Table of contents

- Entity
- Tuple
- CARDINALITY
    - ONE TO ONE
    - ONE TO MANY
    - MANY TO MANY
- Normalization
- Entity Relationship Diagram
- Roles of a Database Administrator

ENTITY

What is an entity?

Entity is a thing with distinct and independent existence.

What is an entity in DBMS?

An entity in DBMS (Database Management System) is a real-world object that has certain properties called attributes that define the nature of the entity. Entities are distinguishable, i.e., each entity in a pair of entities has a property that makes one entity different from the other entity.

For example: If we consider a car entity, it can have its attributes as a car's registration number, car's model, car's name, car's color, number of seats that are there inside the car, etc. Below is the tabular representation of the car entities.

	Registration Number	Model	Name	Colour	Number of Seats
Entity 1 →	DL123	LDI	Ritz	White	5
Entity 2 →	DL234	VDI	Swift	Black	5
Entity 3 →	DL345	ZXI	Dzire	Red	5

In this table, each row represents an entity. Among the attributes present in the above table, there are some attributes that can uniquely represent a row/entity in the given table. It means that all the elements in the table have a distinct value for this particular attribute. This attribute is called the primary attribute and primary key. Here the attribute registration number can be taken as the primary key since the registration number can never be the same for two different cars.

TYPES OF ENTITIES

An entity can be divided into two categories. These two categories of an entity are tangible entities and non tangible entities.

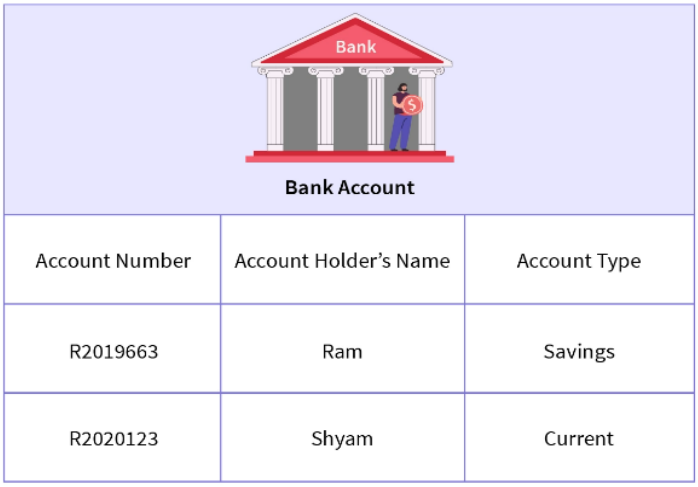
- **Tangible Entities -** These are the entities that physically exist in the real world. For example, the entity of cars, the entity of books, etc.
- **Non-Tangible Entities -** These are entities that do not physically exist in the real world. For example, an entity of email ids, an entity of social media accounts, etc.

ENTITY SET

An entity set in DBMS is a set that collectively represents a group of entities of a similar type. In entity-relationship diagrams, entity sets are represented by a rectangle, and the attributes of an entity set are represented by an ellipse.

For Example: An entity set of cars, an entity set of bank accounts, etc. In DBMS, the whole table in the tabular representation of data is an entity set, while each row inside this table is an entity.

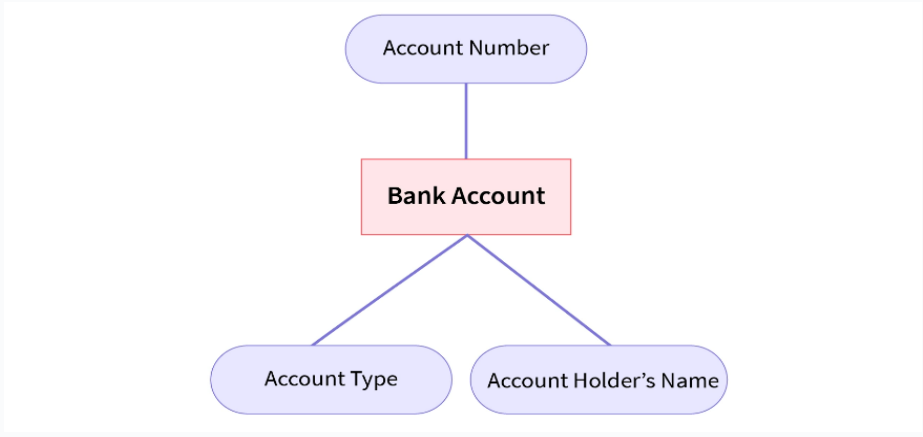
Below is the tabular representation, i.e., entity set of bank accounts.



