

Entity Relationship Diagram

When a company asks you to make them a working, functional Database Management System ([DBMS](#)) which they can work with, there are certain steps to follow. Let us summarize them here:

1. **Gathering information:** This could be a written document that describes the system in question with reasonable amount of details.
2. **Producing ERD:** ERD or Entity Relationship Diagram is a diagrammatic representation of the description we have gathered about the system.
3. **Designing the database:** Out of the ERD we have created, it is very easy to determine the tables, the attributes which the tables must contain and the relationship among these tables.
4. **Normalization:** This is a process of removing different kinds of impurities from the tables we have just created in the above step.

How to Prepare an ERD

Step 1

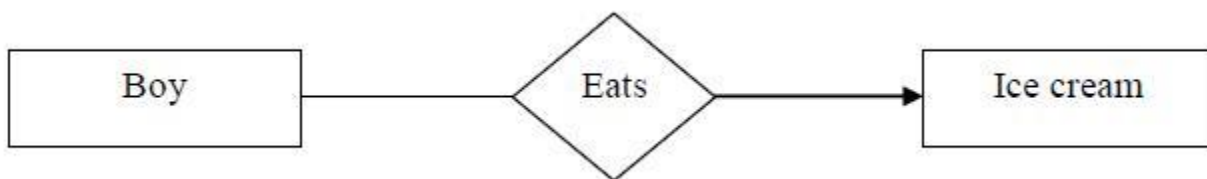
Let us take a very simple example and we try to reach a fully organized database from it. Let us look at the following simple statement:

A boy eats an ice cream.

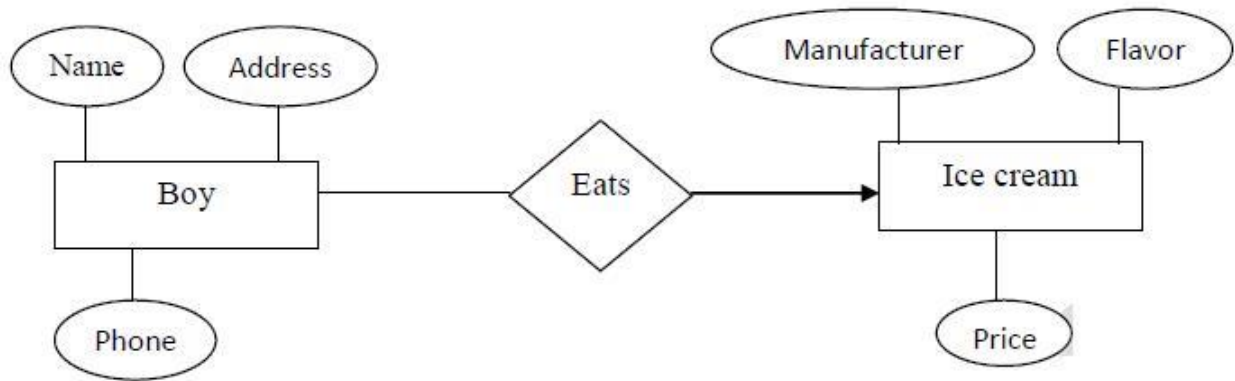
This is a description of a real word activity, and we may consider the above statement as a written document (very short, of course).

Step 2

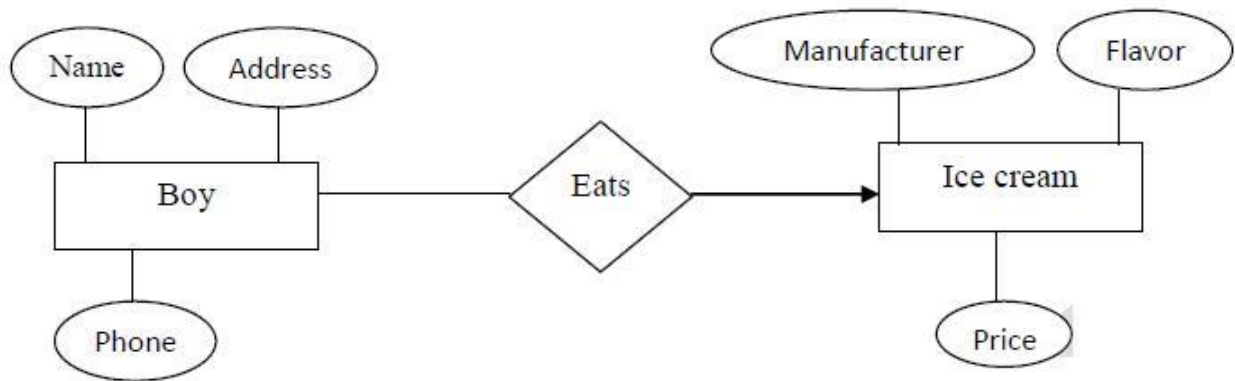
Now we have to prepare the ERD. Before doing that we have to process the statement a little. We can see that the sentence contains a subject (*boy*), an object (*ice cream*) and a verb (*eats*) that defines the relationship between the subject and the object. **Consider the nouns as entities (*boy* and *ice cream*) and the verb (*eats*) as a relationship.** To plot them in the diagram, put the nouns within rectangles and the relationship within a diamond. Also, show the relationship with a directed arrow, starting from the subject entity (*boy*) towards the object entity (*ice cream*).



Well, fine. Up to this point the ERD shows how *boy* and *ice cream* are related. Now, every boy must have a name, address, phone number etc. and every ice cream has a manufacturer, flavor, price etc. Without these the diagram is not complete. These items which we mentioned here are known as attributes, and they must be incorporated in the ERD as connected ovals.



