Entity Relationship Model

Proficieny	Dreadful

▼ How does a relational DBMS internally store its data?

They use B+ Trees. It is a special data structure allowing to efficiently store (access and update) a large sorted dictionary on a block storage device.

Database is a set of tables, and table is a set of indexes. And each index is, essentially a B+ tree storing its data.

We have a primary key and if not, we use a surrogate primary key (such as a specific date and time). We store its data as a dictionary sorted by its primary key in B+ tree structure.

▼ How DBMS query the data efficiently?

When you process a query in database, the query is converted into different commands and are executed sequentially. The output from one command is used as input into another.

Mostly, the operations don't store any entries, they act like UNIX pipe sequentially. This operation is called streaming operations. **Index range scan** and **Full Index Scan** are streaming operations.

Note - Sorting is not a streaming operation and store the result before output.

To fully clarify these concepts, let's see how above plan could look like, if our database engine would be written in Python:

> a = db.get_index('People', 'PK').all()
> b = (entry for entry in a if entry['State'] == 'CA')
> c = sort(b, 'Age')
> d = select(c, 'Name', 'State')return d

- a, b and d are streaming operations here: **Index.all()** extracts index range (full index), and two others are just Python sequences
- c is non-streaming operation, since it requires the whole sequence to be loaded and sorted before it produces the first entry of result.

So it's easy to estimate how much RAM a particular query needs to execute: it's roughly the amount of RAM needed to store all intermediate results of non-streaming operations in query plan. You can assume virtual RAM is used when intermediates don't fit in physical RAM. Actually it's more complex - there are on-disk temporary tables, etc., but it's nearly the same in terms of expected performance.

You may find this is a *suboptimal plan*: it implies DBMS must *read all entries* from People's primary index, and also, *store full **c** operation result*, since sort is non-streaming operation.

But the good thing is: DBMS passes this initial query plan to a *query optimizer*, which should transform it to a *nearly optimal plan*. In our case this plan should be this:

> a = entries from People's table IX_State secondary index with 'CA' key
> b = a joined with People's table primary index by Name column
> c = b sorted by Aged = select only Name and State columns from c
> result = d

Entity-Relationship Model

▼ What is entity relationship data model?

The entity-relationship (ER) data model is a high-level conceptual data model that has existed for over 35 years. It is well suited to data modeling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models, also called ER schemas, are represented by ER diagrams.

▼ What is an entity?

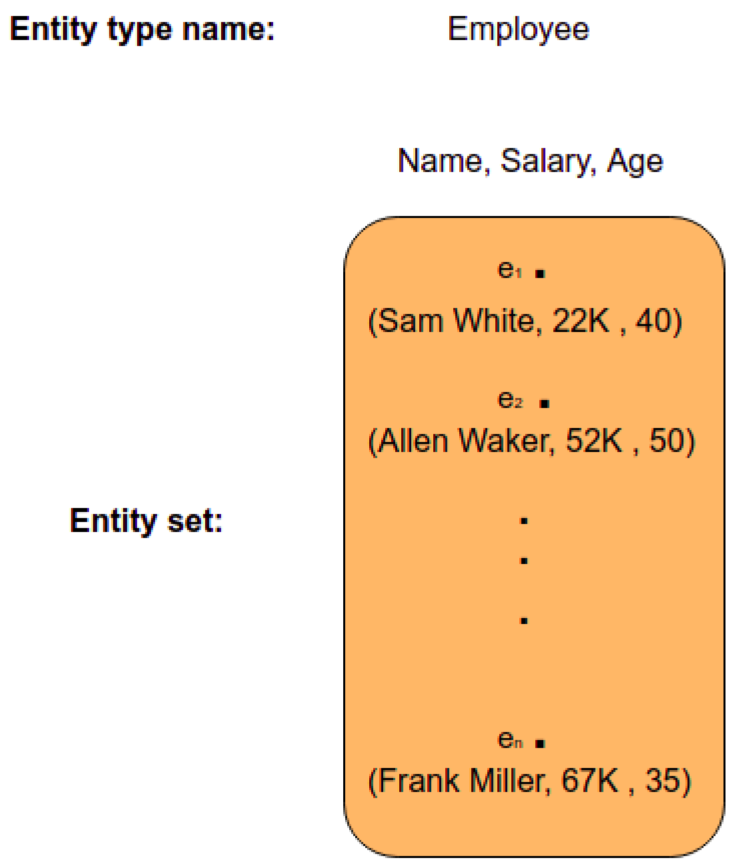
An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be:

- An object with physical existence (e.g., a lecturer, a student, a car).
- An object with conceptual existence (e.g., a course, a job, a position).

Each entity has attributes which are the particular properties that describe it.

▼ Entity types and entity sets

An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

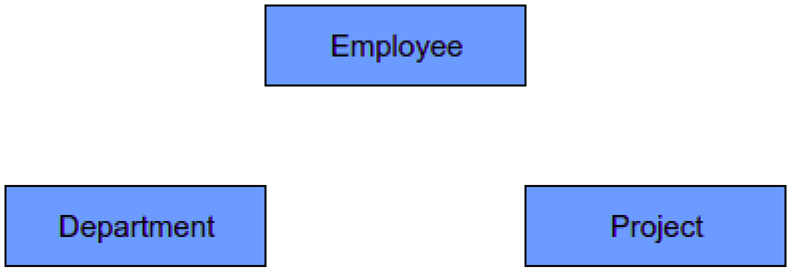


Example of entity type and entity set

The collection of all entities of a particular entity type in the database at any point in time is called an entity set. The entity set is usually referred to using the same name as the entity type, even though they are two separate concepts.

- ▼ Representation of ER Diagrams

  An entity type is represented in ER diagrams as a rectangular box enclosing the name of the entity type.



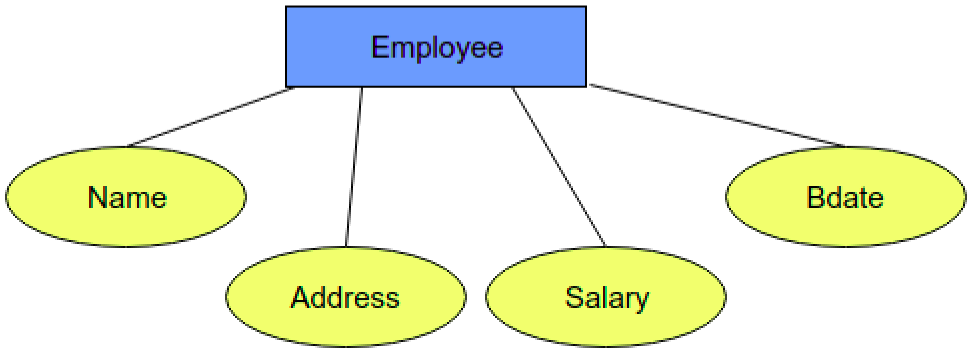
Representation of entities in ER model

- ▼ Attributes

  As stated in the previous lesson, each entity is described by a set of attributes. For example, the EMPLOYEE entity has attributes Name, Address, and Salary, etc.