Bank Account

Account Number	Account Holder's Name	Account Type
R2019663	Ram	Savings
R2020123	Shyam	Current

Below is the representation of the entity set above in an entity relationship diagram:



TYPES OF ENTITY SET

An entity set may be of two types:

- Strong Entity Set
- Weak Entity Set

Strong Entity Set

Strong entity set in DBMS is the entity set in which there exists a primary key, i.e There exists an attribute that can uniquely represent each element in the entity.

For Example: An entity set of cars having attributes such as a car's registration number, car's color, car's name is a strong entity since no two cars can have the same registration number. So, the attributes registration number can be considered as the primary key, and hence the entity set is a strong entity set.

In entity-relationship diagrams, strong entities are represented by a rectangle

Weak Entity Set

Weak entity set in DBMS is an entity set that does not have a primary key, i.e., there exists no attribute that can uniquely represent each entity in the entity set.

For Example: An entity set of cars having attributes such as a car's color, a car's name is a weak entity since two cars can have the same color as well as the same name. So, none of the attributes can be considered as the primary key and hence the entity set is a weak entity set.

In entity-relationship diagrams, weak entity sets are represented by a double rectangle, their relationship by a double diamond, and their connection by a double line.

Difference between an Entity and an Entity Set

- An entity in DBMS is used to represent any real-world object, while an entity set in DBMS is a set or collection of entities of similar types.
- In relational models, i.e., the tabular representation to represent data, each row in a table is a separate entity while the whole table is an entity set.
- We cannot represent an entity in an entity-relationship diagram (ER diagram) while entity sets are represented using a rectangle in an entity-relationship diagram (ER diagram).
- An entity contains the actual data, while an entity set contains the blueprint of an entity.

KEYS

A key in DBMS is an attribute or a set of attributes that help to uniquely identify a tuple (or row) in a relation (or table). Keys are also used to establish relationships between the different tables and columns of a relational database. Individual values in a key are called key values.

For example, every unique identification number is used to identify candidates in an educational institute. These can also help find all the available details maintained on the server about the candidate, such as their address, passport number, or phone number are keys unique to each candidate.

TYPES OF KEYS

- Primary Key
- Candidate Key
- Super Key
- Foreign Key
- Composite Key
- Alternate Key
- Unique Key

PRIMARY KEY

Primary key is a column of a table or a set of columns that helps to identify every record present in that table uniquely. There can be only one primary Key in a table. Also, the primary Key cannot have the same values repeating for any row. Every value of the primary key must be different with no repetitions. Amid many details, a primary key is the most significant one to understand what are keys and what is primary key in DBMS.

The PRIMARY KEY (PK) constraint put on a column or set of columns will not allow them to have any null values or any duplicates. One table can have only one primary key constraint. Any value in the primary key cannot be changed by any foreign keys (explained below) which refer to it.

CANDIDATE KEY

Candidate keys are those attributes that uniquely identify rows of a table. The Primary Key of a table is selected from one of the candidate keys. So, candidate keys have the same properties as the primary keys explained above. There can be more than one candidate keys in a table.

Apart from the primary key, every other attribute is considered the candidate keys.

SUPER KEY

Super Key is the set of all the keys which help to identify rows in a table uniquely. This means that all those columns of a table that are capable of identifying the other columns of that table uniquely will all be considered super keys. Super key is the superset of the candidate keys.

FOREIGN KEY

Foreign Key is used to establish relationships between two tables. A foreign key will require each value in a column or set of columns to match the Primary Key of the referential table. Foreign keys help to maintain data and referential integrity.

COMPOSITE KEY

A composite Key is a set of two or more attributes that help identify each tuple in a table uniquely. The attributes in the set may not be unique when considered separately. However, when taken all together, they will ensure uniqueness.

ALTERNATE KEY

As stated above, a table can have multiple choices for a primary key; however, it can choose only one. So, all the keys which did not become the primary Key are called alternate keys.

UNIQUE KEY

Unique Key is a column or set of columns that uniquely identify each record in a table. All values will have to be unique in this Key. A unique Key differs from a primary key because it can have only one null value, whereas a primary Key cannot have any null values.

TUPLE

A Tuple in DBMS is just a row that contains inter-related data about a particular entity. Tuple is also called "record" in DBMS. Tuples are mostly used in Relational Databases Management Systems (RDBMS).

A tuple is simply a row contained in a table in the tablespace. A table usually contains columns and rows in which rows stand for records while columns stand for attributes. A single row of a table that has a single record for such a relation is known as a tuple. A Tuple is, therefore, a single entry in a table; it is also called a row or record. They usually represent a set of related data, in Math it is simply an ordered list of elements.

