relation.

The value of Candidate Key is unique and non-null for every tuple.

* There can be more than one candidate key in a relation. For Example, STUD\_NO as well as STUD\_PHONE both are candidate keys for relation STUDENT.
* The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD\_NO, COURSE\_NO} is a composite candidate key for relation STUDENT\_COURSE.

**Note –** In Sql Server a unique constraint that has a nullable column, **allows** the value ‘**null**‘ in that column **only once**. That’s why STUD\_PHONE attribute as candidate here, but can not be ‘null’ values in primary key attribute.

**Super Key:**The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD\_NO, (STUD\_NO, STUD\_NAME) etc.

* Adding zero or more attributes to candidate key generates super key.
* A candidate key is a super key but vice versa is not true.

**Primary Key:** There can be more than one candidate key in a relation out of which one can be chosen as primary key. For Example, STUD\_NO as well as STUD\_PHONE both are candidate keys for relation STUDENT but STUD\_NO can be chosen as primary key (only one out of many candidate keys).

**Alternate Key:**The candidate key other than primary key is called as alternate key. For Example, STUD\_NO as well as STUD\_PHONE both are candidate keys for relation STUDENT but STUD\_PHONE will be alternate key (only one out of many candidate keys).

**Foreign Key:** If an attribute can only take the values which are present as values of some other attribute, it will be foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and corresponding attribute is called referenced attribute and the relation which refers to referenced relation is called referencing relation and corresponding attribute is called referencing attribute. Referenced attribute of referencing attribute should be primary key. For Example, STUD\_NO in STUDENT\_COURSE is a foreign key to STUD\_NO in STUDENT relation.

## Why we need a Key?

In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well.

Also, tables store a lot of data in them. Tables generally extends to thousands of records stored in them, unsorted and unorganised.

Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?

To avoid all this, **Keys** are defined to easily identify any row of data in a table.

Let's try to understand about all the keys using a simple example.

|  |  |  |  |
| --- | --- | --- | --- |
| **student\_id** | **name** | **phone** | **age** |
| 1 | Akon | 9876723452 | 17 |
| 2 | Akon | 9991165674 | 19 |
| 3 | Bkon | 7898756543 | 18 |
| 4 | Ckon | 8987867898 | 19 |
| 5 | Dkon | 9990080080 | 17 |

Let's take a simple **Student** table, with fields student\_id, name, phone and age.

### Super Key

**Super Key** is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

In the table defined above super key would include student\_id, (student\_id, name), phoneetc.

Confused? The first one is pretty simple as student\_id is unique for every row of data, hence it can be used to identity each row uniquely.

Next comes, (student\_id, name), now name of two students can be same, but their student\_idcan't be same hence this combination can also be a key.

Similarly, phone number for every student will be unique, hence again, phone can also be a key.

So they all are super keys.

### Candidate Key

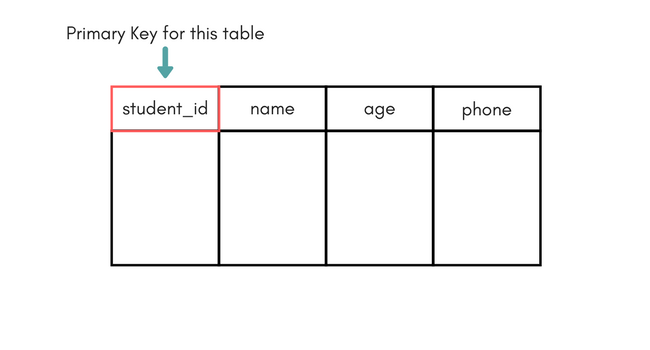
Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

In our example, student\_id and phone both are candidate keys for table **Student**.

* A candiate key can never be NULL or empty. And its value should be unique.
* There can be more than one candidate keys for a table.
* A candidate key can be a combination of more than one columns(attributes).

### Primary Key

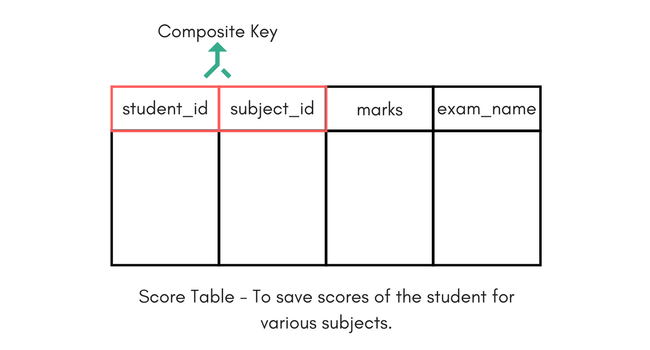
Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.



For the table **Student** we can make the student\_id column as the primary key.

### Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independentely or individually.



In the above picture we have a **Score** table which stores the marks scored by a student in a particular subject.

In this table student\_id and subject\_id together will form the primary key, hence it is a composite key.

### Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

### Non-key Attributes

**Non-key** attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

### Non-prime Attributes

**Non-prime** Attributes are attributes other than **Primary Key attribute(s).**.

# Concepts of ER Model in DBMS

Entity-relationship model is a model used for design and representation of relationships between data.

The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.

In short, to understand about the ER Model, we must understand about:

* Entity and Entity Set
* What are Attributes? And Types of Attributes.
* Keys
* Relationships

Let's take an example to explain everything. For a **School Management Software**, we will have to store **Student** information, **Teacher** information, **Classes**, **Subjects** taught in each class etc.

## ER Model: Entity and Entity Set

Considering the above example, **Student** is an entity, **Teacher** is an entity, similarly, **Class**, **Subject**etc are also entities.

An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.

If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set**

## ER Model: Attributes

If a Student is an Entity, then student's **roll no.**, student's **name**, student's **age**, student's **gender** etc will be its attributes.

An attribute can be of many types, here are different types of attributes defined in ER database model:

1. **Simple attribute:** The attributes with values that are atomic and cannot be broken down further are simple attributes. For example, student's **age**.
2. **Composite attribute:** A composite attribute is made up of more than one simple attribute. For example, student's **address** will contain, **house no.**, **street name**, **pincode** etc.
3. **Derived attribute:** These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example, *average age of students in a class*.
4. **Single-valued attribute:** As the name suggests, they have a single value.
5. **Multi-valued attribute:** And, they can have multiple values.

## ER Model: Keys

If the attribute **roll no.** can uniquely identify a student entity, amongst all the students, then the attribute **roll no.** will be said to be a key.

Following are the types of Keys:

1. Super Key
2. Candidate Key
3. Primary Key

We have covered Keys in details here in [Database Keys](https://www.studytonight.com/dbms/database-key.php) tutorial.

