ENTITY RELATIONSHIP MODEL

Entity relationship (ER) models are based on the real-world entities and their relationships. It is easy for the developers to understand the system by simply looking at the ER diagram. ER models are normally represented by ER-diagrams.

Components

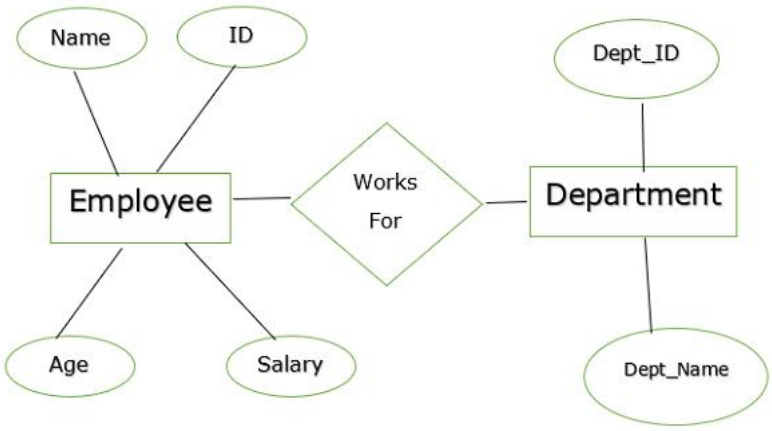
ER diagram basically having three components:

- **Entities** − It is a real-world thing which can be a person, place, or even a concept. For Example: Department, Admin, Courses, Teachers, Students, Building, etc are some of the entities of a School Management System.
- **Attributes** − An entity which contains a real-world property called an attribute. For Example: The entity employee has the property like employee id, salary, age, etc.
- **Relationship** − Relationship tells how two attributes are related. For Example: Employee works for a department.

An entity has a real-world property called attribute and these attributes are defined by a set of values called domain.

Example 1

Given below is another example of ER:



In the above example,

Entities − Employee and Department.

Attributes −

- Employee − Name, id, Age, Salary
- Department − Dept_id, Dept_name

The two entities are connected using the relationship. Here, each employee works for a department.

Features of ER

The features of ER Model are as follows −

- **Graphical Representation is Better Understanding** − It is easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
- **ER Diagram** − ER diagrams are used as a visual tool for representing the model.
- **Database Design** − This model helps the database designers to build the database.

Advantages

The advantages of ER are as follows −

- The ER model is easy to build.
- This model is widely used by database designers for communicating their ideas.
- This model can easily convert to any other model like network model, hierarchical model etc.
- It is integrated with the dominant relational model.

Disadvantages

The disadvantages of ER are as follows −

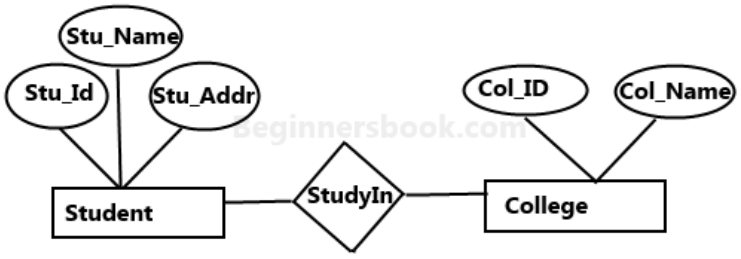
- There is no industry standard for developing an ER model.
- Information might be lost or hidden in the ER model.
- There is no Data Manipulation Language (DML).
- There is limited relationship representation.