Example Of Single Record Or Tuple

Consider the table given below. We have data of some students like their id, name, age, etc. here, each row has almost all the information of the respective student. Like the first row has all the information about a student named "Sophia", similarly, all other rows contain information about other students.

Table for reference:

ID	NAME	AGE	SUBJECT	MARKS
1	Sophia	21	Mathematics	100
2	Samuel	23	Physics	80
3	Promise	29	Chemistry	75
4	Somto	24	Biology	95

A Tuple from the above-given table

In the above-given image, you can see that a Tuple is just a row having attributes of a particular entity like name, age, marks, etc.

CARDINALITY

In database management, cardinality plays an important role. Here cardinality represents the number of times an entity of an entity set participates in a relationship set. Or we can say that the cardinality of a relationship is the number of tuples (rows) in a relationship. Types of cardinality in between tables are:

TYPES OF CARDINALITIES

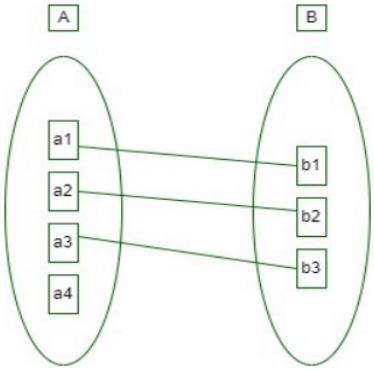
1. One-to-One
2. One-to-Many
3. Many-to-One
4. Many-to-Many

Mapping Cardinalities

In a database, the mapping cardinality or cardinality ratio means to denote the number of entities to which another entity can be linked through a certain relation set. Mapping cardinality is most useful in describing binary relation sets, although they can contribute to the description of relation sets containing more than two entity sets. Here, we will focus only on binary relation sets means we will find the relation between entity sets A and B for the set R. So we can map any one of following the cardinality:

1. ONE TO ONE

In this type of cardinality mapping, an entity in A is connected to at most one entity in B. Or we can say that a unit or item in B is connected to at most one unit or item in A.

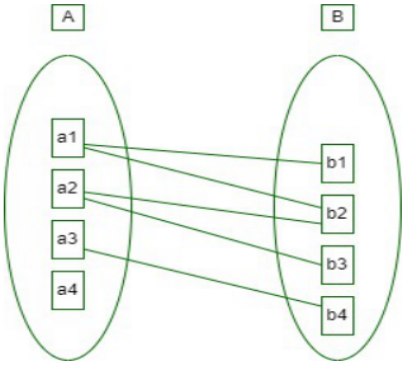


Example:

In a particular hospital, the surgeon department has one head of department. They both serve one-to-one relationships.

2. ONE TO MANY

In this type of cardinality mapping, an entity in A is associated with any number of entities in B. Or we can say that one unit or item in B can be connected to at most one unit or item in A.

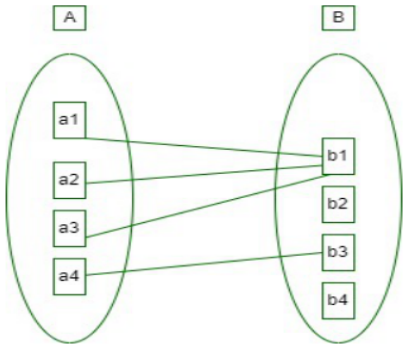


Example:

In a particular hospital, the surgeon department has multiple doctors. They serve one-to-many relationships.

3. MANY TO ONE

In this type of cardinality mapping, an entity in A is connected to at most one entity in B. Or we can say a unit or item in B can be associated with any number (zero or more) of entities or items in A.

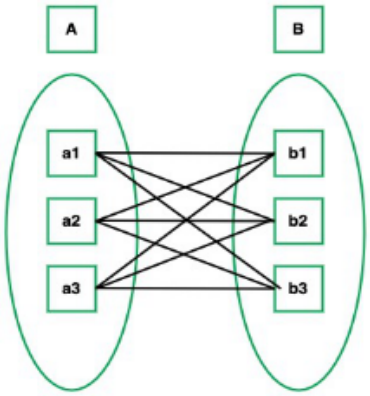


Example:

In a particular hospital, multiple surgeries are done by a single surgeon. Such a type of relationship is known as a many-to-one relationship.

4. MANY TO MANY

In this type of cardinality mapping, an entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A.



Example:
In a particular company, multiple people work on multiple projects. They serve many-to-many relationships.

NORMALIZATION

What is Normalization?

Normalization also known as data normalization is the process of reorganizing data in a database so that it meets two basic requirements:

1. There is no redundancy of data, all data is stored in only one place.
2. Data dependencies are logical,all related data items are stored together.