## ER Model: Relationships

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

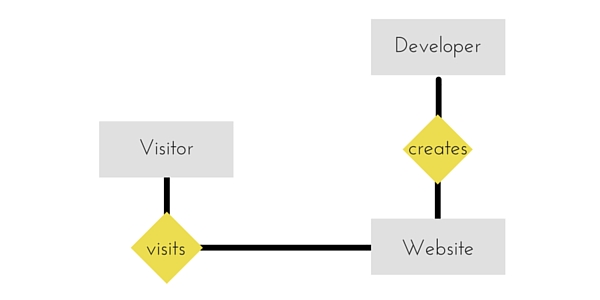
Depending upon the number of entities involved, a **degree** is assigned to relationships.

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

# Working with ER Diagrams

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

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



## Components of ER Diagram

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

Let's see how we can represent these in our ER Diagram.

|  |  |
| --- | --- |
| Entity Simple rectangular box represents an Entity. | Entity in ER diagram |

#### Relationships between Entities - Weak and Strong

|  |  |
| --- | --- |
| Rhombus is used to setup relationships between two or more entities. | Relationships in ER diagram |

#### Attributes for any Entity

|  |  |
| --- | --- |
| Ellipse is used to represent attributes of any entity. It is connected to the entity. | Attribute in ER diagram |

#### Weak Entity

|  |  |
| --- | --- |
| A weak Entity is represented using double rectangular boxes. It is generally connected to another entity. | Weak Entity in ER diagram |

#### Key Attribute for any Entity

|  |  |
| --- | --- |
| To represent a Key attribute, the attribute name inside the Ellipse is underlined. | Key Attribute in ER diagram |

#### Derived Attribute for any Entity

|  |  |
| --- | --- |
| Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.  To represent a derived attribute, another dotted ellipse is created inside the main ellipse. | Derived Attribute in ER diagram |

#### Multivalued Attribute for any Entity

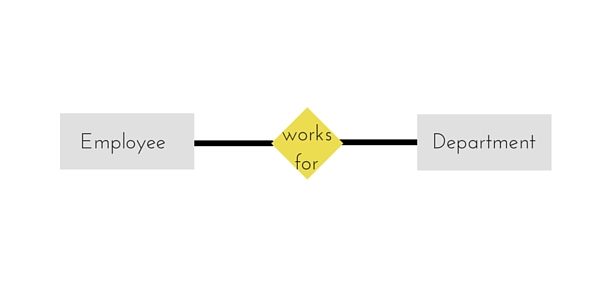
|  |  |
| --- | --- |
| Double Ellipse, one inside another, represents the attribute which can have multiple values. | Multivalued Attribute in ER diagram |

#### Composite Attribute for any Entity

|  |  |
| --- | --- |
| A composite attribute is the attribute, which also has attributes. | Composite Attribute in ER diagram |

### ER Diagram: Entity

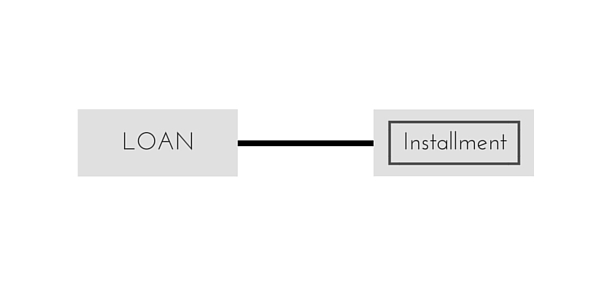
An **Entity** can be any object, place, person or class. In ER Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.



The yellow rhombus in between represents a relationship.

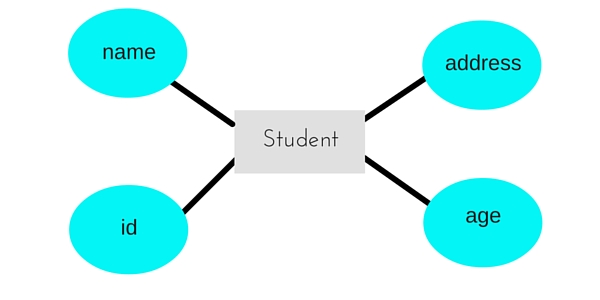
### ER Diagram: Weak Entity

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



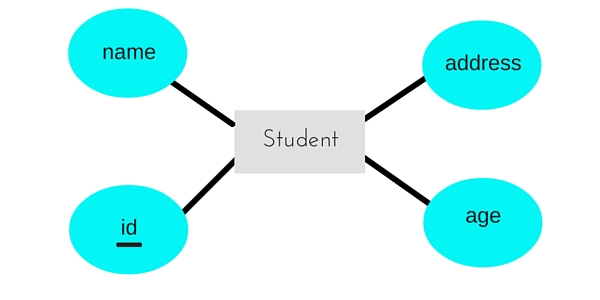
### ER Diagram: Attribute

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



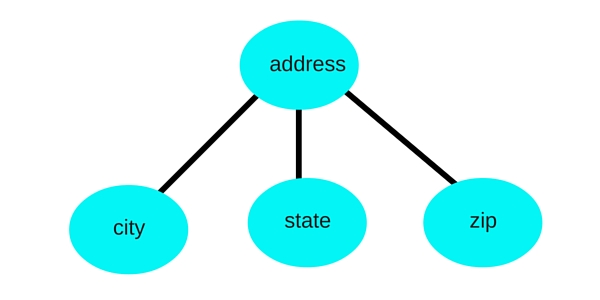
### ER Diagram: Key Attribute

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



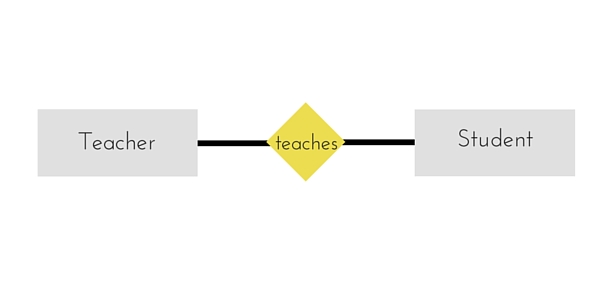
### ER Diagram: Composite Attribute

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



### ER Diagram: Relationship

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



There are three types of relationship that exist between Entities.