But can only entities have attributes? Certainly not. If we want then the relationship must have their attributes too. These attribute do not inform anything more either about the *boy* or the *ice cream*, but they provide additional information about the relationships between the *boy* and the *ice cream*.



Step 3

We are almost complete now. If you look carefully, we now have defined structures for at least three tables like the following:

Boy

Name	Address	Phone
------	---------	-------

Ice Cream

Manufacturer	Flavor	Price
--------------	--------	-------

Eats

Date	Time
------	------

However, this is still not a working database, because by definition, database should be “collection of related tables.” To make them connected, the tables must have some common attributes. If we chose the attribute Name of the **Boy** table to play the role of the common attribute, then the revised structure of the above tables become something like the following.

Boy

Name	Address	Phone
------	---------	-------

Ice Cream

Manufacturer	Flavor	Price	Name
--------------	--------	-------	------

Eats

Date	Time	Name
------	------	------

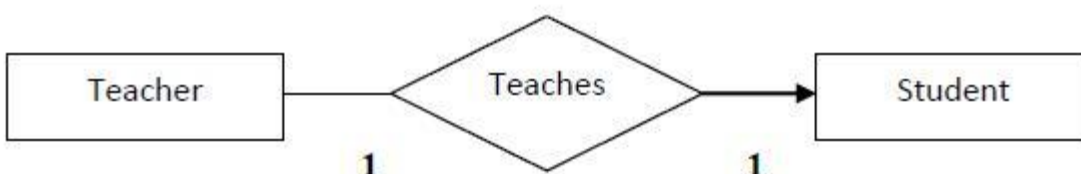
his is as complete as it can be. We now have information about the boy, about the ice cream he has eaten and about the date and time when the eating was done.

Cardinality of Relationship

While creating relationship between two entities, we may often need to face the cardinality problem. This simply means that how many entities of the first set are related to how many entities of the second set. Cardinality can be of the following three types.

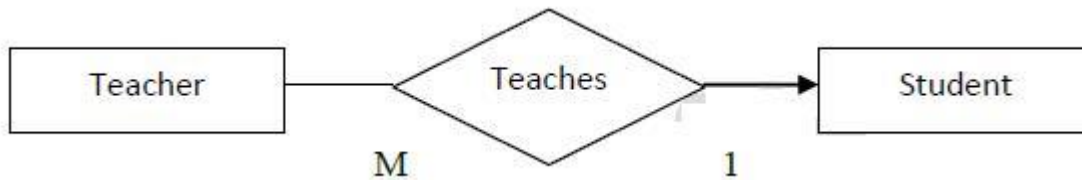
One-to-One

Only one entity of the first set is related to only one entity of the second set. E.g. *A teacher teaches a student*. Only one teacher is teaching only one student. This can be expressed in the following diagram as:



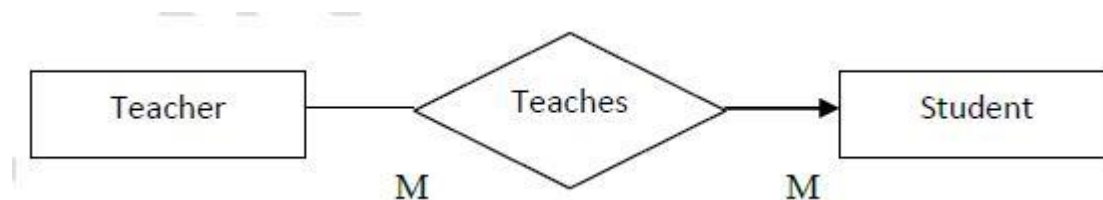
Many-to-One

Multiple entities of the first set are related to multiple entities of the second set. E.g. *Teachers teach a student*. Many teachers are teaching only one student. This can be expressed in the following diagram as:



Many-to-Many

Multiple entities of the first set is related to multiple entities of the second set. E.g. *Teachers teach students*. In any school or college many teachers are teaching many students. This can be considered as a two way one-to-many relationship. This can be expressed in the following diagram as:



In this discussion we have not included the attributes, but you can understand that they can be used without any problem if we want to.

The Concept of Keys

A key is an attribute of a table which helps to identify a row. There can be many different types of keys which are explained here.

Super Key or Candidate Key: It is such an attribute of a table that can uniquely identify a row in a table. Generally they contain unique values and can never contain NULL values. There can be more than one super key or candidate key in a table e.g. within a STUDENT table Roll and Mobile No. can both serve to uniquely identify a student.

Primary Key: It is one of the candidate keys that are chosen to be the identifying key for the entire table. E.g. although there are two candidate keys in the STUDENT table, the college would obviously use Roll as the primary key of the table.

Alternate Key: This is the candidate key which is not chosen as the primary key of the table. They are named so because although not the primary key, they can still identify a row.

Composite Key: Sometimes one key is not enough to uniquely identify a row. E.g. in a single class Roll is enough to find a student, but in the entire school, merely searching by the Roll is not enough, because there could be 10 classes in the school and each one of them may contain a certain roll no 5. To uniquely identify the student we have to say something like “class VII, roll no 5”. So,

a combination of two or more attributes is combined to create a unique combination of values, such as Class + Roll.

Foreign Key: Sometimes we may have to work with an attribute that does not have a primary key of its own. To identify its rows, we have to use the primary attribute of a related table. Such a copy of another related table's primary key is called foreign key.