The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend the theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Why do we need Normalization?

Normalization is important for many reasons. The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure, but chiefly because it allows databases to take up as little disk space as possible, resulting in increased performance.

Normalization rules divide larger tables into smaller tables and link them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

What is a data modification anomaly?

A modification anomaly is an unexpected side effect from trying to insert, update, or delete a row.

Data modification anomalies can be categorized into three types;

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

NORMAL FORMS

Normalization works through a series of steps called normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Types of Normal Forms

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)

- 5NF (Fifth Normal Form)

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 attributes.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example Relation EMPLOYEE is not in 1NF because of multi valued attributes in 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, relations 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 does not contain any transitive partial dependency.
- 3NF is used to reduce data duplication. It is also used to achieve data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in the third normal form.

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

- X is a super key.
- 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:

> {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) are transitively dependent on the 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

EMPLOYEE_ZIP Table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

NOTE.

What is a transitive dependency?

The given functional dependency can only be transitive when it is formed indirectly by two FDs. For example, Thus, to achieve 3NF, one must eliminate the Transitive Dependency..

P -> R happens to be a transitive dependency when the following functional dependencies hold true:

- *P -> Q*
- *Q does not -> P*
- *Q -> R*

Example

<Show_Telecast>

Show_ID	Telecast_ID	Telecast_Type	CD_Cost ($)
F08	S09	Thriller	50
F03	S05	Romantic	30
F05	S09	Comedy	20

The table above is not in its 3NF because it includes a transitive functional dependency.

Show_ID -> Telecast_ID

Telecast_ID -> Telecast_Type

Thus, the following has a transitive type of functional dependency.

Show_ID -> Telecast_Type

The statement given above states the relation <Show_Telecast> violates the 3NF (3rd Normal Form). If we want to remove this violation, then we have to split the tables for the removal of the transitive functional dependency.

<Show>

Show_ID	Telecast_ID	CD_Cost ($)
F08	S09	50
F03	S05	30
F05	S09	20

<Telecast>

Telecast_ID	Telecast_Type
S09	Thriller
S05	Romantic
S09	Comedy

Now the above relation is in the Third Normal Form (3NF) of Normalization.

NOTE ENDS

BOYCE CODD NORMAL FORM (BCNF)