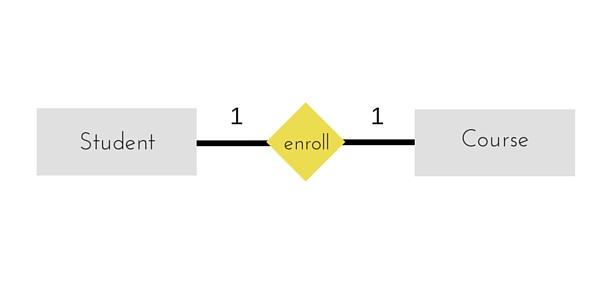
1. Binary Relationship
2. Recursive Relationship
3. Ternary Relationship

### ER Diagram: Binary Relationship

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

#### One to One Relationship

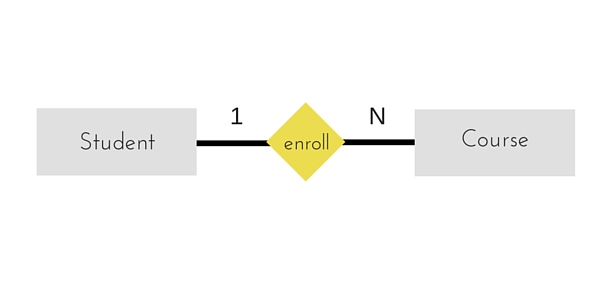
This type of relationship is rarely seen in real world.



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

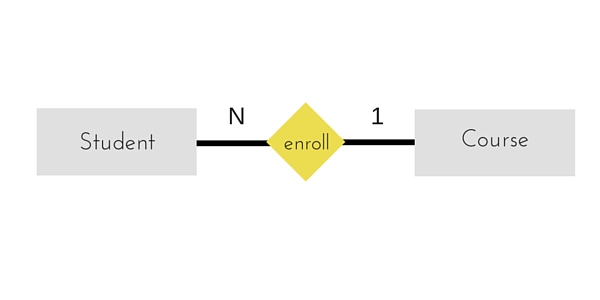
#### One to Many Relationship

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

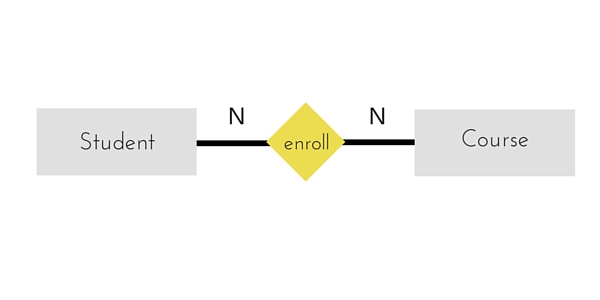


#### Many to One Relationship

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



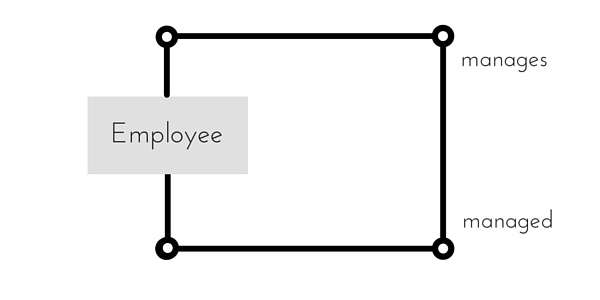
Many to Many Relationship



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

### ER Diagram: Recursive Relationship

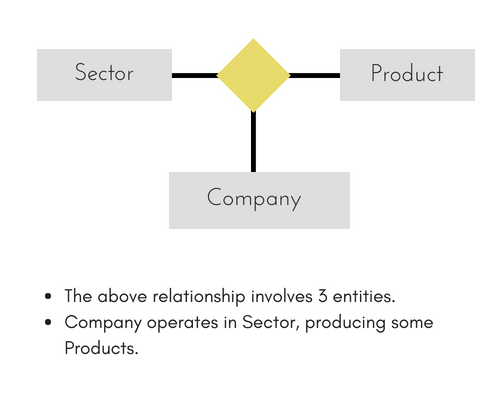
When an Entity is related with itself it is known as **Recursive** Relationship.



### ER Diagram: Ternary Relationship

Relationship of degree three is called Ternary relationship.

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



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

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

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

Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

# The Enhanced ER Model

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

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

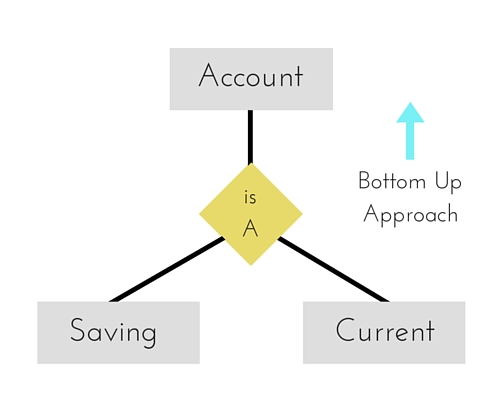
1. Generalization
2. Specialization
3. Aggregration

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

## Generalization

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

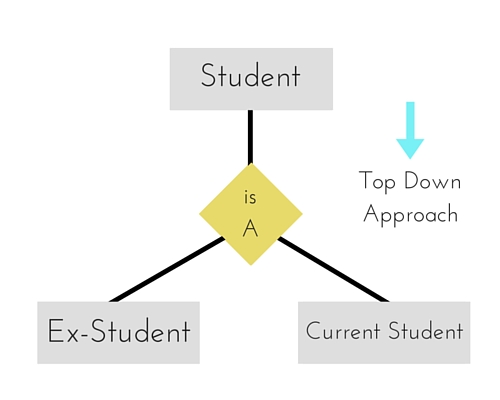
It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalized entity, in other words, sub-classes are combined to form a super-class.



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

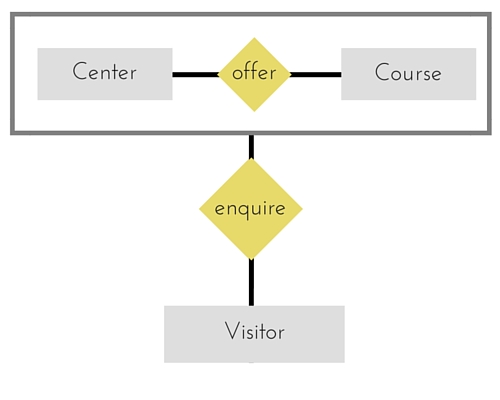
## Specialization

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



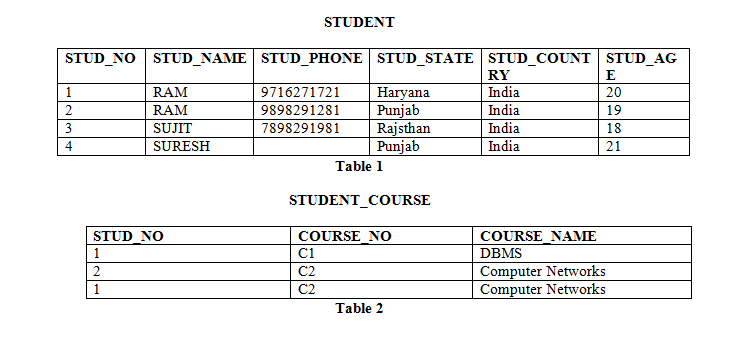
## Aggregration

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



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

**Anomalies in Relational Databases**  
There are different types of anomalies which can occur in referencing and referenced relation which can be discussed as:

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/image7.png)

**Insertion anomaly:** If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation. For Example, If we try to insert a record in STUDENT\_COURSE with STUD\_NO =7, it will not allow.