  Each attribute has a name, is associated with an entity, and has a range of values it is allowed to take (an employee’s salary cannot be negative). However, information about the domain of an attribute is not presented in the ER diagram.

  - ▼ Representation

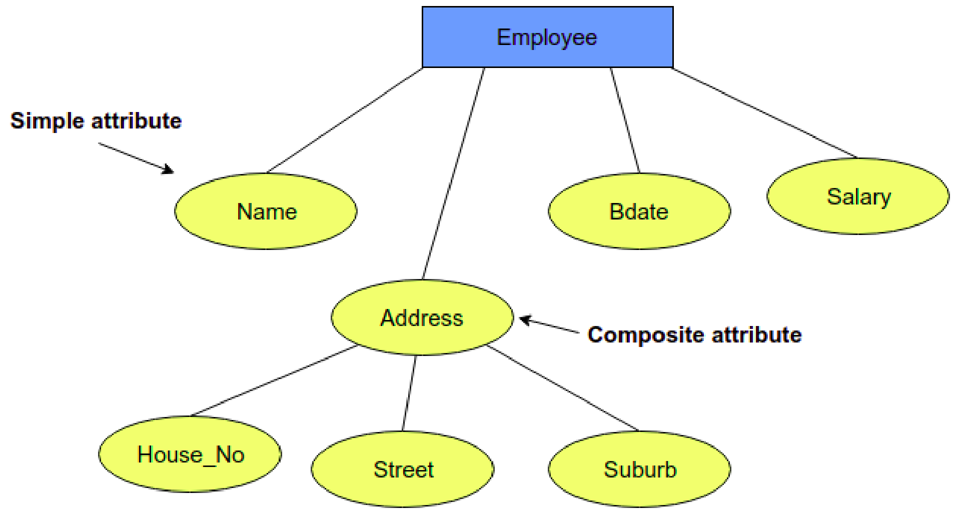


An EMPLOYEE entity represented along with its attributes.

- Types of Attributes
  - Simple Attributes

    Simple attributes are the atomic value, i.e., they cannot be further divided. They are also called single-value attributes. In the COMPANY database, an example of this would be: Name = 'John' and Age = 23.
  - Composite Attributes

    Composite attributes can be divided into smaller subpart, which represent more basic attributes with independent meanings.
    - Representation

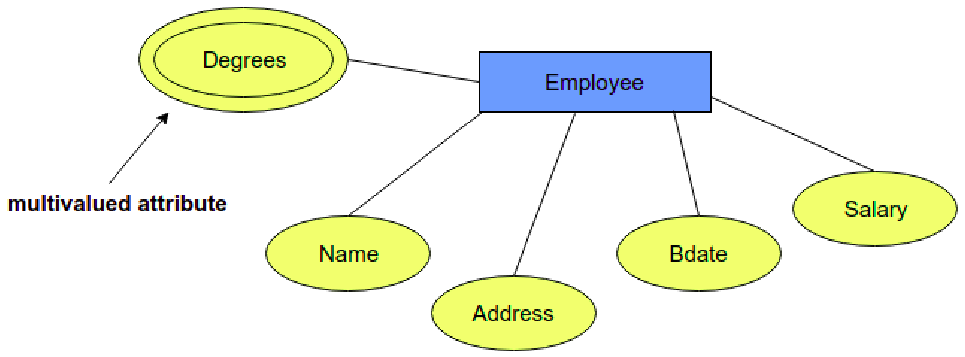


We can see simple and composite attributes highlighted in the diagram.

▼ Multivalued Attributes

Multivalued attributes have a set of values for each entity. An example of a multivalued attribute from the COMPANY database; one employee may not have any college degrees, another may have one, and a third person may have two or more degrees. Therefore, different people can have a different amount of values for the Degrees attribute.

▼ Representation

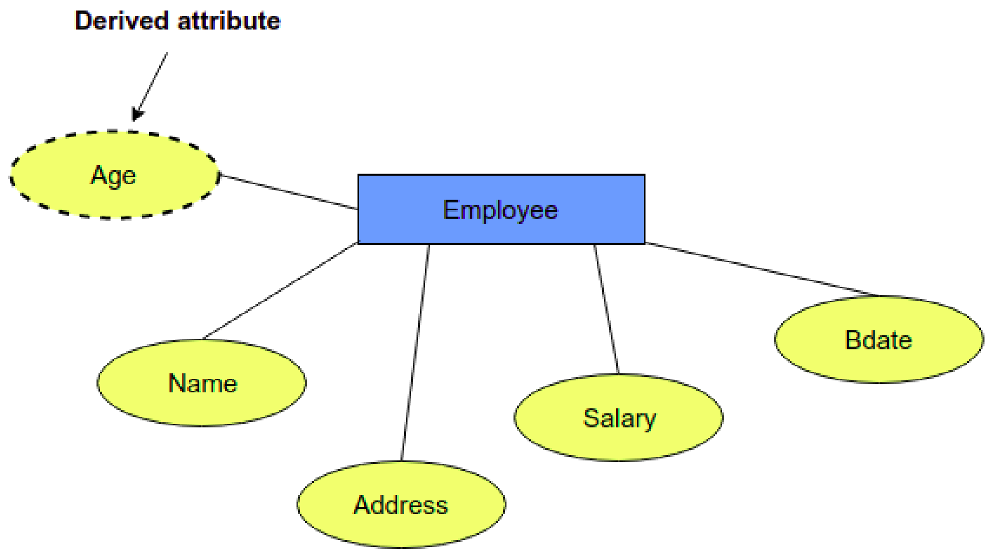


In ER diagrams multivalued attributes are represented with double outlines.

▼ Derived Attributes

Derived attributes are attributes that contain values calculated from other attributes. An example from the COMPANY database is that Age can be derived from the attribute Bdate. In this situation, Bdate is called a stored attribute, which is physically saved to the database.

▼ Representation

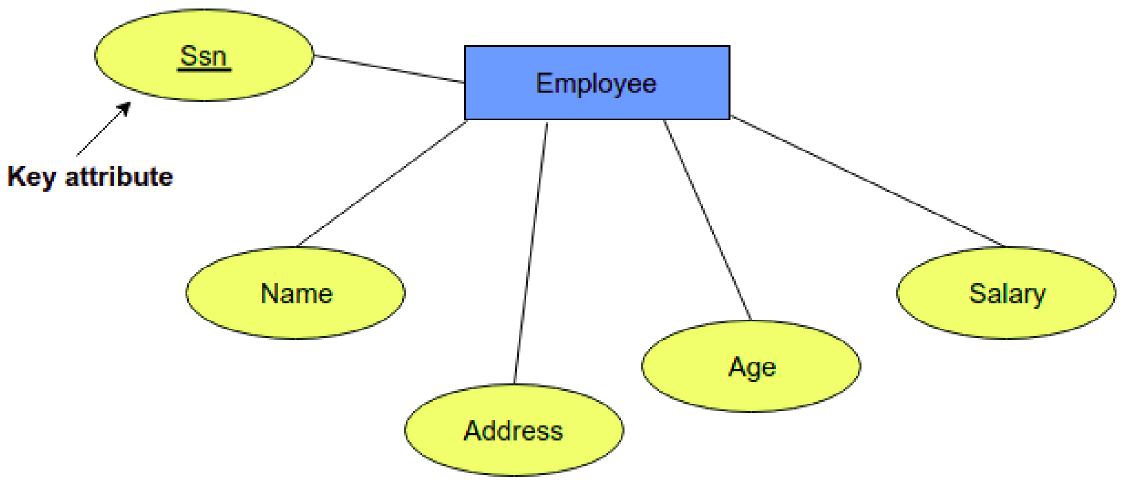


In ER diagrams multivalued attributes are represented with dotted outlines.