- BCNF is the advanced 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 a 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:

```
EMP_ID → EMP_COUNTRY
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:

> EMP_ID → EMP_COUNTRY
> 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 the left side part of both the functional dependencies is a key.

FOURTH NORMAL FORM (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multivalued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multivalued dependency.

Example:

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entities. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

FIFTH NORMAL FORM (5NF)

- A relation is in 5NF if it is in 4NF and does not contain any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

EXAMPLE:

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, a combination of all these fields is required to identify valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together act as a primary key, so we can't leave the other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

ENTITY-RELATIONSHIP DIAGRAM

Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important pieces of information: The major entities within the system scope, and the inter-relationships among these entities.

ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

When we talk about entities in ERD, very often we are referring to business objects such as people/roles (e.g. Student), tangible business objects (e.g. Product), intangible business objects (e.g. Log), etc. "Relationship" is about how these entities relate to each other within the system.

WHEN TO USE AN ER DIAGRAM:

While ER models are mostly developed for designing relational databases in terms of concept visualization and in terms of physical database design, there are still other situations when ER diagrams can help. Here are some typical use cases.

- **Database design** - Depending on the scale of change, it can be risky to alter a database structure directly in a DBMS. To avoid ruining the data in a production database, it is important to plan out the changes carefully. ERD is a tool that helps. By drawing ER diagrams to visualize database design ideas, you have a chance to identify the mistakes and design flaws, and to make corrections before executing the changes in the database.
- **Database debugging** - To debug database issues can be challenging, especially when the database contains many tables, which require writing complex SQL in getting the information you need. By visualizing a database schema with an ERD, you have a full picture of the entire database schema. You can easily locate entities, view their attributes and identify the relationships they have with others. All these allow you to analyze an existing database and to reveal database problems easier.
- **Database creation and patching** - Visual Paradigm, an ERD tool, supports a database generation tool that can automate the database creation and patching process by means of ER diagrams. So, with this ER Diagram tool, your ER design is no longer just a static diagram but a mirror that reflects truly the physical database structure.
- **Aid in requirements gathering -** Determine the requirements of an information system by drawing a conceptual ERD that depicts the high-level business objects of the system. Such an initial model can also be evolved into a physical database model that aids the creation of a relational database, or aids in the creation of process maps and data flow modes.