Strong and Weak Entity

Based on the concept of foreign key, there may arise a situation when we have to relate an entity having a primary key of its own and an entity not having a primary key of its own. In such a case, the entity having its own primary key is called a strong entity and the entity not having its own primary key is called a weak entity. Whenever we need to relate a strong and a weak entity together, the ERD would change just a little.

Say, for example, we have a statement “A *Student lives in a Home*.” STUDENT is obviously a strong entity having a primary key Roll. But HOME may not have a unique primary key, as its only attribute Address may be shared by many homes (what if it is a housing estate?). HOME is a weak entity in this case.

The ERD of this statement would be like the following



As you can see, the weak entity itself and the relationship linking a strong and weak entity must have double border.

Different Types of Database

There are three different types of data base. The difference lies in the organization of the database and the storage structure of the data. We shall briefly mention them here.

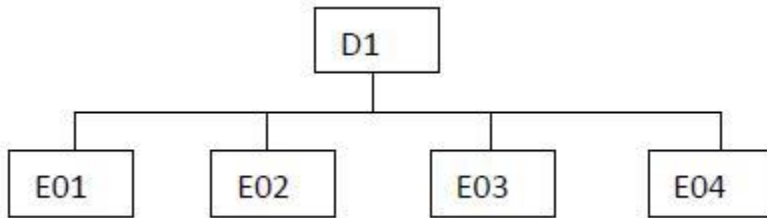
Relational DBMS

This is our subject of study. A DBMS is relational if the data is organized into relations, that is, tables. In RDBMS, all data are stored in the well-known row-column format.

Hierarchical DBMS

In HDBMS, data is organized in a tree like manner. There is a parent-child relationship among data items and the data model is very suitable for representing one-to-many relationship. To access the data items, some kind of tree-traversal techniques are used, such as preorder traversal.

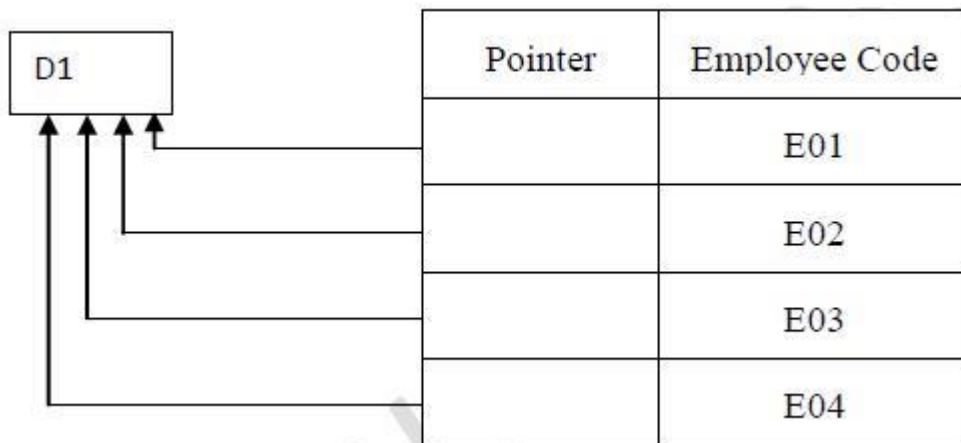
Because HDBMS is built on the one-to-many model, we have to face a little bit of difficulty to organize a hierarchical database into row column format. For example, consider the following hierarchical database that shows four employees (E01, E02, E03, and E04) belonging to the same department D1.



There are two ways to represent the above one-to-many information into a relation that is built in one-to-one relationship. The first is called **Replication**, where the department id is replicated a number of times in the table like the following.

Dept-Id	Employee Code
D1	E01
D1	E02
D1	E03
D1	E04

Replication makes the same data item redundant and is an inefficient way to store data. A better way is to use a technique called the **Virtual Record**. While using this, the repeating data item is not used in the table. It is kept at a separate place. The table, instead of containing the repeating information, contains a pointer to that place where the data item is stored.



This organization saves a lot of space as data is not made redundant.

Network DBMS

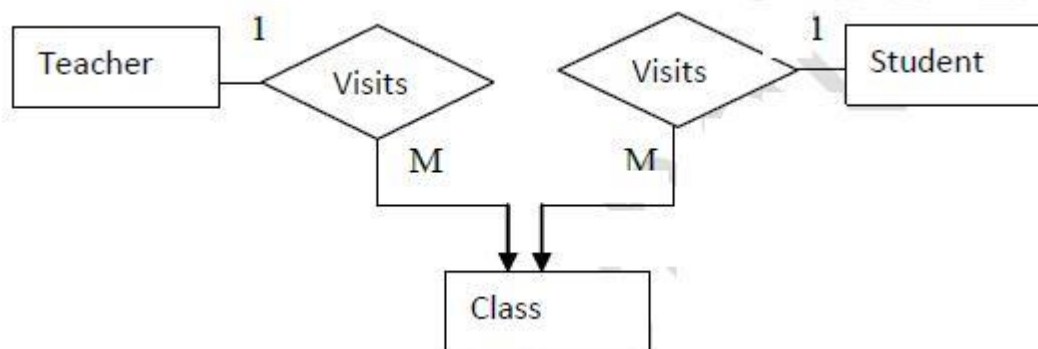
The NDBMS is built primarily on a one-to-many relationship, but where a parent-child representation among the data items cannot be ensured. This may happen in any real world situation where any entity can be linked to any entity. The NDBMS was proposed by a group of theorists known as the **Database Task Group (DBTG)**. What they said looks like this:

In NDBMS, all entities are called **Records** and all relationships are called **Sets**. The record from where the relationship starts is called the **Owner Record** and where it ends is called **MemberRecord**. The relationship or set is strictly one-to-many.