**Deletion and Updation anomaly:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will not allow deleting the tuple from referenced relation. For Example, If we try to delete a record from STUDENT with STUD\_NO =1, it will not allow. To avoid this, following can be used in query:

* **ON DELETE/UPDATE SET NULL:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and set the value of referenced attribute to NULL.
* **ON DELETE/UPDATE CASCADE:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and referencing relation as well.

II ER Model

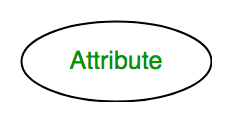
# Database Management System | ER Model

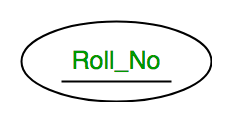
ER Model is used to model the logical view of the system from data perspective which consists of these components:

**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 | https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Database-Management-System-ER-Model.png |

**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.



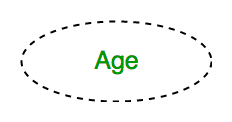
1. **Key Attribute –**  
   The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll\_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.
2. **Composite Attribute –**  
   An attribute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

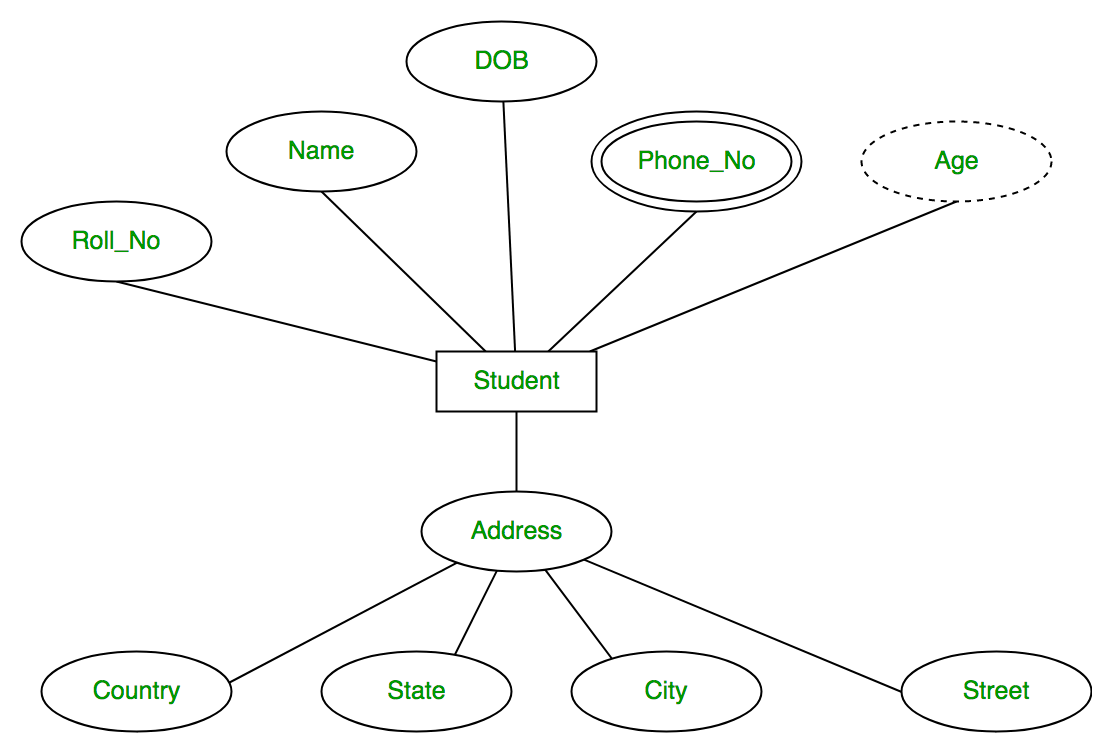


1. **Multivalued Attribute –**  
   An attribute consisting **more than one value** for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.



1. **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.

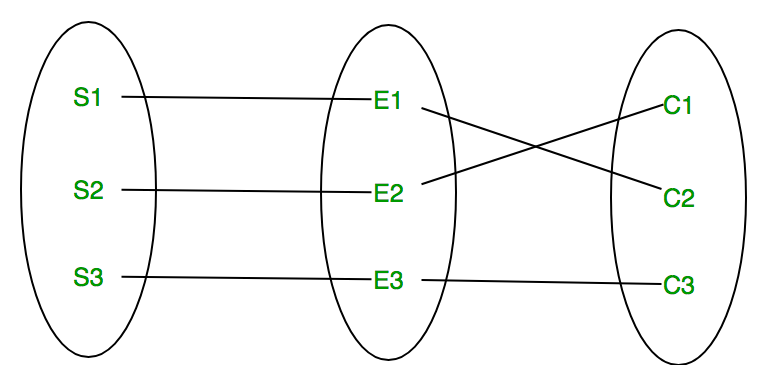
The complete entity type**Student** with its attributes can be represented as:

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Database-Management-System-ER-Model-7.png)

**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.

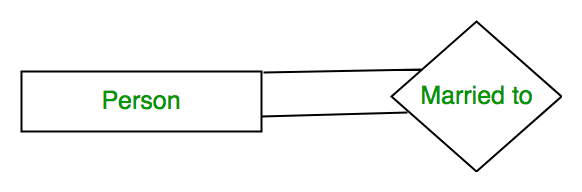


A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



**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.



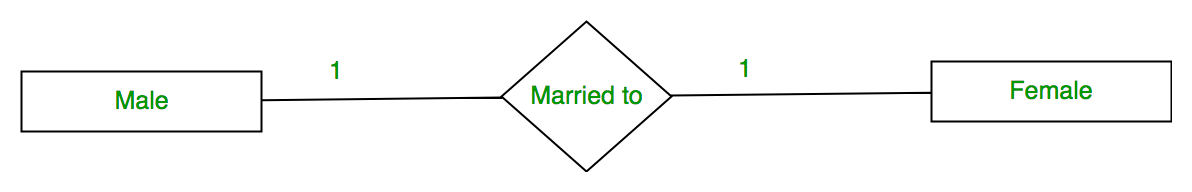
1. **Binary Relationship –**  
   When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.