- **Education** - Databases are today's method of storing relational information for educational purposes and later retrieval, so ER Diagrams can be valuable in planning those data structures.
- **Research** - Since so much research focuses on structured data, ER diagrams can play a key role in setting up useful databases to analyze the data.

COMPONENTS AND FEATURES OF AN ER DIAGRAM

ER Diagrams are composed of entities, relationships and attributes. They also depict cardinality, which defines relationships in terms of numbers.

Entity

A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle. Read above to have a better understanding of entities or click here.

Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets. The three main cardinal relationships are one-to-one, one-to-many, and many-many. A one-to-one example would be one student associated with one mailing address. A one-to-many example (or many-to-one, depending on the relationship direction): One student registers for multiple courses, but all those courses have a single line back to that one student. Many-to-many example: Students as a group are associated with multiple faculty members, and faculty members in turn are associated with multiple students. Read above to have a better understanding of cardinality or click here.

Relationship

How entities act upon each other or are associated with each other. Think of relationships as verbs. For example, the named student might register for a course. The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the

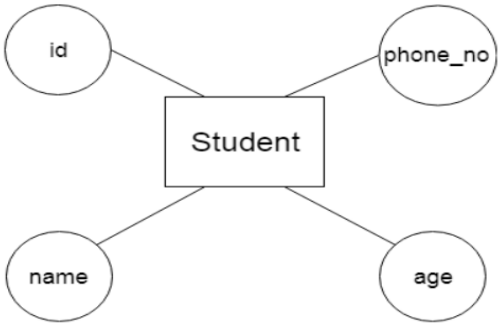
two entities in that way. Relationships are typically shown as diamonds or labels directly on the connecting lines.

Recursive relationship: The same entity participates more than once in the relationship.

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.

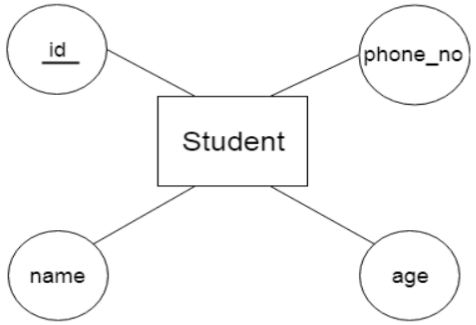


Descriptive attribute: A property or characteristic of a relationship (versus of an entity.)

Attribute categories: Attributes are categorized as simple, composite, derived, as well as single-value or multi-value. Simple: Means the attribute value is atomic and can't be further divided, such as a phone number. Composite: Sub-attributes spring from an attribute. Derived: Attributed is calculated or otherwise derived from another attribute, such as age from a birthdate.

1. 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.



2. Multivalued Attribute

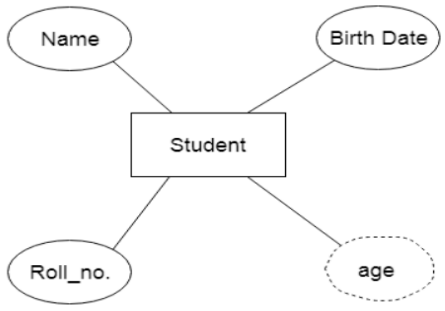
An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent a multivalued attribute.

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

3. Derived Attribute

An attribute that can be derived from another 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.



Mapping Natural Languages

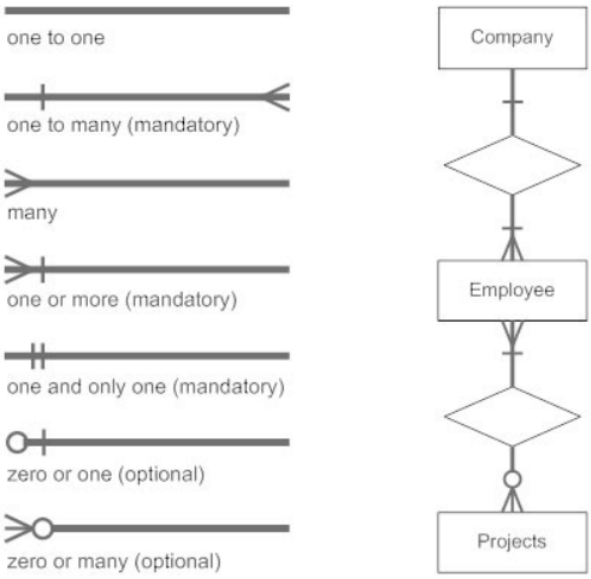
ER components can be equated to parts of speech, as Peter Chen did. This shows how an ER Diagram compares to a grammar diagram:

- Common noun: Entity type. Example: student.
- Proper noun: Entity. Example: Sally Smith.
- Verb: Relationship type. Example: Enrolls. (Such as in a course, which would be another entity type.)
- Adjective: Attribute for entity. Example: sophomore.
