ER model

* ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
* It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
* In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

**For example,** Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



Component of ER Diagram



1. Entity:

An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



**a. Weak Entity**

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



2. Attribute

The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

**For example,** id, age, contact number, name, etc. can be attributes of a student.



**a. Key Attribute**

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



**b. Composite Attribute**

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



**c. Multivalued Attribute**

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

**For example,** a student can have more than one phone number.



**d. Derived Attribute**

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

**For example,** A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

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



Types of relationship are as follows:

**a. One-to-One Relationship**

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

**For example,** A female can marry to one male, and a male can marry to one female.



**b. One-to-many relationship**

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

**For example,** Scientist can invent many inventions, but the invention is done by the only specific scientist.



**c. Many-to-one relationship**

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

**For example,** Student enrolls for only one course, but a course can have many students.



**d. Many-to-many relationship**

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

**For example,** Employee can assign by many projects and project can have many employees.



# Notation of ER diagram

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



**Fig: Notations of ER diagram**

Functional Dependency

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

1. X   →   Y

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

**For example:**

Assume we have an employee table with attributes: Emp\_Id, Emp\_Name, Emp\_Address.

Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp\_Id → Emp\_Name

We can say that Emp\_Name is functionally dependent on Emp\_Id.

Types of Functional dependency



1. Trivial functional dependency

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

**Example:**

1. Consider a table with two columns Employee\_Id and Employee\_Name.
2. {Employee\_id, Employee\_Name}   →    Employee\_Id is a trivial functional dependency as
3. Employee\_Id is a subset of {Employee\_Id, Employee\_Name}.
4. Also, Employee\_Id → Employee\_Id and Employee\_Name   →    Employee\_Name are trivial dependencies too.

2. Non-trivial functional dependency

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

**Example:**

1. ID   →    Name,
2. Name   →    DOB

First Normal Form (1NF)

* A relation will be 1NF if it contains an atomic value.
* It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385,9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389,8589830302 | Punjab |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

Second Normal Form (2NF)

* In the 2NF, relational must be in 1NF.
* In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

**TEACHER\_DETAIL table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **TEACHER\_AGE** |
| 25 | 30 |
| 47 | 35 |
| 83 | 38 |

**TEACHER\_SUBJECT table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |

Third Normal Form (3NF)

* A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
* 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
* If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

**Example:**

**EMPLOYEE\_DETAIL table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

* 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |

|  |  |  |
| --- | --- | --- |
| **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

**EMPLOYEE\_ZIP table:**

Boyce Codd normal form (BCNF)

* BCNF is the advance version of 3NF. It is stricter than 3NF.
* A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
* For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** | **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** |
| 264 | India |
| 264 | India |

**EMP\_DEPT table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP\_DEPT\_MAPPING table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_DEPT** |
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

**Functional dependencies:**

1. EMP\_ID   →    EMP\_COUNTRY
2. EMP\_DEPT   →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate keys:**

**For the first table:** EMP\_ID
**For the second table:** EMP\_DEPT
**For the third table:** {EMP\_ID, EMP\_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

# Dependency-Preserving Decomposition

The **dependency preservation decomposition** is another property of decomposed relational database schema D in which each functional dependency X -> Y specified in F either appeared directly in one of the relation schemas Ri in the decomposed D or could be inferred from the dependencies that appear in some Ri.

Decomposition D = { R1 , R2, R3,,.., ,Rm} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each Ri , in D is equivalent to F. In other words, R ⊂  join of R1, R1 over X. The dependencies are preserved because each dependency in F represents a constraint on the database. If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

### *Example:*

Let a relation R(A,B,C,D) and set a FDs F = { A -> B ,  A -> C  , C -> D}  are given.
A relation R is decomposed into -

R1 = (A, B, C) with FDs F1 = {A -> B, A -> C}, and

R2 = (C, D) with FDs F2 = {C -> D}.

 F' = F1 ∪ F2 = {A -> B, A -> C, C -> D}

 so, F' = F.

 And so, F'+ = F+.

Thus, the decomposition is dependency preserving decomposition.

## Lossless Decomposition

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example −

**<EmpInfo>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** | **Dept\_ID** | **Dept\_Name** |
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Decompose the above table into two tables:

**<EmpDetails>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** |
| E001 | Jacob | 29 | Alabama |
| E002 | Henry | 32 | Alabama |
| E003 | Tom | 22 | Texas |

**<DeptDetails>**

|  |  |  |
| --- | --- | --- |
| **Dept\_ID** | **Emp\_ID** | **Dept\_Name** |
| Dpt1 | E001 | Operations |
| Dpt2 | E002 | HR |
| Dpt3 | E003 | Finance |

Now, Natural Join is applied on the above two tables −

The result will be −

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** | **Dept\_ID** | **Dept\_Name** |
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Therefore, the above relation had lossless decomposition i.e. no loss of information.