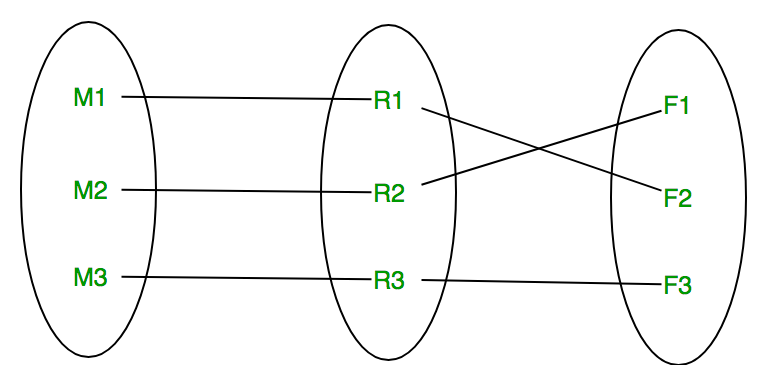
1. **n-ary Relationship –**  
   When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

**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:

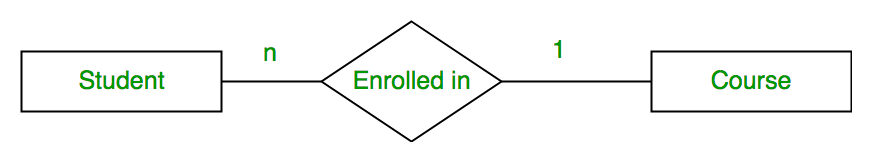
1. **One to one –** When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

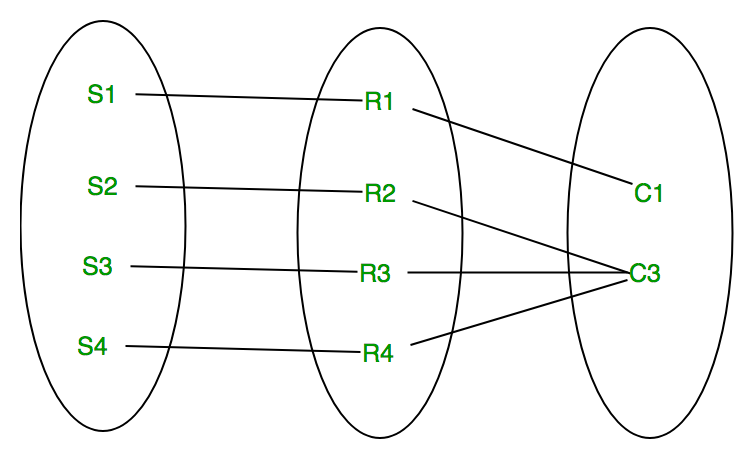


Using Sets, it can be represented as:



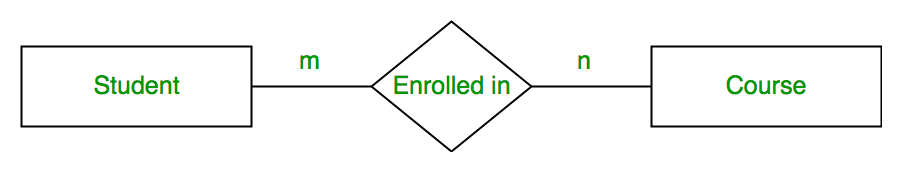
1. **Many to one –** When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken 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.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Database-Management-System-ER-Model-14.png)Using Sets, it can be represented as:

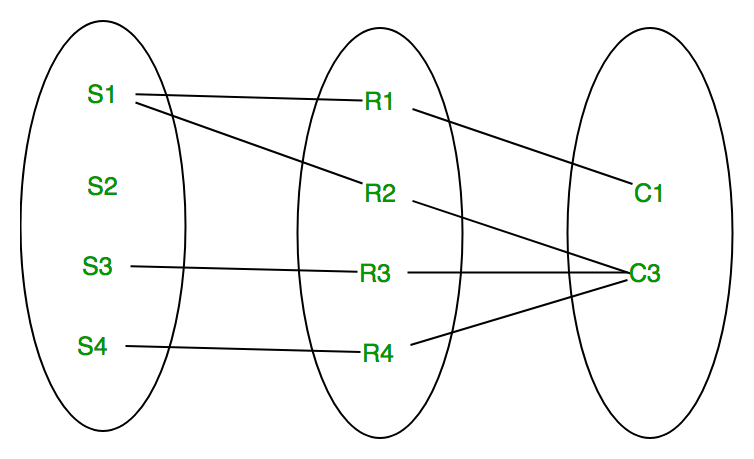


In this case, each student is taking only 1 course but 1 course has been taken by many students.

1. **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:

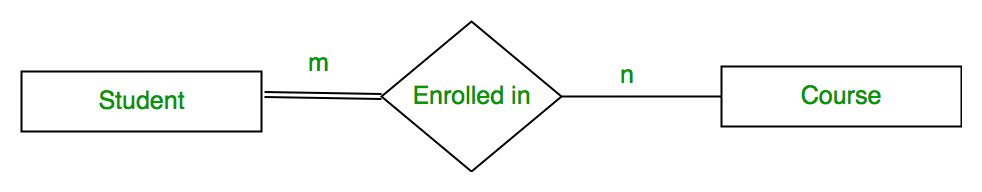


In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

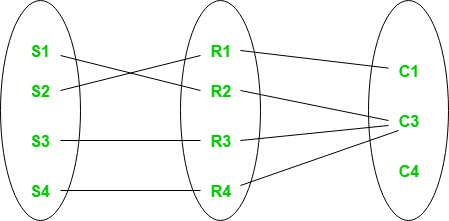
**Participation Constraint:**  
Participation Constraint is applied on the entity participating in the relationship set.

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,

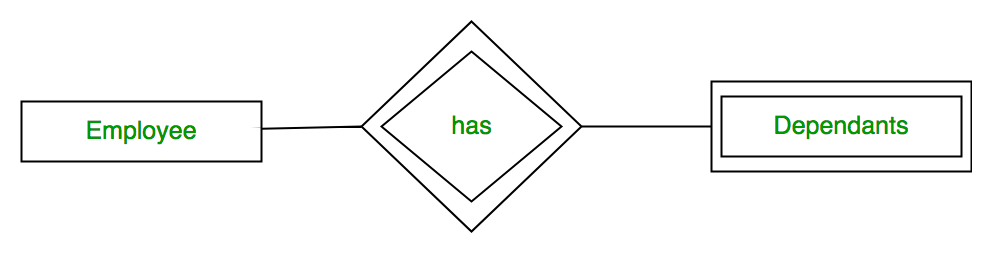


Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

**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 dependants (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 Dependant.