- Adverb: Attribute for relationship. Example: digitally.

The database query language ERROL(Entity Relationship Role Oriented) actually mimics natural language constructs. ERROL is based on reshaped relational algebra (RRA) and works with ER models, capturing their linguistic aspects.

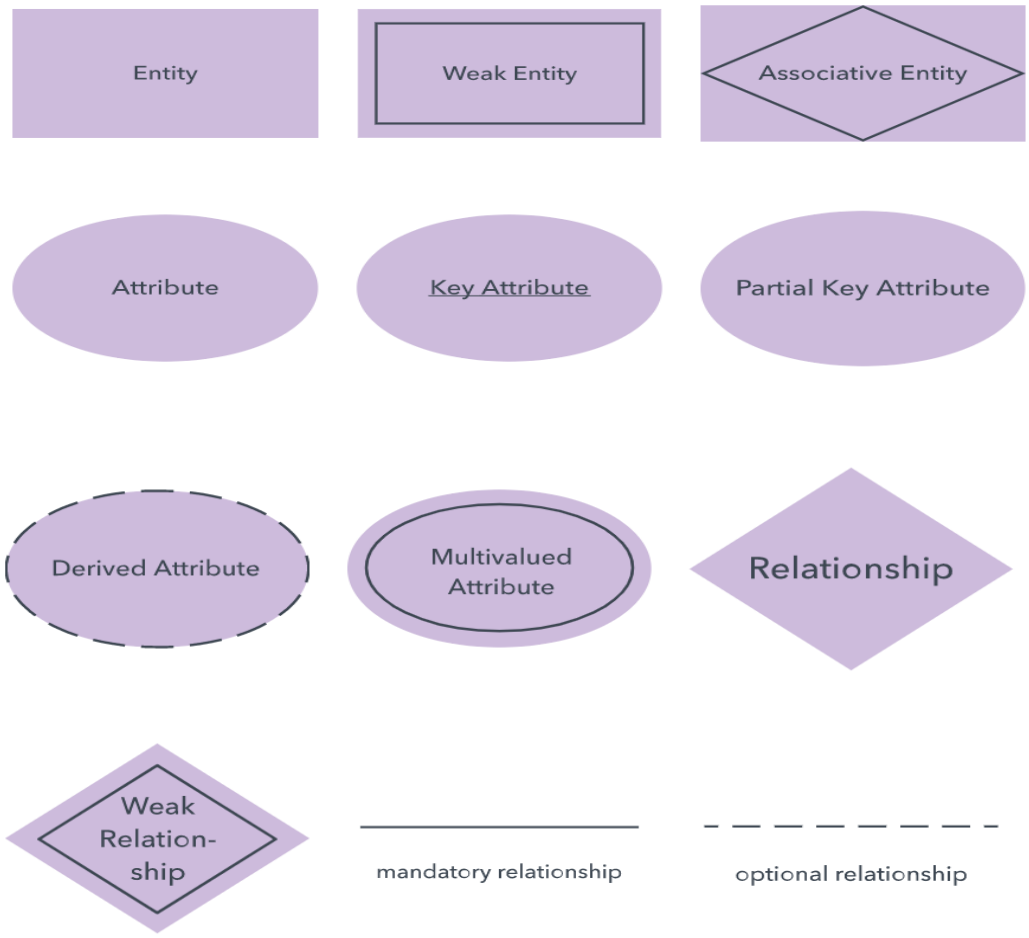
ERD SYMBOLS AND NOTATION

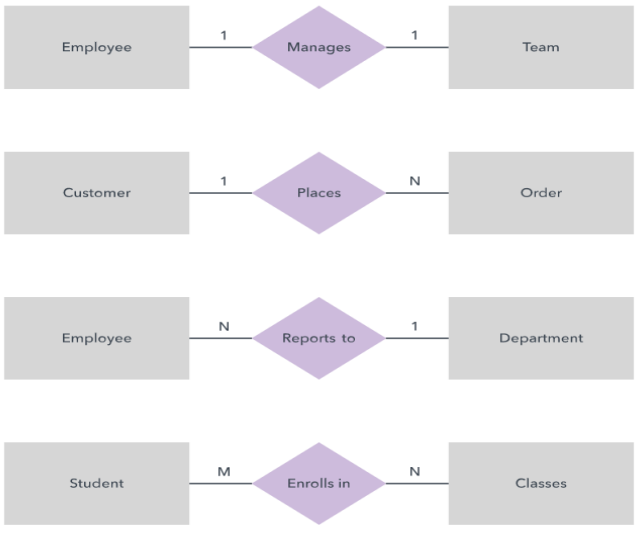
There are several notation systems, which are similar but vary in a few specifics. But the commonly used notation is:



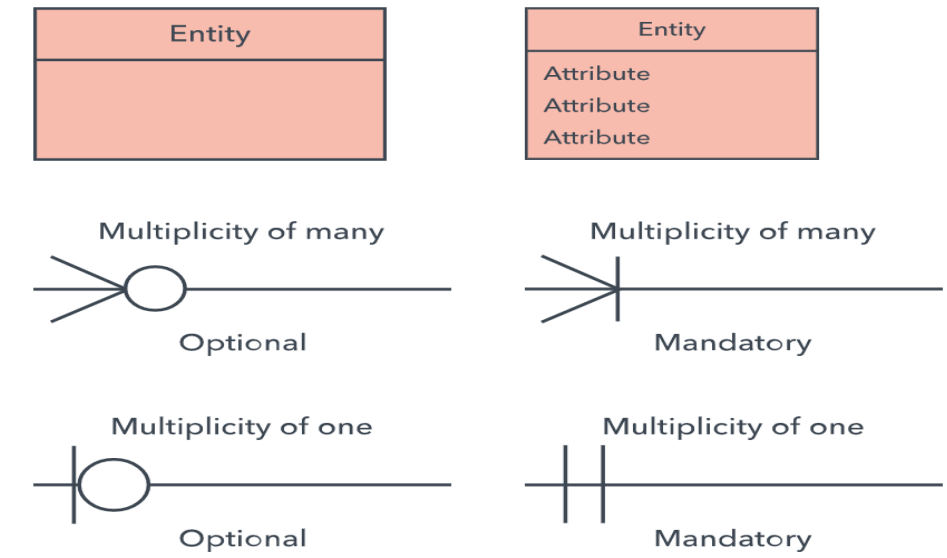
Other types of notation symbols are:

- **CHEN NOTATION STYLE**

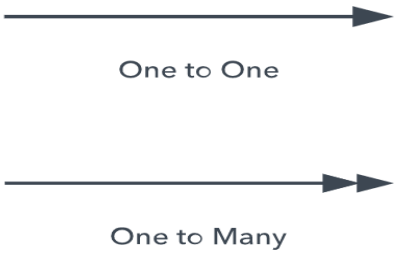




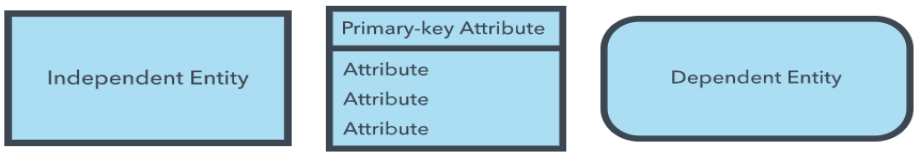
- **CROW'S FOOT/ MARTIN/ INFORMATION ENGINEERING STYLE**

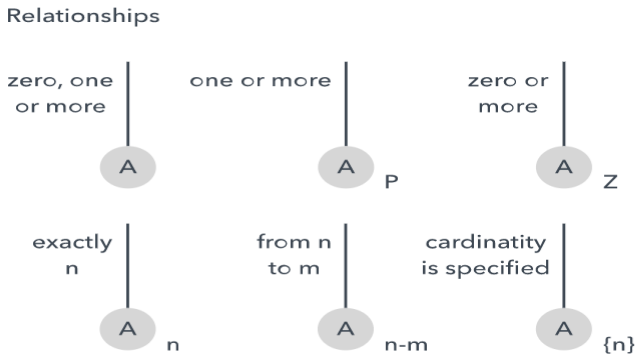


- **BACHMAN STYLE**

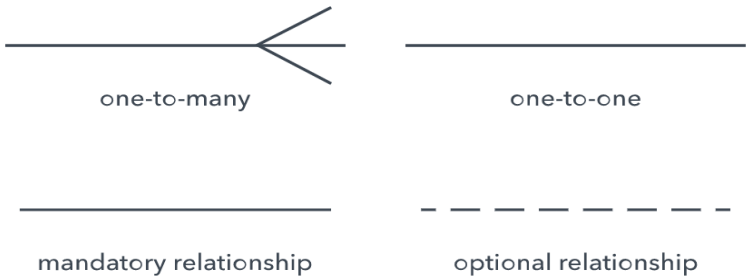


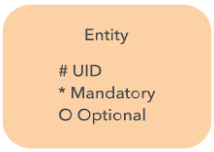
- **IDEF1X STYLE**

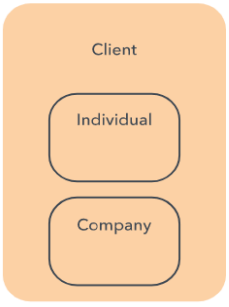




- **BARKER STYLE**







Subtypes

How to draw a basic ER diagram

- **Purpose and scope**: Define the purpose and scope of what you're analyzing or modeling.
- **Entities**: Identify the entities that are involved. When you're ready, start drawing them in rectangles (or your system's choice of shape) and labeling them as nouns.
- **Relationships**: Determine how the entities are all related. Draw lines between them to signify the relationships and label them. Some entities may not be related, and that's fine. In different notation systems, the relationship could be labeled in a diamond, another rectangle or directly on top of the connecting line.
- **Attributes**: Layer in more detail by adding key attributes of entities. Attributes are often shown as ovals.
- **Cardinality:** Show whether the relationship is 1-1, 1-many or many-to-many.

ROLES OF A DBA(DATABASE ADMINISTRATOR)

Database Administration consists of everything required to manage a database and make it available as needed. A database administrator is a person responsible for the design and management of an organization's databases as well as the evaluation, selection and implementation of the database management system (DBMS) software. He ensures the availability of the data produced and consumed by today's organizations via their IT systems.

ROLES OF A DBA

1. Deciding the hardware device

Depending upon the cost, performance and efficiency of the hardware, it is DBA who has the duty of deciding which hardware device will suit the company requirement. It is hardware that is an interface between end users and database so it needs to be of best quality.

2. Managing Data Integrity

Data integrity should be managed accurately because it protects the data from unauthorized use. DBA manages relationships between the data to maintain data consistency.

3. Decides Data Recovery and Back up method

If any company is having a big database, then it is likely that the database may fail at any instance. It is required that a DBA takes backup of the entire database in a regular time span. DBA has to decide how much data should be backed up and how frequently the back should be taken. Also the recovery of the database is done by DBA if they have lost the database.

4. Tuning Database Performance

Database performance plays an important role for any business. If a user is not able to fetch data speedily then it may lose company business. So by tuning and modifying sql commands a DBA can improve the performance of the database.