Entity Relationship Diagram – ER Diagram in DBMS

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute



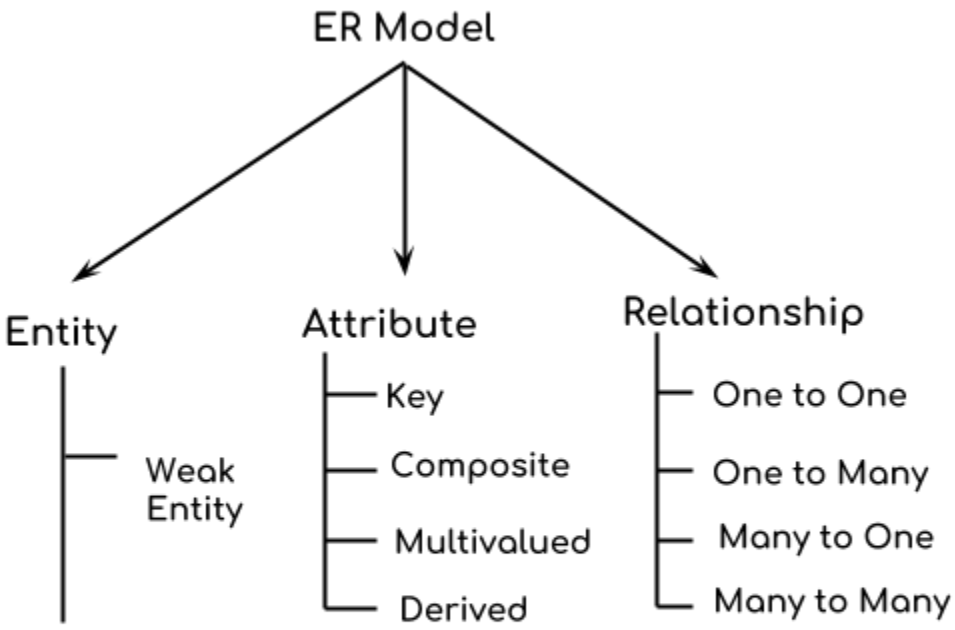
Sample E-R Diagram

of a table in database.

A simple ER Diagram:

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

Weak Entity:

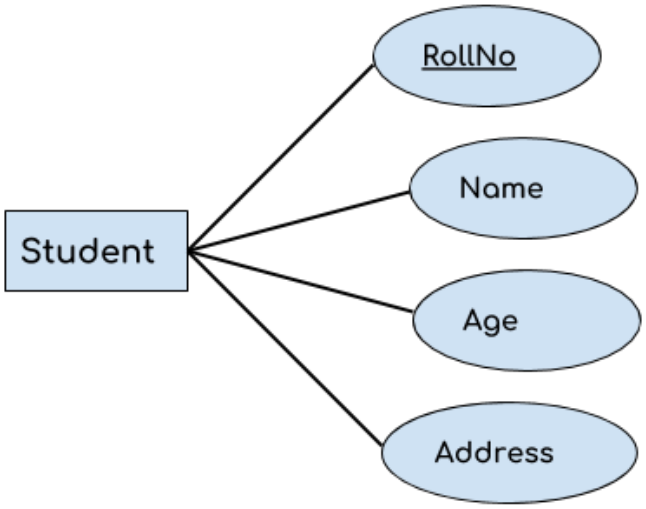
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

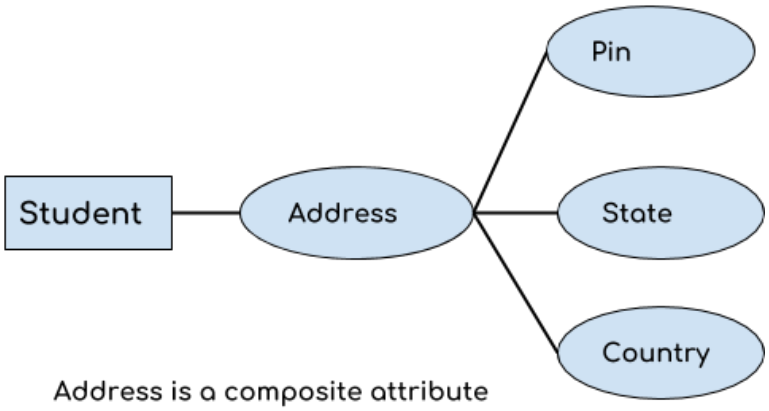
1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:



A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.

2. Composite attribute:



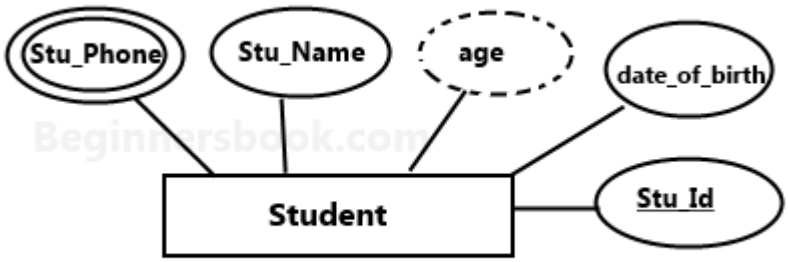
An attribute that is a combination of other attributes is known as composite attribute.

For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram.