▼ Keys

An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. For the EMPLOYEE entity type, a typical key attribute is Ssn (Social Security number) as each person has a unique social security number.

▼ Representation



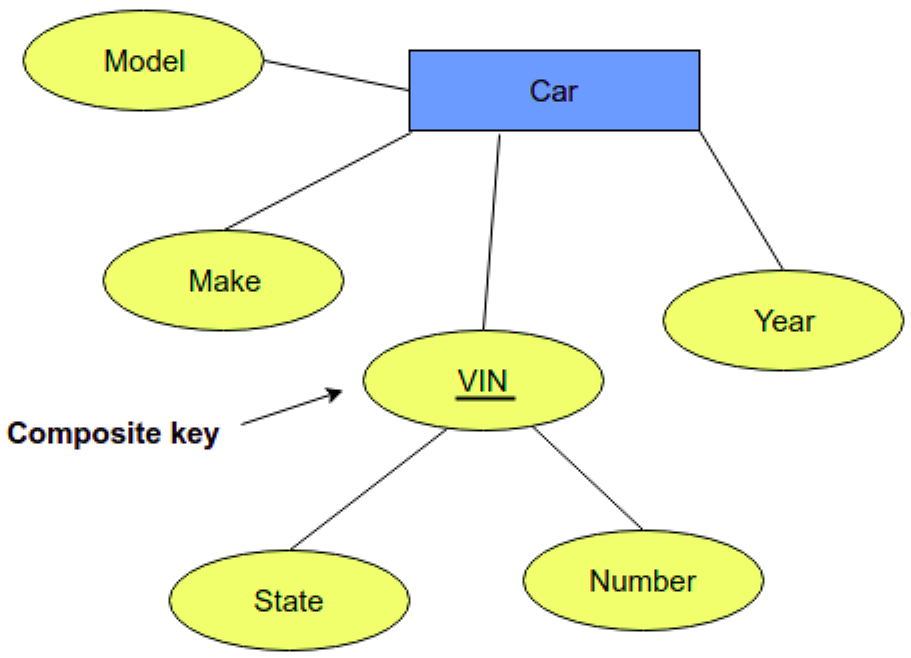
Ssn can be used to uniquely identify each employee so it is the key attribute.

- Primary Key

  It is a constraint that prohibits any two entities from having the same value for the key attribute at the same time. This unique attribute is also known as the primary key.

- Composite Key

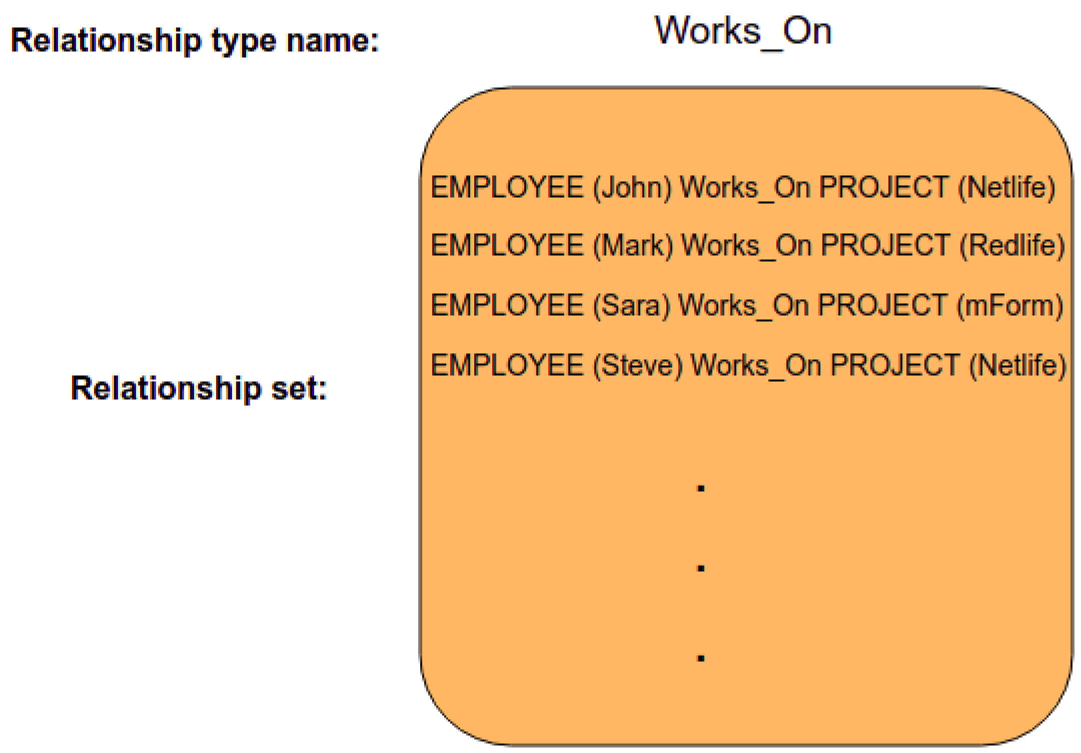
  Several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model that we describe here is to define a composite attribute and designate it as a key attribute of the entity type. This is called a composite key.



VIN is a composite key made up of State and Number

▼ Relationships

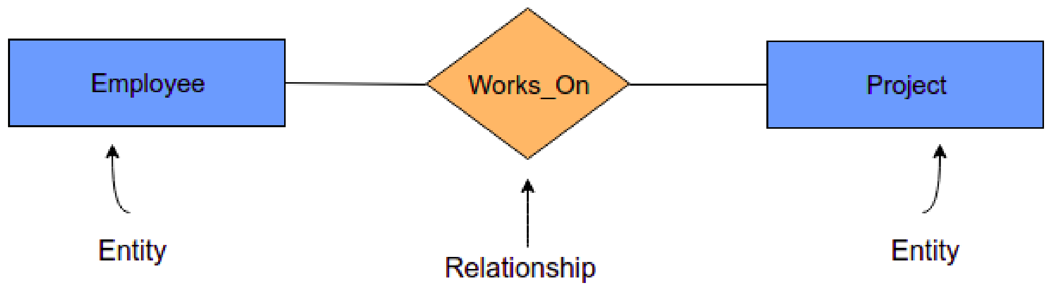
A relationship is an association between two entities, for example, an entity EMPLOYEE WORKS_ON PROJECT, which is another entity. However, note that different instances of employees will be working on different projects. So an employee, John WORKS_ON Project Netlife and maybe some other employee Sara (who is also an instance of an employee) WORKS_ON Project mForm.