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.



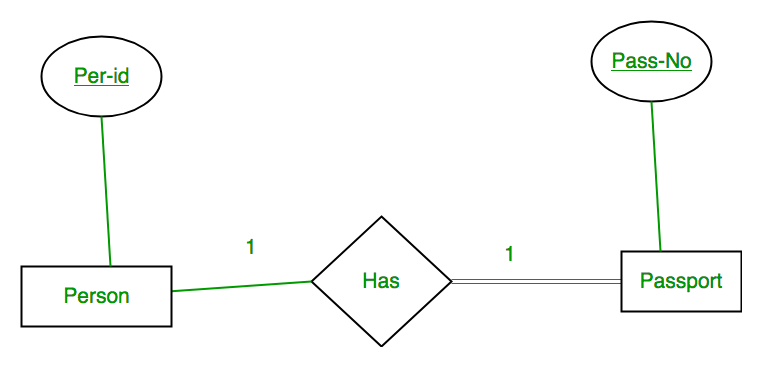
# Mapping from ER Model to Relational Model

To understand this, you should have an idea about:

[ER model](http://quiz.geeksforgeeks.org/database-management-system-er-model/) - [Relation model](http://quiz.geeksforgeeks.org/relational-model/" \t "_blank)

After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc.  In this article we will discuss how to convert ER diagram to Relational Model for different scenarios.

**Case 1:  Binary Relationship with 1:1 cardinality with total participation of an entity**

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Binary-Relationship-with-1_1-cardinality-with-total-participation-of-an-entity.png)A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.

**First Convert each entity and relationship to tables.**  Person table corresponds to Person Entity with key as Per-Id. Similarly Passport table corresponds to Passport Entity with key as Pass-No. Has Table represents relationship between Person and Passport (Which person has which passport). So it will take attribute Per-Id from Person and Pass-No from Passport.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Person** | |  | **Has** | |  | **Passport** | |
| **Per-Id** | **Other Person Attribute** | **Per-Id** | **Pass-No** | **Pass-No** | **Other PassportAttribute** |
| PR1 | – | PR1 | PS1 | PS1 | – |
| PR2 | – | PR2 | PS2 | PS2 | – |
| PR3 | – |  |  |  |  |  |  |

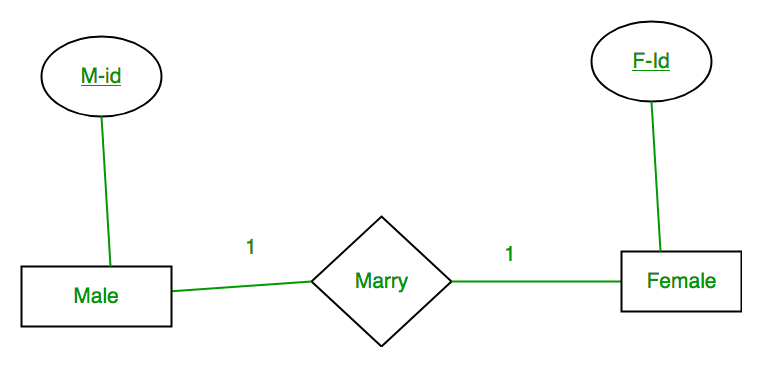
**Table 1**

As we can see from Table 1, each Per-Id and Pass-No has only one entry in **Has** table. So we can merge all three tables into 1 with attributes shown in Table 2. Each Per-Id will be unique and not null. So it will be the key. Pass-No can’t be key because for some person, it can be NULL.

|  |  |  |  |
| --- | --- | --- | --- |
| Per-Id | Other Person Attribute | Pass-No | Other Passport Attribute |

**Table 2**

**Case 2: Binary Relationship with 1:1 cardinality and partial participation of both entities**

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Binary-Relationship-with-1_1-cardinality-and-partial-participation-of-both-entities.png)

A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both. First Convert each entity and relationship to tables.  Male table corresponds to Male Entity with key as M-Id. Similarly Female table corresponds to Female Entity with key as F-Id. Marry Table represents relationship between Male and Female (Which Male marries which female). So it will take attribute M-Id from Male and F-Id from Female.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Male** | |  | **Marry** | |  | **Female** | |
| M-Id | Other Male Attribute | M-Id | F-Id | F-Id | Other FemaleAttribute |
| M1 | – | M1 | F2 | F1 | – |
| M2 | – | M2 | F1 | F2 | – |
| M3 | – |  |  |  |  | F3 | – |

**Table 3**

As we can see from Table 3, some males and some females do not marry. If we merge 3 tables into 1, for some M-Id, F-Id will be NULL and for some F-Id, M-Id will be NULL. So there is no attribute which is always not NULL. So we can’t merge all three tables into 1. We can convert into 2 tables. In table 4, M-Id who are married will have F-Id associated. For others, it will be NULL. Table 5 will have information of all females. Primary Keys have been underlined.

|  |  |  |
| --- | --- | --- |
| M-Id | Other Male Attribute | F-Id |

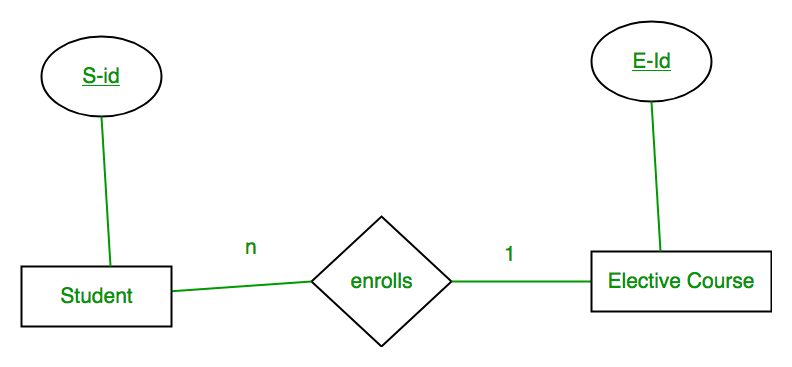
**Table 4**

|  |  |
| --- | --- |
| F-Id | Other FemaleAttribute |

**Table 5**

**Note:** Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship. If atleast 1 entity has total participation, number of tables required will be 1.