For example – A person can have more than one phone numbers so the phone number attribute

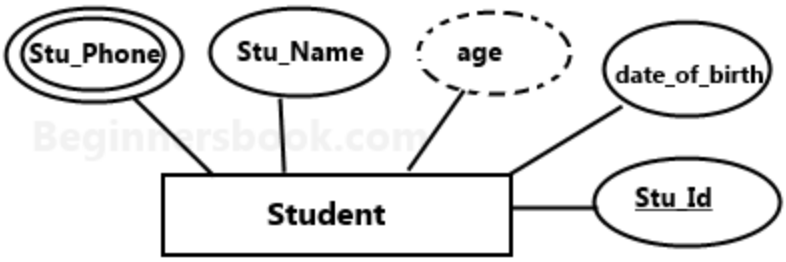


is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram.

For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.

What is Functional Dependency

Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other.

Example

The following is an example that would make it easier to understand functional dependency −

We have a **<Department>** table with two attributes − **DeptId** and **DeptName**.

DeptId = Department ID**DeptName** = Department Name

The **DeptId** is our primary key. Here, **DeptId** uniquely identifies the **DeptName** attribute. This is because if you want to know the department name, then at first you need to have the **DeptId**.

DeptId	**DeptName**
001	Finance
002	Marketing
003	HR

Therefore, the above functional dependency between **DeptId** and **DeptName** can be determined as **DeptId** is functionally dependent on **DeptName** −

DeptId -> DeptName

Types of Functional Dependency

Functional Dependency has three forms −

- Trivial Functional Dependency
- Non-Trivial Functional Dependency
- Completely Non-Trivial Functional Dependency

Let us begin with Trivial Functional Dependency −

Trivial Functional Dependency

It occurs when B is a subset of A in −

A ->B

Example

We are considering the same **<Department>** table with two attributes to understand the concept of trivial dependency.

The following is a trivial functional dependency since **DeptId** is a subset of **DeptId** and **DeptName**

{ DeptId, DeptName } -> Dept Id

Non –Trivial Functional Dependency

It occurs when B is not a subset of A in −

A ->B

Example

DeptId -> DeptName

The above is a non-trivial functional dependency since DeptName is a not a subset of DeptId.

Completely Non - Trivial Functional Dependency

It occurs when A intersection B is null in −

A ->B

NON LOSS DECOMPOSITIONFUNCTIONAL DEPENDENCIES:

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

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

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

The decomposition is lossless when it satisfies the following statement −

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

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

Here,

R = (A, B, C)

R1 = (A, B)

R2 = (B, C)

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

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

Draw a table of Relation R with Raw Data −

R (A, B, C)

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

It decomposes into the two sub relations −

R1 (A, B)

A	B
12	25
10	36
12	42

R2 (B, C)

B	C
25	34
36	09
42	30

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

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

$R_1 U R_2 = R$

We get the following result −

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

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

Normal Forms in DBMS

1. First Normal Form –

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- **Example –**

- ID Name Courses
- ----------
- 1 A c1, c2
- 2 E c3
- 3 M C2, c3

In the above table Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi-valued attribute

ID Name Course

---

1 A c1

1 A c2

2 E c3

3 M c2

3 M c3

2. Second Normal Form –

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1 –** Consider table-3 as following below.
- STUD_NO COURSE_NO COURSE_FEE
- 1 C1 1000
- 2 C2 1500
- 1 C4 2000
- 4 C3 1000
- 4 C1 1000
- 2 C5 2000

{Note that, there are many courses having the same course fee. }

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;
Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;
But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,
we need to split the table into two tables such as :
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE

Table 1 **Table 2**

STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500

1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000
2	C5		

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

- **Example 2 –** Consider following functional dependencies in relation R (A, B , C, D)
- AB -> C [A and B together determine C]

  BC -> D [B and C together determine D]

  In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form –

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.
A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 –** In relation STUDENT given in Table 4,
  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
  STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 –** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in above relation are {A, E, CD, BC} All attributes are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) –

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

- **Example 1 –** Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}
  Step 1. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
  Step 2. Prime attributes are those attributes that are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.
  Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.
  The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
  The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.
  So the highest normal form of relation will be 2nd Normal form.
- **Example 2 –**For example consider relation R(A, B, C)
  A -> BC,
  B ->
  A and B both are super keys so above relation is in BCNF.

Key Points –

3. BCNF is free from redundancy.
4. If a relation is in BCNF, then 3NF is also satisfied.
5. If all attributes of relation are prime attribute, then the relation is always in 3NF.
6. A relation in a Relational Database is always and at least in 1NF form.
7. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.

8. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
9. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
10. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).