In case we need to represent a many-to-many relationship, an interesting thing happens. In NDBMS, Owner and Member can only have one-to-many relationship. We have to introduce a third common record with which both the Owner and Member can have one-to-many relationship. Using this common record, the Owner and Member can be linked by a many-to-many relationship.

Suppose we have to represent the statement *Teachers teach students*. We have to introduce a third record, suppose CLASS to which both teacher and the student can have a many-to-many relationship. Using the class in the middle, teacher and student can be linked to a virtual many-to-many relationship.



Problems with E-R Model

The E-R model can result problems due to limitations in the way the entities are related in the relational databases. These problems are called connection traps. These problems often occur due to a misinterpretation of the meaning of certain relationships.

Two main types of connection traps are called fan traps and chasm traps.

Fan Trap. It occurs when a model represents a relationship between entity types, but pathway between certain entity occurrences is ambiguous.

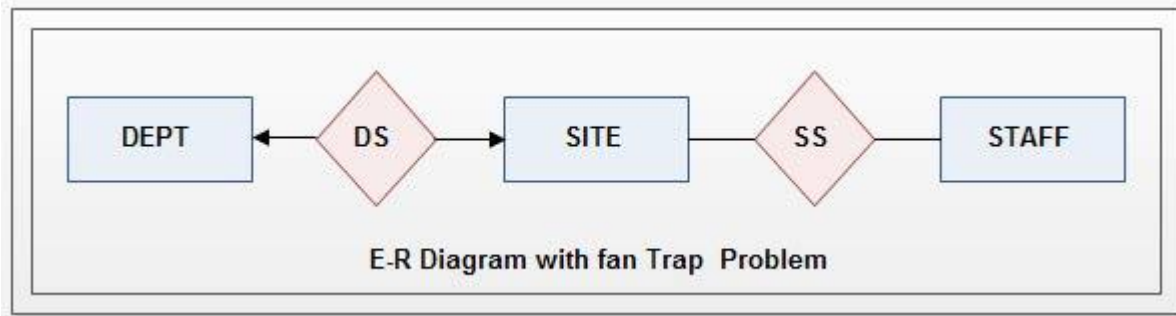
Chasm Trap. It occurs when a model suggests the existence of a relationship between entity types, but pathway does not exist between certain entity occurrences.

Now, we will discuss the each trap in detail address)

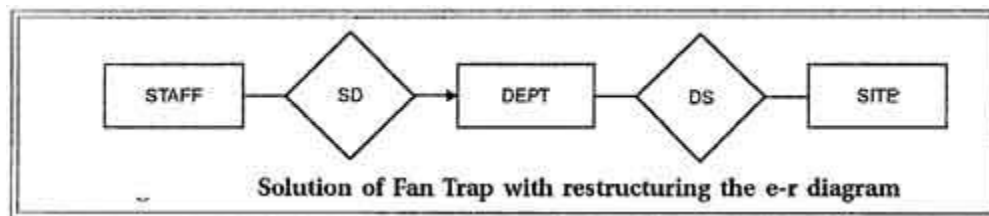
Fan Trap

A fan trap occurs when one to many relationships fan out from a single entity.

For example: Consider a database of Department, Site and Staff, where one site can contain number of department, but a department is situated only at a single site. There are multiple staff members working at a single site and a staff member can work from a single site. The above case is represented in e-r diagram shown.

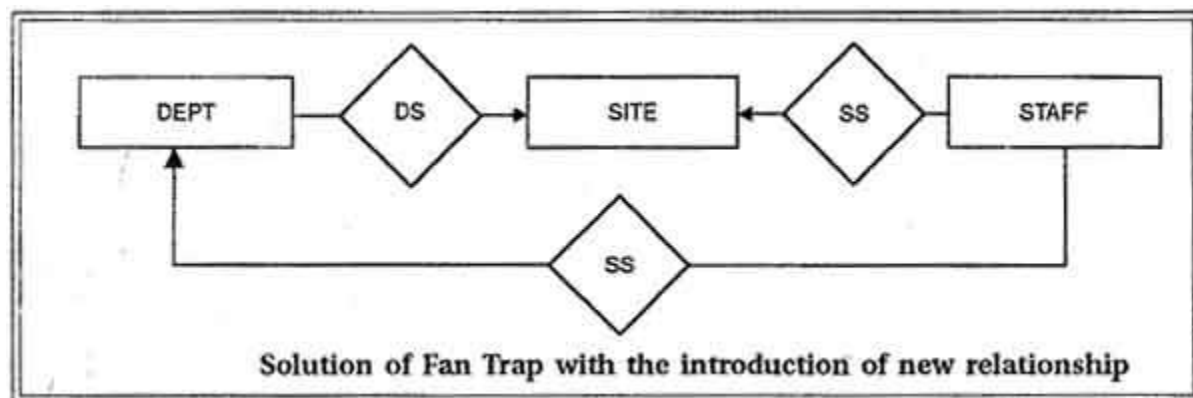


The problem of above e-r diagram is that, which staff works in a particular department remain answered. The solution is to restructure the original E-R model to represent the correct association as shown.