5. Capacity Issues

All the databases have their limits of storing data in it and the physical memory also has some limitations. DBA has to decide the limit and capacity of the database and all the issues related to it.

6. Database design

The logical design of the database is designed by the DBA. Also a DBA is responsible for physical design, external model design, and integrity control.

7. Database accessibility

DBA writes subschemas to decide the accessibility of the database. He decides the users of the database and also which data is to be used by which user. No user has the power to access the entire database without the permission of DBA.

8. Decides validation checks on data

DBA has to decide which data should be used and what kind of data is accurate for the company. So he always puts validation checks on data to make it more accurate and consistent.

9. Monitoring performance

If the database is working properly then it doesn't mean that there is no task for the DBA. Yes of course, he has to monitor the performance of the database. A DBA monitors the CPU and memory usage.

10. Decides content of the database

A database system has many kinds of content information in it. DBA decides fields, types of fields, and range of values of the content in the database system. One can say that DBA decides the structure of database files.

11. Provides help and support to user

If any user needs help at any time then it is the duty of DBA to help him. Complete support is given to the users who are new to the database by the DBA.

12. Database implementation

Database has to be implemented before anyone can start using it. So DBA implements the database system. DBA has to supervise the database loading at the time of its implementation.

13. Improve query processing performance

Queries made by the users should be performed speedily. As we have discussed, users need fast retrieval of answers so DBA improves query processing by improving their performance.

14. Data Move

A database administrator has the responsibility of moving a database set, say from a physical base to a cloud base, or from an existing application to a new application.

15. Database Upgrade

A database administrator has the responsibility of upgrading database software files when there is a new update for them, as this protects software from security breaches.

16. Error Log Review

A database administrator has the responsibility of interpreting the error messages sent by a database when there is a fault or bridge.

There are two common database administrator specialties:

- **System DBAs**

  System DBAs are responsible for the physical and technical aspects of a database, such as installing upgrades and patches to fix program bugs. They typically have a background in system architecture and ensure that the database in a firm's computer system works properly.

- **Application DBAs**

  Application DBAs support a database that has been designed for a specific application or a set of applications, such as customer service software. Using complex programming languages, they may write or debug programs and must be able to manage the aspects of the applications that work with the database. They also do all the tasks of a general DBA, but only for their particular application.