Relationship type name:
Works_On
Relationship set:
EMPLOYEE (John) Works_On PROJECT (Netlife)
EMPLOYEE (Mark) Works_On PROJECT (Redlife)
EMPLOYEE (Sara) Works_On PROJECT (mForm)
EMPLOYEE (Steve) Works_On PROJECT (Netlife)
.
.
.

Example of relationship type and set

▼ Representation in ER Diagrams

In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes that represent the participating entity types.



Representation of relationships in ER model

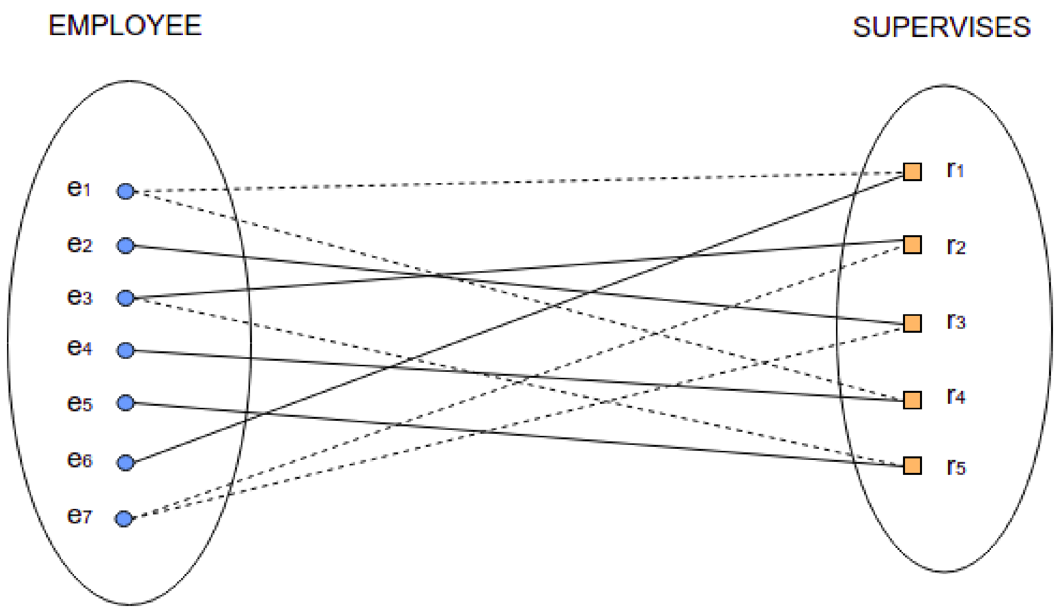
▼ Degree of Relationship Types

The degree of a relationship type is the number of participating entities types. We will focus on mainly three types of degrees:

▼ The Unary (recursive) relationship type

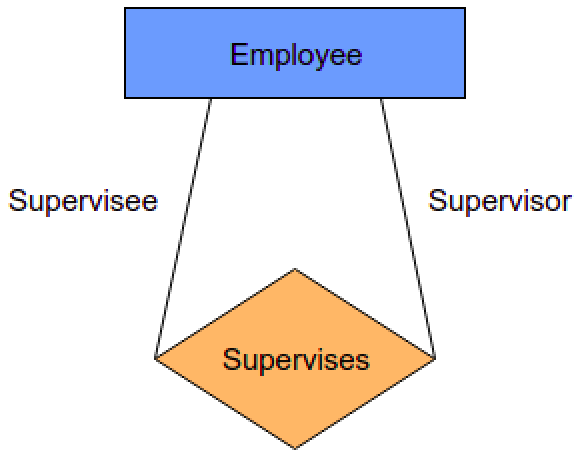
The unary relationship type involves only one entity type. However, the same entity type participates in the relationship type in different roles.

▼ Rep



Unary (Recursive) Relationship Type

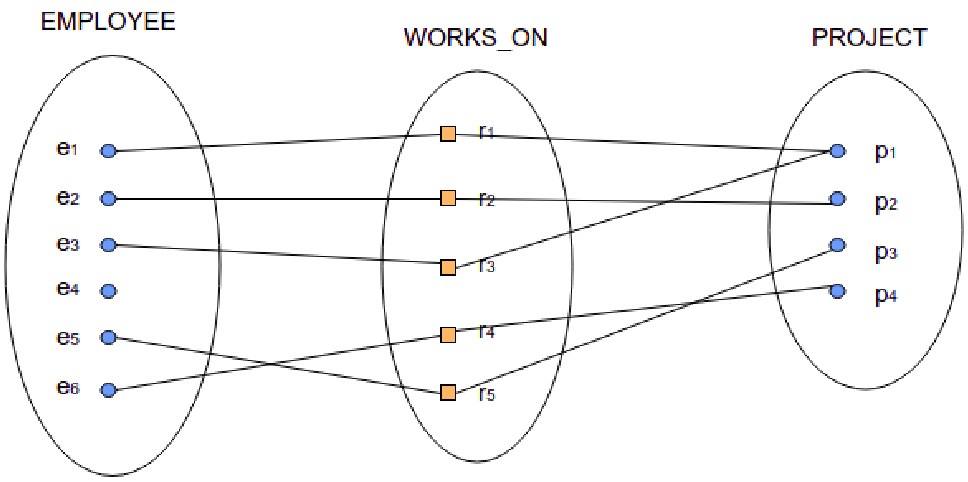
▼ ER



▼ The Binary Relationship Type

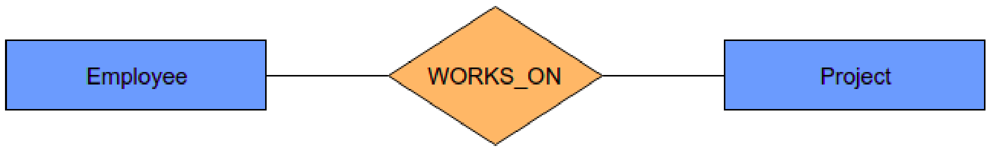
This relationship type has two entity types linked together. This is the most common relationship type. For example, consider a relationship type WORKS_ON between the two entity types EMPLOYEE and PROJECT, which associates each employee with the project he/she is working on.