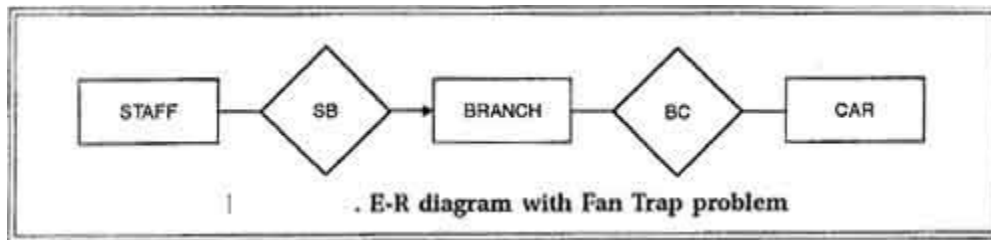


In other words the two entities should have a direct relationship between them to provide the necessary [information](#).

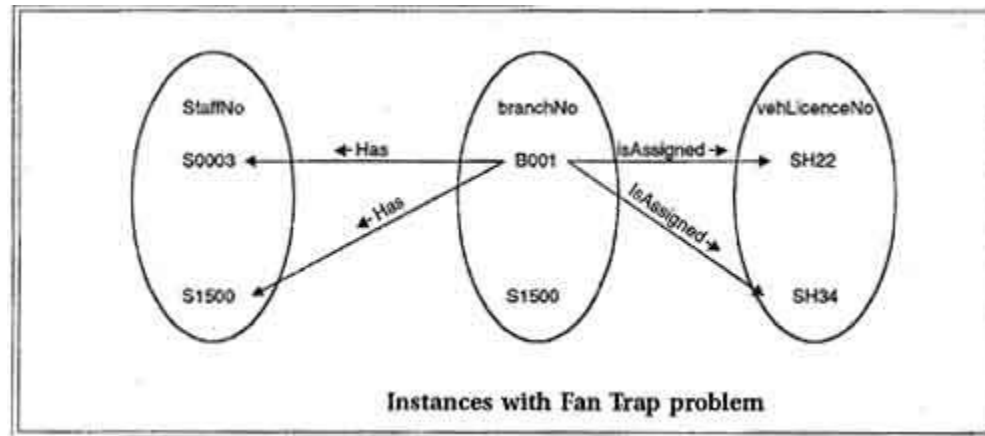
There is one another way to solve the problem of e-r diagram of figure, by introducing direct relationship between DEPT and STAFF as shown in figure.



Another example: Let us consider another case, where one branch contains multiple staff members and cars, which are represented.



The problem of above E-R diagram is that, it is unable to tell which member of staff uses a particular, which is represented. It is not possible tell which member of staff uses' car SH34.





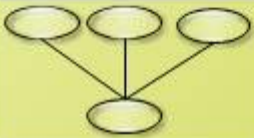

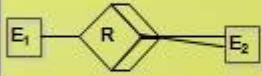
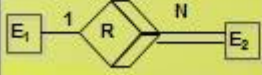
E-R NOTATION


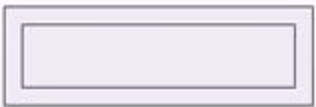



There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation.

All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of connection. The symbols used for the basic ER constructs are:

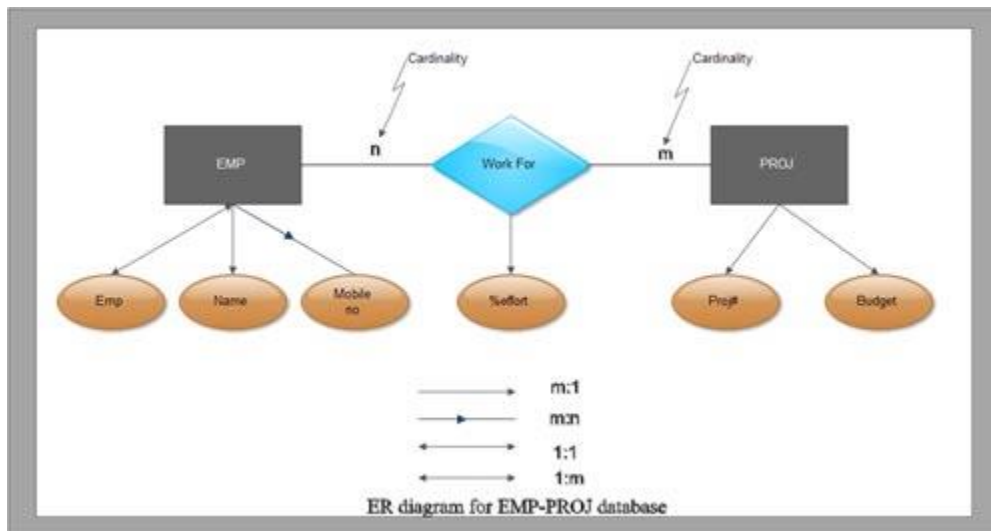
- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Attributes are represented by Ellipses.
- A solid line connecting two entities represents relationships. The name of the relationship is written above the line. Relationship names should be verbs and diamonds sign is used to represent relationship sets.
- Attributes, when included, are listed inside the entity rectangle. Attributes, which are identifiers, are underlined. Attribute names should be singular nouns.
- Multi-valued attributes are represented by double ellipses.
- Directed line is used to indicate one occurrence and undirected line is used to indicate many occurrences in a relation.

The symbols used to design an ER diagram are shown.