**Case 3: Binary Relationship with n: 1 cardinality**

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Binary-Relationship-with-n_-1-cardinality.png)In this scenario, every student can enroll only in one elective course but for an elective course there can be more than one student. First Convert each entity and relationship to tables.  Student table corresponds to Student Entity with key as S-Id. Similarly Elective\_Course table corresponds to Elective\_Course Entity with key as E-Id. Enrolls Table represents relationship between Student and Elective\_Course (Which student enrolls in which course). So it will take attribute S-Id from and Student E-Id from Elective\_Course.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Student** | |  | **Enrolls** | |  | **Elective\_Course** | |
| S-Id | Other Student Attribute | S-Id | E-Id | E-Id | Other Elective CourseAttribute |
| S1 | – | S1 | E1 | E1 | – |
| S2 | – | S2 | E2 | E2 | – |
| S3 | – |  | S3 | E1 |  | E3 | – |
| S4 | – |  | S4 | E1 |  |  |  |

**Table 6**

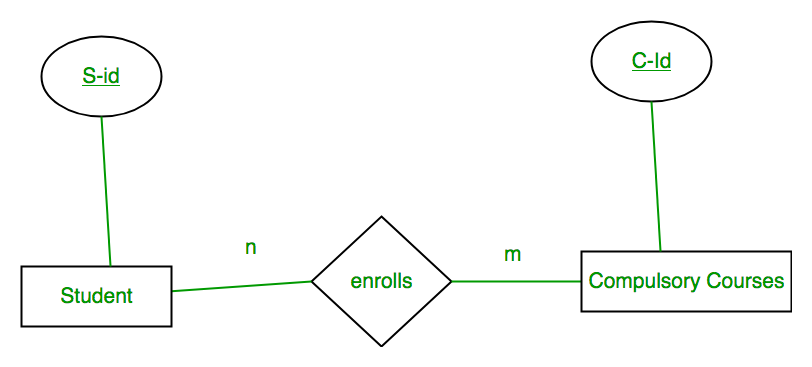
As we can see from Table 6, S-Id is not repeating in Enrolls Table. So it can be considered as a key of Enrolls table. Both Student and Enrolls Table’s key is same; we can merge it as a single table. The resultant tables are shown in Table 7 and Table 8. Primary Keys have been underlined.

|  |  |  |
| --- | --- | --- |
| S-Id | Other Student Attribute | E-Id |

**Table 7**

|  |  |
| --- | --- |
| E-Id | Other Elective CourseAttribute |

**Table 8**

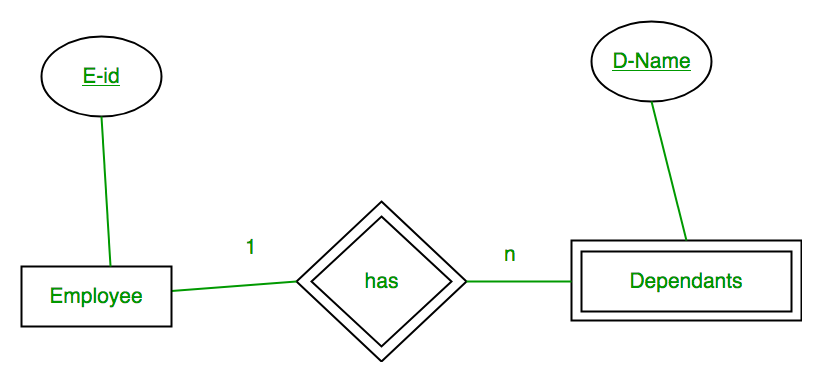
**Case 4: Binary Relationship with m: n cardinality[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Binary-Relationship-with-m_-n-cardinality.png)**

In this scenario, every student can enroll in more than 1 compulsory course and for a compulsory course there can be more than 1 student. First Convert each entity and relationship to tables.  Student table corresponds to Student Entity with key as S-Id. Similarly Compulsory\_Courses table corresponds to Compulsory Courses Entity with key as C-Id. Enrolls Table represents relationship between Student and Compulsory\_Courses (Which student enrolls in which course). So it will take attribute S-Id from Person and C-Id from Compulsory\_Courses.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Student** | |  | **Enrolls** | |  | **Compulsory\_Courses** | |
| S-Id | Other Student Attribute | S-Id | C-Id | C-Id | Other Compulsory CourseAttribute |
| S1 | – | S1 | C1 | C1 | – |
| S2 | – | S1 | C2 | C2 | – |
| S3 | – |  | S3 | C1 |  | C3 | – |
| S4 | – |  | S4 | C3 |  | C4 | – |
|  |  |  | S4 | C2 |  |  |  |
|  |  |  | S3 | C3 |  |  |  |

**Table 9**

As we can see from Table 9, S-Id and C-Id both are repeating in Enrolls Table. But its combination is unique; so it can be considered as a key of Enrolls table. All tables’ keys are different, these can’t be merged.  Primary Keys of all tables have been underlined.

**Case 5: Binary Relationship with weak entity[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Binary-Relationship-with-weak-entity.png)**

In this scenario, an employee can have many dependants and one dependant can depend on one employee. A dependant does not have any existence without an employee (e.g; you as a child can be dependant of your father in his company). So it will be a weak entity and its participation will always be total. Weak Entity does not have key of its own. So its key will be combination of key of its identifying entity (E-Id of Employee in this case) and its partial key (D-Name).

First Convert each entity and relationship to tables.  Employee table corresponds to Employee Entity with key as E-Id. Similarly Dependants table corresponds to Dependant Entity with key as  D-Name and E-Id. Has Table represents relationship between Employee and Dependants (Which employee has which dependants). So it will take attribute E-Id from Employee and D-Name from Dependants.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Employee** | |  | **Has** | |  | **Dependants** | | |
| E-Id | Other Employee Attribute | E-Id | D-Name | D-Name | E-Id | Other DependantsAttribute |
| E1 | – | E1 | RAM | RAM | E1 | – |
| E2 | – | E1 | SRINI | SRINI | E1 | – |
| E3 | – | E2 | RAM | RAM | E2 | – |
|  |  | E3 | ASHISH | ASHISH | E3 | – |

**Table 10**

As we can see from Table 10, E-Id, D-Name is key for **Has** as well as Dependants Table. So we can merge these two into 1. So the resultant tables are shown in Tables 11 and 12. Primary Keys of all tables have been underlined.

|  |  |
| --- | --- |
| E-Id | Other Employee Attribute |

**Table 11**

|  |  |  |
| --- | --- | --- |
| D-Name | E-Id | Other DependantsAttribute |

Relation algebra Basic operators

**Six basic operators**

**– select: σ**

**– project: ∏**

**– union: ∪**

**– set difference: –**

**– Cartesian product: x**

**– rename: ρ**

**• The operators take one or two relations as inputs and produce a new relation as a result**

Relational Algebra is a procedural query language which takes relations as an input and returns relation as an output. There are some basic operators which can be applied on relations to produce required results which we will discuss one by one. We will use STUDENT\_SPORTS, EMPLOYEE and STUDENT relations as given in Table 1, Table 2 and Table 3 respectively to understand the various operators.