▼ Rep



Binary Relationship Type

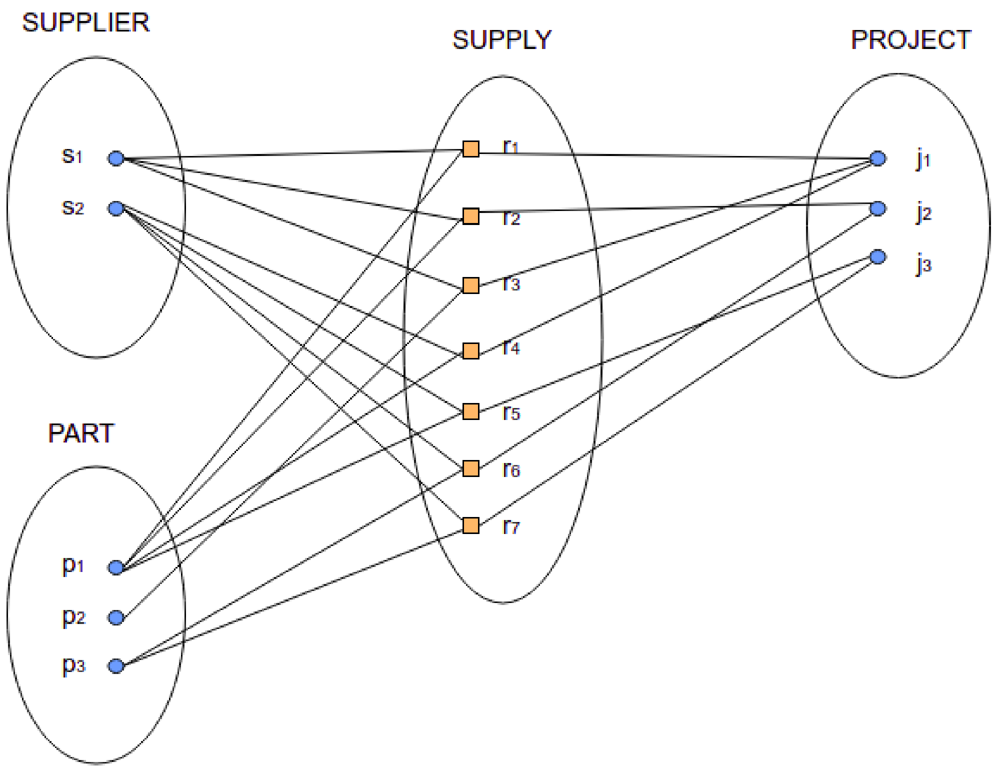
▼ ER Diagram



▼ The Ternary relationship type

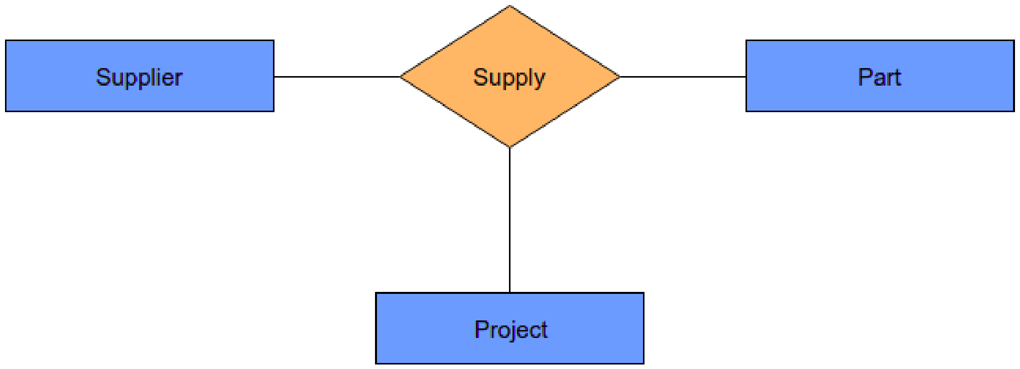
If there are three entity types linked together, the relationship is called a ternary relationship.

▼ Rep



Ternary Relationship Type

▼ ER Diagram



▼ Binary Relationship Type Constraints

In the previous lesson, we learned that there is a degree of relationship that exists between entities. However, sometimes this degree is affected by the constraints of the organization or a particular scenario. Consider, for example, a case where the company has a rule that each employee must work for exactly one department. In this and similar cases, we would like to describe this constraint in our schema. Such rules are usually called the “constraints” on the relationship types that exist in our schema.

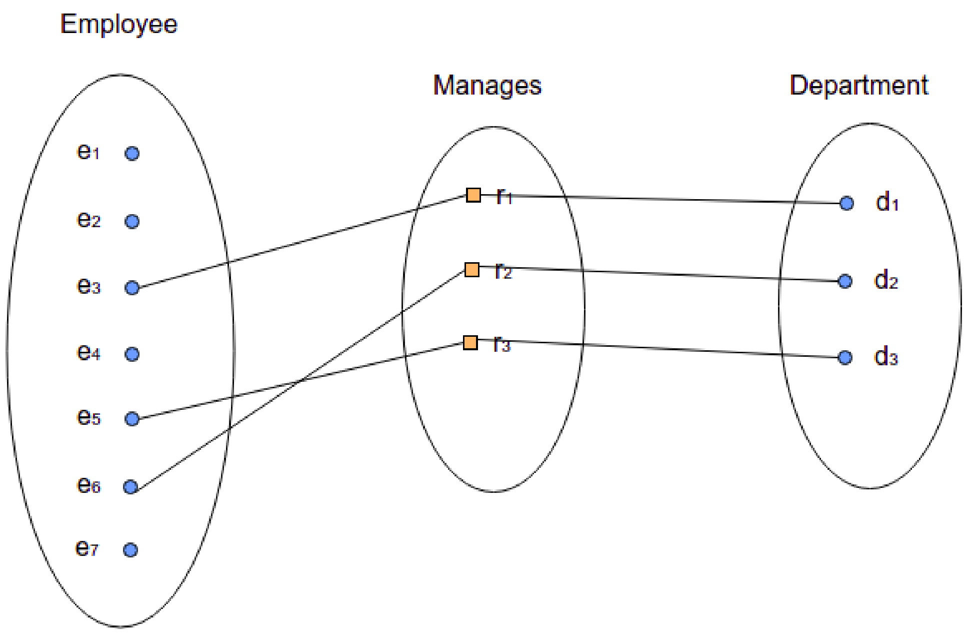
▼ Mapping Cardinality

Mapping cardinality describes the maximum number of entities that a given entity can be associated with via a relationship. In this lesson, we consider only the cardinality constraint for the binary relationship. The possible cardinality for binary relationship types are One to One (1:1), One to Many (1:N), and Many to Many (M:N).

▼ The one to one relationship

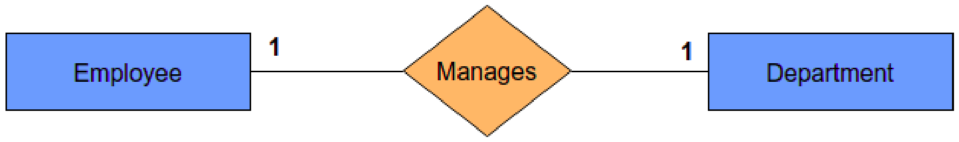
Given two entity sets A and B, there is a one to one relationship between A and B if each entity in set A is associated with at most one entity in set B and vice versa.