Symbols used in E-R diagram	
Symbol	Meaning
	Key Attribute
	Multivaluated Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Relation 1:N for $E_1:E_2$ in R

SYMBOL	MEANING
	Entity Type
	Weak Entity Type
	Relationship Type
	Identifying Relationship Type
	Attribute

The ER diagram showing the usage of different symbols



What is attributes?

Attributes means characteristics. For instance, in a database or a spreadsheet you can apply attributes to each *field* or *cell* to customize your document. As a general attribute, you can choose whether it is to be a text field or a numeric field or perhaps a computed field, whose value the application calculates for you.

Then you can apply more specific attributes to the field or cell, such as making the text bold and right-aligned and perhaps in a particular typeface. If a field is numeric, you will have other attribute options available, such as how many decimal places to display, whether to use a dollar sign or a percent symbol, or whether to start a formula in the cell.

What is a Database Architecture

An early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the DBTG (Data Base Task Group) appointed by the Conference on Data Systems and Languages (CODASYL, 1971). The DBTG recognized the need for a two level approach with a system view called the schema and user views called subschema. The American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC) produced a similar terminology mid architecture in 1975 (ANSI 1975). ANSI-SPARC recognized the need for a three level approach with a system catalog.

There are following three levels or layers 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 (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 view 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](#) 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.

Field Name

↓

RollNO	Name	Class	Address	TM	MO	%age
1234	Hitesh	BTech-III	23-Mall Asr	500	382	74
1249	Anand	MCA-II	144 Green Ave Asr	500	410	82
2315	Dimple	BCA-1	42 kashmir Ave Asr	1600	1120	70

↑

Field or Column
or attribute

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 types](#) of attributes (for example, integer, real, character) and their length (such as the maximum number

of digits or characters), but not any storage 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](#). 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](#). It interfaces with the [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 its entirety. Similarly there will be only one internal or physical view, representing the total database, as it is physically stored.

Schema

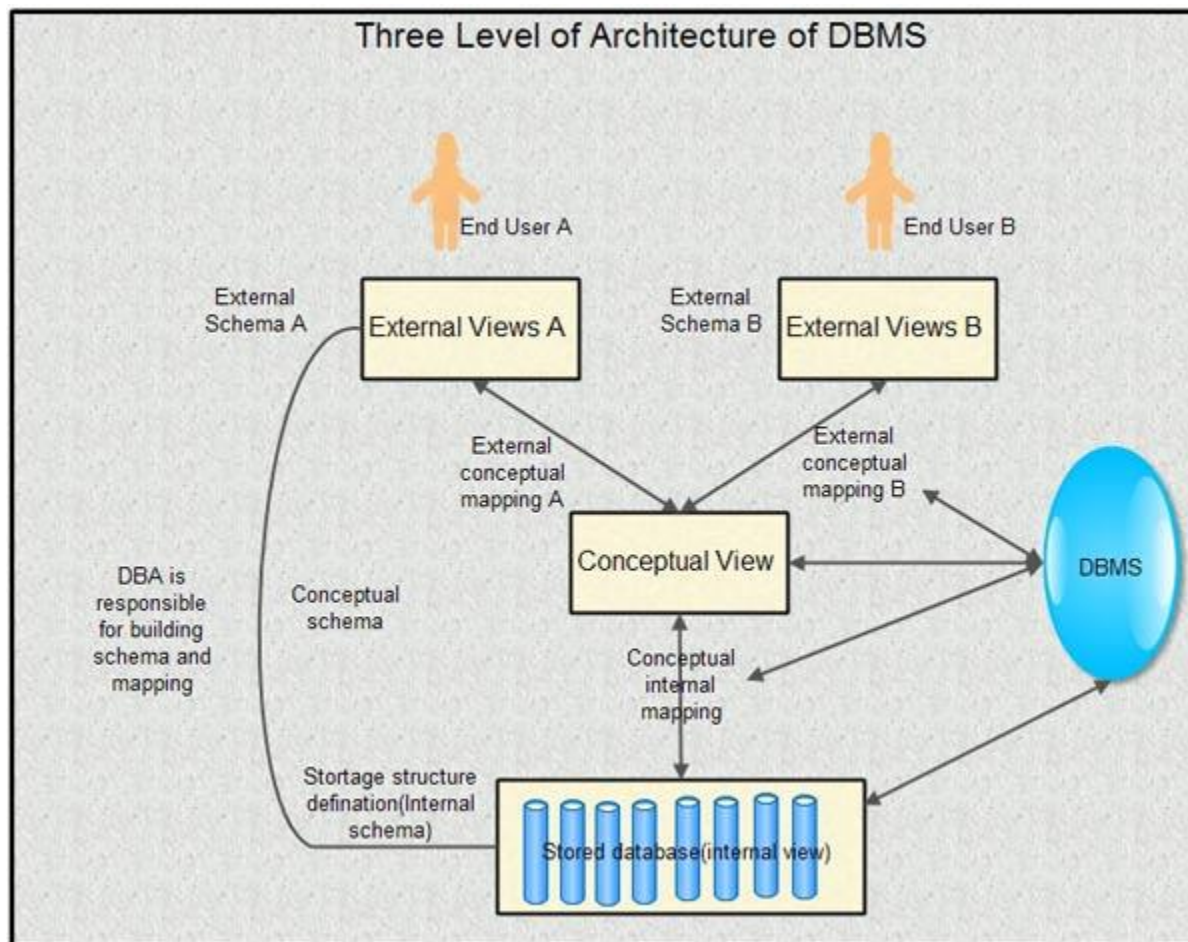
It is important to note that the data in the database changes frequently, while the plans or schemes remain the same over long periods of time. The database plans consist of types of entities that a database deals with, the relationship among these entities and the ways in which the entities and relationships 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.