**Table 1 : STUDENT\_SPORTS Table 2 : EMPLOYEE**

|  |  |
| --- | --- |
| **ROLL\_NO** | **SPORTS** |
| 1 | Badminton |
| 2 | Cricket |
| 2 | Badminton |
| 4 | Badminton |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 5 | NARESH | HISAR | 9782918192 | 22 |
| 6 | SWETA | RANCHI | 9852617621 | 21 |
| 4 | SURESH | DELHI | 9156768971 | 18 |

**Table 3 : STUDENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9156768971 | 18 |

***Selection operator (σ):***Selection operator is used to select tuples from a relation based on some condition. Syntax:

**σ (Cond)(Relation Name)**

Extract students whose age is greater than 18 from STUDENT relation given in Table 1

**σ (AGE>18)(STUDENT)**

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |

***Projection Operator (∏):*** Projection operator is used to project particular columns from a relation. Syntax:

**∏(Column 1,Column 2….Column n)(Relation Name)**

Extract ROLL\_NO and NAME from STUDENT relation given in Table 1

**∏(ROLL\_NO,NAME)(STUDENT)**

**RESULT:**

|  |  |
| --- | --- |
| **ROLL\_NO** | **NAME** |
| 1 | RAM |
| 2 | RAMESH |
| 3 | SUJIT |
| 4 | SURESH |

If resultant relation after projection has duplicate rows, it will be removed. For Example:  **∏(ADDRESS)(STUDENT)** will remove one duplicate row with value DELHI and return three rows.

***Cross Product(X):*** Cross product is used to join two relations. For every row of Relation1, each row of Relation2 is concatenated. If Relation1 has m tuples and and Relation2 has n tuples, cross product of Relation1 and Relation2 will have m X n tuples. Syntax:

**Relation1 X Relation2**

To apply Cross Product on STUDENT relation given in Table 1 and STUDENT\_SPORTS relation given in Table 2,

**STUDENT X STUDENT\_SPORTS**

**RESULT:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** | **ROLL\_NO** | **SPORTS** |
| 1 | RAM | DELHI | 9455123451 | 18 | 1 | Badminton |
| 1 | RAM | DELHI | 9455123451 | 18 | 2 | Cricket |
| 1 | RAM | DELHI | 9455123451 | 18 | 2 | Badminton |

***Union (U):***Union on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (These two relation should have same number of attributes and corresponding attributes in two relations have same domain) . Union operator when applied on two relations R1 and R2 will give a relation with tuples which are either in R1 or in R2. The tuples which are in both R1 and R2 will appear only once in result relation. Syntax:

**Relation1 U Relation2**

Find person who are either student or employee, we can use Union operator like:

**STUDENT U EMPLOYEE**

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9156768971 | 18 |
| 5 | NARESH | HISAR | 9782918192 | 22 |
| 6 | SWETA | RANCHI | 9852617621 | 21 |

***Minus (-):*** Minus on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible**. Minus operator when applied on two relations as R1-R2 will give a relation with tuples which are in R1 but not in R2. Syntax:

**Relation1 - Relation2**

Find person who are student but not employee, we can use minus operator like:

**STUDENT - EMPLOYEE**

**RESULT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |

***Rename(ρ):***Rename operator is used to give another name to a relation. Syntax:

**ρ(Relation2, Relation1)**

To rename STUDENT relation to STUDENT1, we can use rename operator like:

**ρ(STUDENT1, STUDENT)**

If you want to create a relation STUDENT\_NAMES with ROLL\_NO and NAME from STUDENT, it can be done using rename operator as:

**ρ(STUDENT\_NAMES, ∏(ROLL\_NO, NAME)(STUDENT))**

# What is Relational Calculus?

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, **Relational Calculus** in non-procedural query language and has no description about how the query will work or the data will b fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

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

## Tuple Relational Calculus (TRC)

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

**Syntax:** { T | Condition }

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

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

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

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

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

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

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

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

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

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

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

For example, to specify the range of a tuple variable S as the Staff relation, we write:

Staff(S)

To express the query 'Find the set of all tuples S such that F(S) is true', we can write:

{S | F(S)}

Here, F is called a formula (well-formed formula, or wff in mathematical logic). For example, to express the query 'Find the staffNo, fName, lName, position, sex, DOB, salary, and branchNo of all staff earning more than £10,000', we can write:

{S | Staff(S) ∧ S.salary > 10000}

**Example:**

{t | TEACHER (t) and t.SALARY>20000}

- It implies that it selects the tuples from TEACHER in such a way that the resulting teacher tuples will have the salary greater than 20000. This is an example of selecting a range of values.

{t | TEACHER (t) AND t.DEPT\_ID = 6}

- T select all the tuples of teachers name who work under Department 8.  Any tuple variable with 'For All' (?) or 'there exists' (?) condition is termed as a bound variable. In the last example, for any range of values of SALARY greater than 20000, the meaning of the condition does not alter. Bound variables are those ranges of tuple variables whose meaning will not alter if the tuple variable is replaced by another tuple variable.

In the second example, you have used DEPT\_ID= 8 which means only for DEPT\_ID = 8 display the teacher details. Such variable is called free variable. Any tuple variable without any 'For All' or 'there exists' condition is called Free Variable.

Domain Relational Calculus (DRC)

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

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

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

For example,

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

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

**EXAMPLE 2**

 A domain relational calculus expression has the following general format:

{d1, d2, . . . , dn | F(d1, d2, . . . , dm)} m ≥ n

where d1, d2, . . . , dn, . . . , dm stand for domain variables and F(d1, d2, . . . , dm) stands for a formula composed of atoms.

Example:  
select TCHR\_ID and TCHR\_NAME of teachers who work for department 8, (where suppose - dept. 8 is Computer Application Department)

{<tchr\_id, tchr\_name=""> | <tchr\_id, tchr\_name=""> ? TEACHER Λ DEPT\_ID = 10}

Get the name of the department name where Karlos works:

{DEPT\_NAME |< DEPT\_NAME > ? DEPT Λ ? DEPT\_ID ( ? TEACHER Λ TCHR\_NAME = Karlos)}

It is to be noted that these queries are safe. The use domain relational calculus is restricted to safe expressions; moreover, it is equivalent to the tuple relational calculus which in turn is similar to the relational algebra.