▼ Rep



An example of a 1:1 relationship

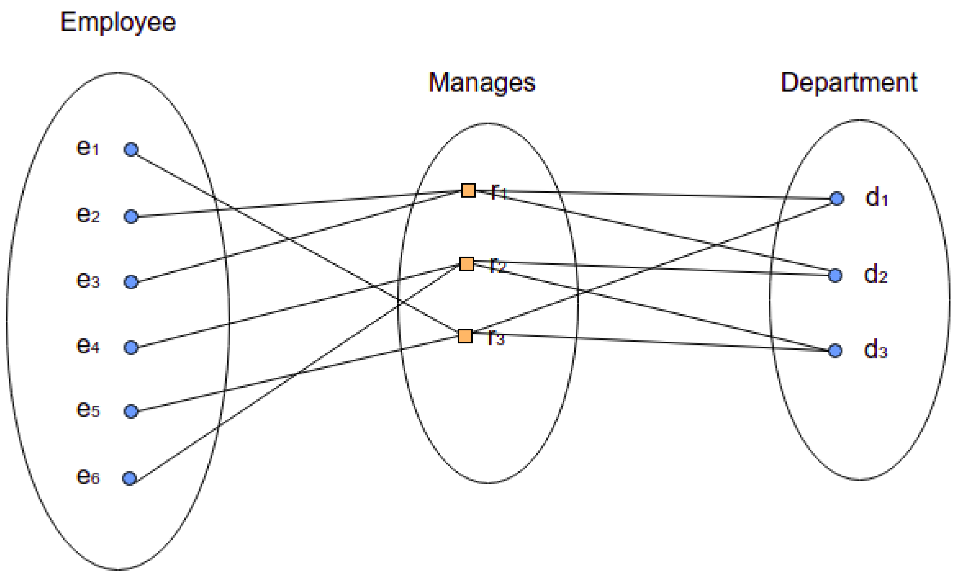
▼ ER Diagram



▼ The one to many relationship

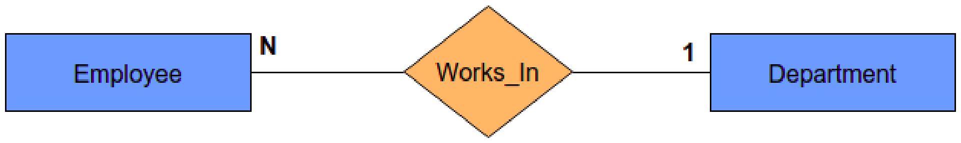
A one to many relationship set associates two entity sets A and B if each entity in A is associated with several entities in B however, each entity in B is associated with at most one entity in A. An example is that there are many employees working in a department, however, an employee can work for only one department.

▼ Rep



An example of a 1:N relationship

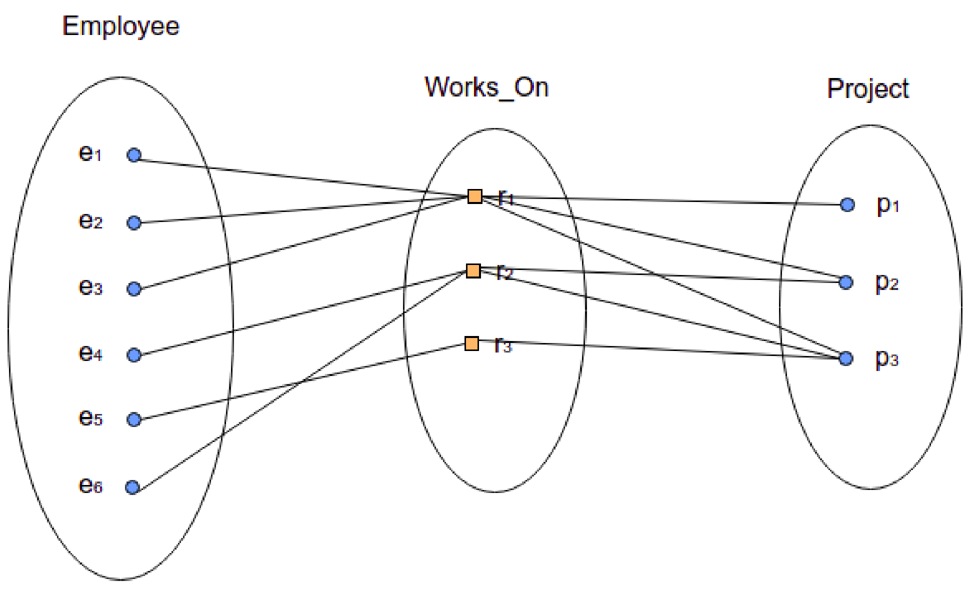
▼ ER Diagram



▼ The many to many relationship

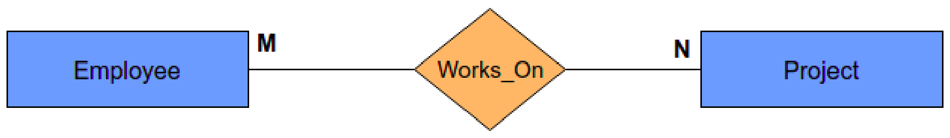
A many to many relationship set associates two entity sets A and B if each entity in A is associated with several entities in B, and, each entity in B is associated with several entities in A. The relationship type WORKS_ON is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.

▼ Rep



An example of a M:N relationship

▼ ER Diagram

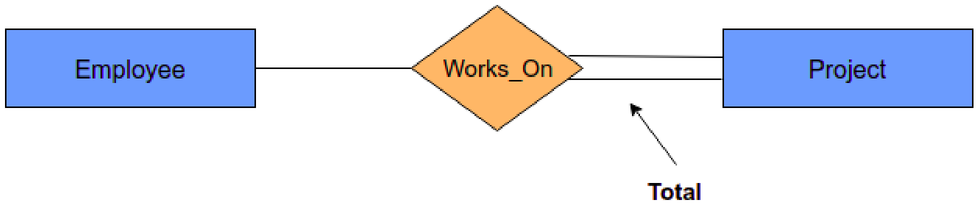


▼ Participation

The participation constraint specifies whether the existence of an entity depends on it being related to another entity via the relationship type.

▼ Total Participation

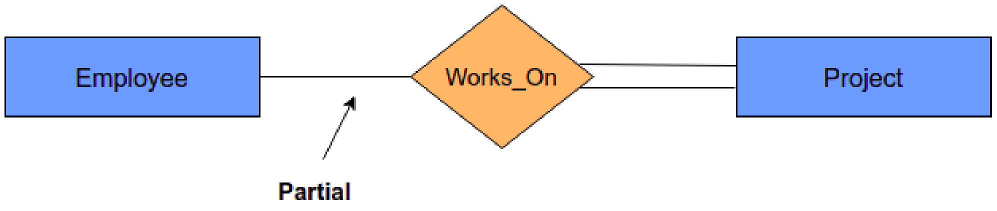
This specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why it is also known as mandatory participation. Total participation is represented using a double line between the entity set and relationship set.



The double line between the PROJECT entity and relationship WORKS_ON signifies total participation. It specifies that each project must be assigned to at least one employee. In other words, without an EMPLOYEE entity, the PROJECT entity would not exist.

- ▼ Partial Participation

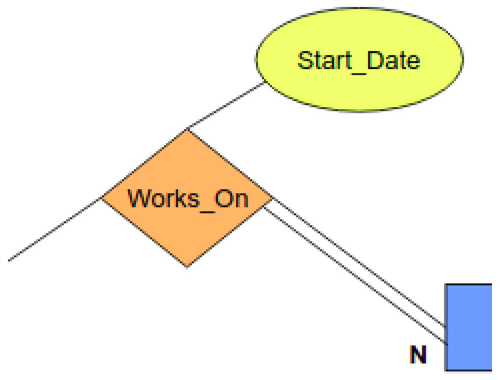
This specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why it is also known as optional participation. Partial participation is represented using a single line between an entity set and a relationship set.