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.

Relational Model

Relational model stores data in the form of tables. This concept purposed by Dr. E.F. Codd, a researcher of IBM in the year 1960s. The relational model consists of three major components:

1. The set of relations and set of domains that defines the way data can be represented (data structure).
2. Integrity rules that define the procedure to protect the data (data integrity).
3. The operations that can be performed on data (data manipulation).

A rational model database is defined as a database that allows you to group its data items into one or more independent tables that can be related to one another by using fields common to each related table.

Characteristics of Relational Database

Relational database systems have the following characteristics:

- The whole data is conceptually represented as an orderly arrangement of data into rows and columns, called a relation or table.
- All values are scalar. That is, at any given row/column position in the relation there is one and only one value.
- All operations are performed on an entire relation and result is an entire relation, a concept known as closure.

Dr. Codd, when formulating the relational model, chose the term "relation" because it was comparatively free of connotations, unlike, for example, the word "table". It is a common misconception that the relational model is so called because relationships are established between tables. In fact, the name is derived from the relations on whom it is based. Notice that the model requires only that data be conceptually represented as a relation, it does not specify how the data should be physically implemented. A relation is a relation provided that it is arranged in row and column format and its values are scalar. Its existence is completely independent of any physical representation.

Basic Terminology used in Relational Model

The figure shows a relation with the. Formal names of the basic components marked the entire structure is, as we have said, a relation.

Attributes	Emp_Code	Name	Year
Tuples	21130	Amar Jain	1
	30745	Kuldeep	3
	41894	Manoj	2
	51207	Rita bajaj	6

Tuples of a Relation

Each row of data is a tuple. Actually, each row is an n-tuple, but the "n-" is usually dropped.

Cardinality of a relation: The number of tuples in a relation determines its cardinality. In this case, the relation has a cardinality of 4.

Degree of a relation: Each column in the tuple is called an attribute. The number of attributes in a relation determines its degree. The relation in figure has a degree of 3.

Domains: A domain definition specifies the kind of data represented by the attribute.

More- particularly, a domain is the set of all possible values that an attribute may validly contain. Domains are often confused with [data types](#), but this is inaccurate. Data type is a physical concept while domain is a logical one. "Number" is a data type and "Age" is a domain. To give another example "StreetName" and "Surname" might both be represented as text fields, but they are obviously different kinds of text fields; they belong to different domains.

Domain is also a broader concept than data type, in that a domain definition includes a more specific description of the valid data. For example, the domain Degree A awarded, which represents the degrees awarded by a university. In the database schema, this attribute might be defined as Text [3], but it's not just any three-character string, it's a member of the set {BA, BS, MA, MS, PhD, LLB, MD}. Of course, not all domains can be defined by simply listing their values. Age, for example, contains a hundred or so values if we are talking about people, but tens of thousands if we are talking about museum exhibits. In such instances it's useful to define the domain in terms of the rules, which can be used to determine the membership of any specific value in the set of all valid values.

For example, Person Age could be defined as "an integer in the range 0 to 120" whereas Exhibit Age (age of any object for exhibition) might simply by "an integer equal to or greater than 0."

Body of a Relation: The body of the relation consists of an unordered set of zero or more tuples. There are some important concepts here. First the relation is unordered. Record numbers do not apply to relations. Second a relation with no tuples still qualifies as a relation. Third, a relation is a set. The items in a set are, by definition, uniquely identifiable. Therefore, for a table to qualify as a relation each record must be uniquely identifiable and the table must contain no duplicate records.

Keys of a Relation

It is a set of one or more columns whose combined values are *unique* among all occurrences in a given table. A key is the relational means of specifying uniqueness. Some different types of keys are:

Primary key is an attribute or a set of attributes of a relation which posses the properties of uniqueness and irreducibility (No subset should be unique). For example: Supplier number in S table is primary key, Part number in P table is primary key and the combination of Supplier number and Part Number in SP table is a primary key

Foreign key is the attributes of a table, which refers to the primary key of some another table. Foreign key permit only those values, which appears in the primary key of the table to which it refers or may be null (Unknown value). For example: SNO in SP table refers the SNO of S table, which is the primary key of S table, so we can say that SNO in SP table is the foreign key. PNO in SP table refers the PNO of P table, which is the primary key of P table, so we can say that PNO in SP table is the foreign key.

The database of Customer-Loan, which we discussed earlier for hierarchical model and network model, is now represented for Relational model as shown.

It can easily be understood that, this model is very simple and has no redundancy. The total database is divided into two tables. Customer table contains the [information](#) about the customers with CNO as the primary key. The Customer_Loan table stores the information about CNO, LNO and AMOUNT. It has the primary key combination of CNO and LNO. Here, CNO also acts as the foreign key and refers to CNO of Customer table. It means, only those customer number are allowed in transaction table Customer_Loan that have their entry in the master Customer table.

Customer Table			
CNO	NAME	ADDRESS	CITY
C1	Rahal	Thapar campus	Patiala
C2	Ruhi	Tagore Nagar	Jalandhar
C3	Chachat	Dharampura	Qadian
C4	Pooja	GNDU	Amritsar

Customer_Loan Table		
CNO	LNO	AMOUNT
C1	L1	10000
C2	L1	10000
C3	L2	15000
C3	L3	25000
C4	L4	35000

Relational Model of Customer-Loan database

Relational View of Sample database

Let us take an example of a sample database consisting of supplier, parts and shipments tables. The table structure and some sample records for supplier, parts and shipments tables are given as Tables as shown below:

The Supplier Records			
Sno	Name	Status	Status
S1	Suneet	20	Qadian
S2	Ankit	10	Amritsar
S3	Amit	10	Qadian

The Part Records				
Pno	Name	Status	Weight	City
P1	Nut	Red	12	Qadian
P2	Bolt	Green	17	Amritsar
P3	Screw	Blue	17	Jalandhar
P4	Screw	Red	14	Qadian

The Shipment Records		
Sno	Name	Qty
S1	P1	250
S1	P2	300
S1	P3	500
S2	P1	250
S2	P2	500
S3	P2	300

As we discussed earlier, we assume that each row in Supplier table is identified by a unique SNo (Supplier Number), which uniquely identifies the entire row of the table. Likewise each part has a unique PNo (Part Number). Also, we assume that no more than one shipment exists for a given supplier/part combination in the shipments table.

Note that the relations Parts and Shipments have PNo (Part Number) in common and Supplier and Shipments relations have SNo (Supplier Number) in common. The Supplier and Parts relations have City in common. For example, the fact that supplier S3 and part P2 are located in the same city is represented by the appearance of the same value, Amritsar, in the city column of the two tuples in relations.

Operations in Relational Model

The four basic operations Insert, Update, Delete and Retrieve operations are shown below on the sample database in relational model:

Insert Operation: Suppose we wish to insert the information of supplier who does not supply any part, can be inserted in S table without any anomaly e.g. S4 can be inserted in Stable. Similarly, if we wish to insert information of a new part that is not supplied by any supplier can be inserted into a P table. If a supplier starts supplying any new part, then this information can be stored in shipment

table SP with the supplier number, part number and supplied quantity. So, we can say that insert operations can be performed in all the cases without any anomaly.

Update Operation: Suppose supplier S1 has moved from Qadian to Jalandhar. In that case we need to make changes in the record, so that the supplier table is up-to-date. Since supplier number is the primary key in the S (supplier) table, so there is only a single entry of S 1, which needs a single update and problem of data inconsistencies would not arise. Similarly, part and shipment information can be updated by a single modification in the tables P and SP respectively without the problem of inconsistency. Update operation in relational model is very simple and without any anomaly in case of relational model.

Delete Operation: Suppose if supplier S3 stops the supply of part P2, then we have to delete the shipment connecting part P2 and supplier S3 from shipment table SP. This information can be deleted from SP table without affecting the details of supplier of S3 in supplier table and part P2 information in part table. Similarly, we can delete the information of parts in P table and their shipments in SP table and we can delete the information suppliers in S table and their shipments in SP table.

Record Retrieval: Record retrieval methods for relational model are simple and symmetric which can be clarified with the following queries:

Advantages and Disadvantages of Relational Model

The major **advantages** of the relational model are:

Structural independence: In relational model, changes in the database structure do not affect the data access. When it is possible to make change to the database structure without affecting the [DBMS](#)'s capability to access data, we can say that structural independence has been achieved. So, relational database model has structural independence.

Conceptual simplicity: We have seen that both the hierarchical and the network database model were conceptually simple. But the relational database model is even simpler at the conceptual level. Since the relational data model frees the designer from the physical data storage details, the designers can concentrate on the logical view of the database.

Design, implementation, maintenance and usage ease: The relational database model achieves both data independence and structure independence making the database design, maintenance, administration and usage much easier than the other models.

Ad hoc query capability: The presence of very powerful, flexible and easy-to-use query capability is one of the main reasons for the immense popularity of the relational database model. The query language of the relational database models structured query language or SQL makes ad hoc queries a reality. SQL is a fourth generation language (4GL). A 4 GL allows the user to specify what must be done without specifying how it must be done. So, using SQL the users can specify what information they want and leave the details of how to get the information to the database.

Disadvantages of Relational Model

The relational model's disadvantages are very minor as compared to the advantages and their capabilities far outweigh the shortcomings. Also, the drawbacks of the relational database systems could be avoided if proper corrective measures are taken. The drawbacks are not because of the shortcomings in the database model, but the way it is being implemented.

Some of the disadvantages are:

Hardware overheads: Relational database system hides the implementation complexities and the physical data storage details from the users. For doing this, i.e. for making things easier for the users, the relational database systems need more powerful hardware [computers](#) and data [storage devices](#). So, the RDBMS needs powerful machines to run smoothly. But, as the processing power of modern computers is increasing at an exponential rate and in today's scenario, the need for more processing power is no longer a very big issue.

Ease of design can lead to bad design: The relational database is an easy to design and use. The users need not know the complex details of physical data storage. They need not know how the data is actually stored to access it. This ease of design and use can lead to the development and implementation of very poorly designed database management systems. Since the database is efficient, these design inefficiencies will not come to light when the database is designed and when there is only a small amount of data. As the database grows, the poorly designed databases will slow the system down and will result in performance degradation and data corruption.

'Information island' phenomenon: As we have said before, the relational database systems are easy to implement and use. This will create a situation where too many people or departments will create their own databases and applications.

These information islands will prevent the information integration that is essential for the smooth and efficient functioning of the organization. These individual databases will also create problems like data inconsistency, data duplication, data redundancy and so on.

But as we have said all these issues are minor when compared to the advantages and all these issues could be avoided if the organization has a properly designed database and has enforced good database standards.