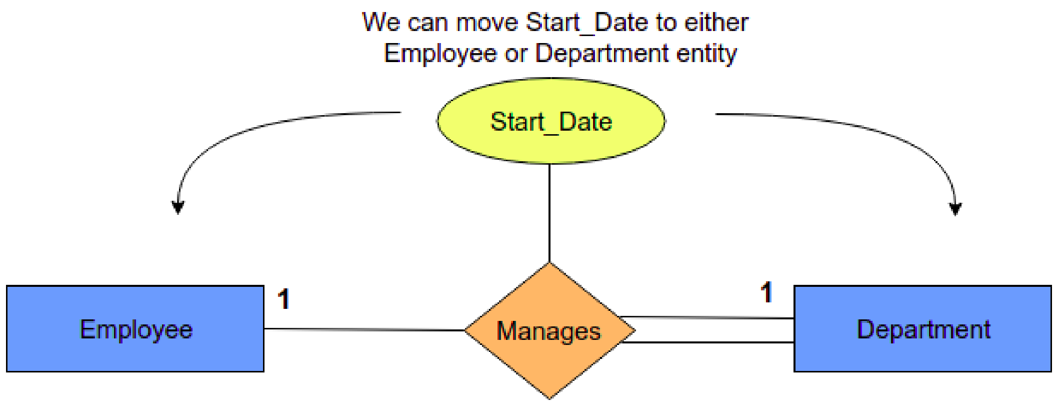
A single line between the entity EMPLOYEE and the relationship WORKS_ON signifies partial participation. It specifies that some employees may work on a project and some may not.

▼ Attributes of Relationship Types

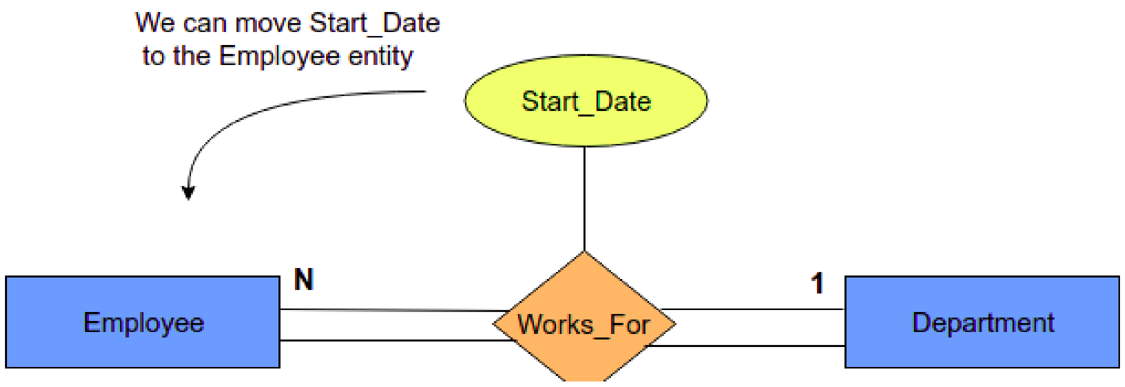
Relationship types can also have attributes, similar to those of entity types. Hence the relationship WORKS_ON between EMPLOYEE and PROJECT has an attribute `Start_Date`. This is illustrated in the diagram below:



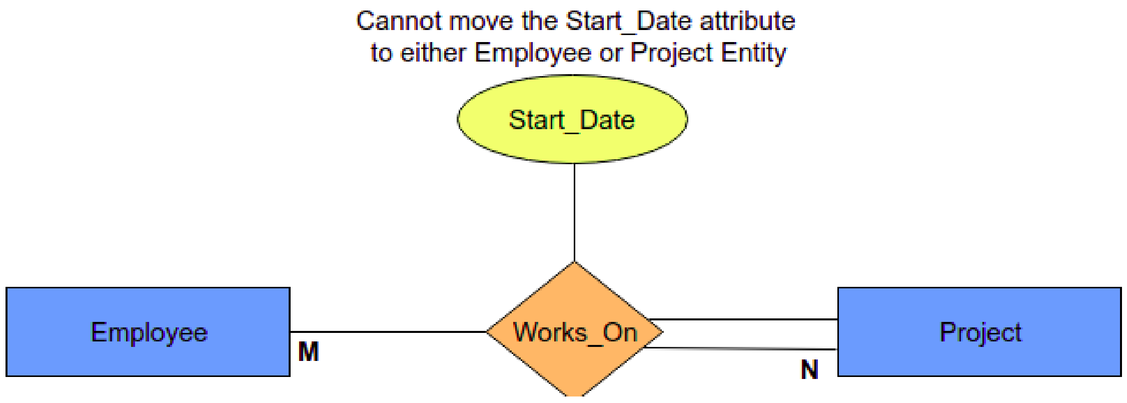
One - to - One:


We can move Start_Date to either Employee or Department entity
Start_Date
Employee
1
Manages
1
Department

One - to - many:



Many - to - Many:


Cannot move the Start_Date attribute to either Employee or Project Entity
Start_Date
Employee
Works_On
Project
M
N

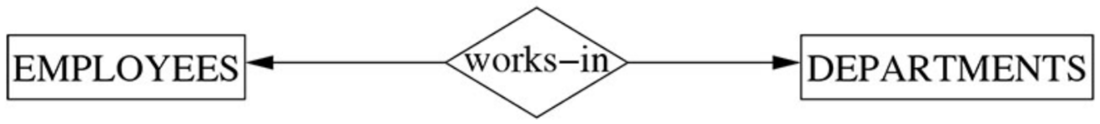
▼ Types of Relationship in ER Diagrams

`One to One Relationsip` - This exists when zero or one instance of entity A can be associated with zero or one instance of entity B, and vice versa. (1:1)

Example - Marriage of two people.

▼ Diagram

1) One To One



An employee can work in at most one department, and a department can have at most one employee.

A manager can only manage one department, and a department can have maximum of only one manager.

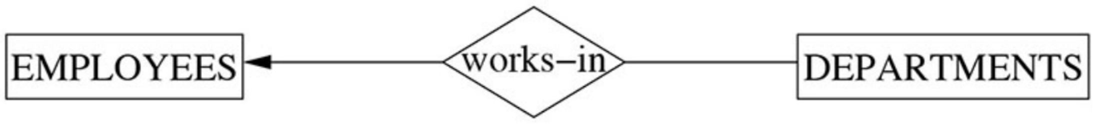
`One to Many Relationship` - For one instance of A, there can zero, one or more than one instance of B. For any instance of B, there can only be zero or one instance of A. (1:N)

Example - A child can only have one biological father. A father can have many biological children.

- Diagram

  The CEO can manage many departments, but all the departments will have only one CEO.

  2) One To Many

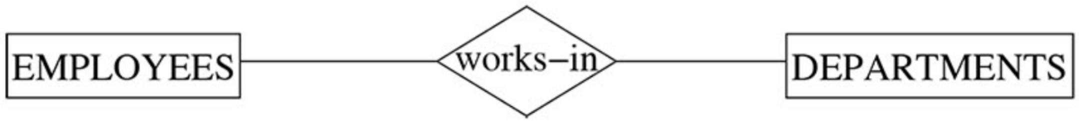
  An employee can work in many departments (>=0), but a department can have at most one employee.

`Many to Many Relationship` - For one instance of A, there exists zero, one or more than one instance of B. As well as, for one instance of B, there can be zero, one or more than one instance of A. (N:M)

Example - A student can enroll in multiple classes. The class also can have multiple students.

- Diagram

  4) Many To Many (default)

  An employee can work in many departments (>=0), and a department can have several employees

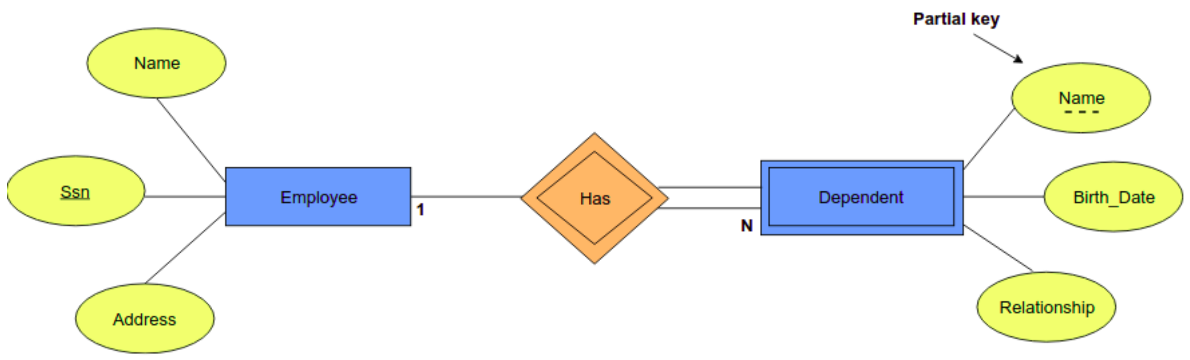
- Weak Entity Types

Entity types that do not have key attributes of their own are called weak entity types like the DEPENDENT entity type in the company database.

In contrast, regular entity types that do have a key attribute are called strong entity types.

Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or owner entity type, and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines.



Furthermore, a weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.

▼ 1. Design an ER Diagram -

The university database

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in. The database is a long way from one that would be suitable for a large tertiary institution, but it does illustrate relationships that are interesting to visualize.

Consider the following requirements list:

Entities:

- STUDENT entity.
- PROGRAM entity.
- COURSE entity.

Attributes:

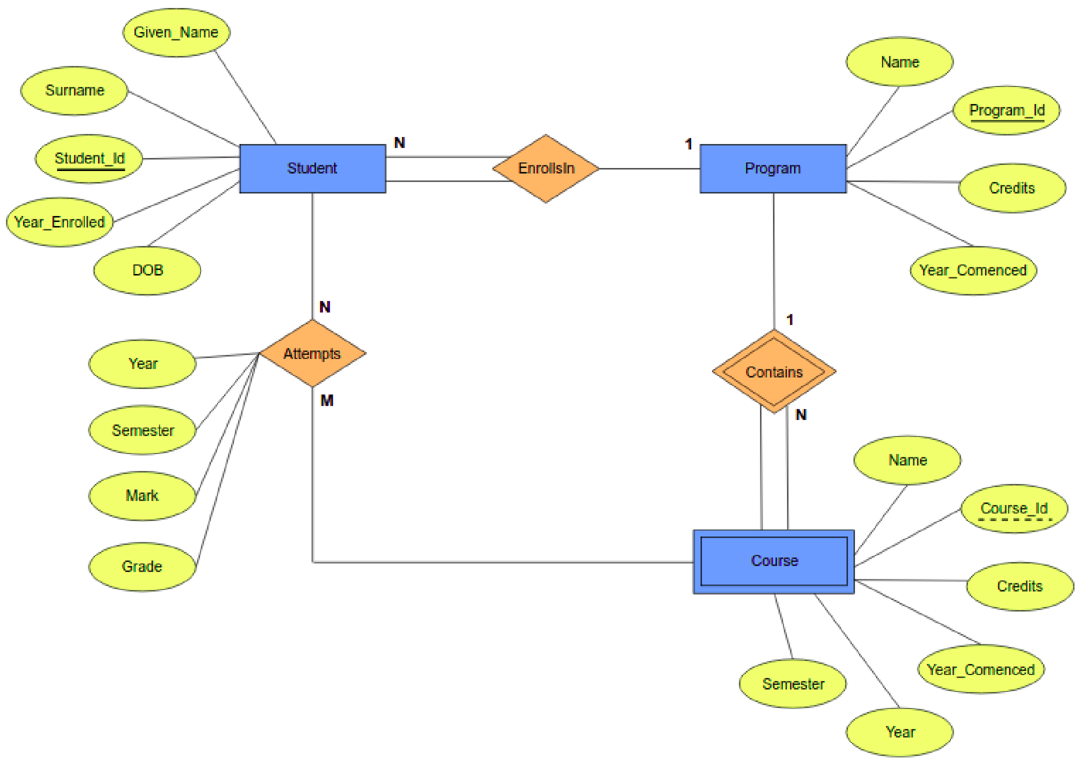
- Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, “John Paul”
- A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
- A course has a name, a course identifier, a credit point value, a year (for example, year 1), and a semester (for example, semester 1).

Relationships:

- The university offers one or more programs.
- A program is made up of one or more courses.
- A student must enroll in a program.
- A student takes the courses that are part of his/her program.
- When a student takes a course, the year and semester he/she attempted it are recorded. When he/she finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.

Try to design the ER diagram on your own.

▼ Solution



▼ Explanation

In our design:

- STUDENT is a strong entity, with an identifier, `Student_Id`, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).
- PROGRAM is a strong entity, with the identifier `Program_Id` as the primary key used to distinguish between programs.
- Each student must be enrolled in a program, so the STUDENT entity participates totally in the many-to-one ENROLLS_IN relationship with PROGRAM. A program can exist without having any enrolled students, so it participates partially in this relationship.
- A COURSE has meaning only in the context of a PROGRAM, so it's a weak entity, with `Course_Id` as a weak key. This means that a COURSE entity is uniquely identified using its `Course_Id` and the `Program_Id` of its owning program.

- As a weak entity, COURSE participates totally in the many-to-one identifying relationship with its owning PROGRAM.
- STUDENT and COURSE are related through the many-to-many, ATTEMPTS relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.
- When a student attempts a course, there are attributes needed to capture the `Year`, `Semester`, `Mark` and `Grade` of that course.

▼ 2. Design an ER Diagram -

The flight database

The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same.

Consider the following requirements list:

Entities:

- AIRPLANE entity
- FLIGHT entity
- PASSENGER entity
- BOOKING entity

Attributes:

- An airplane has a model number, a unique registration number, and the capacity to take one or more passengers.
- An airplane flight has a unique flight number, a departure airport, a destination airport, departure date and time, and arrival date and time.
- A passenger has a first name, surname, and unique email address.

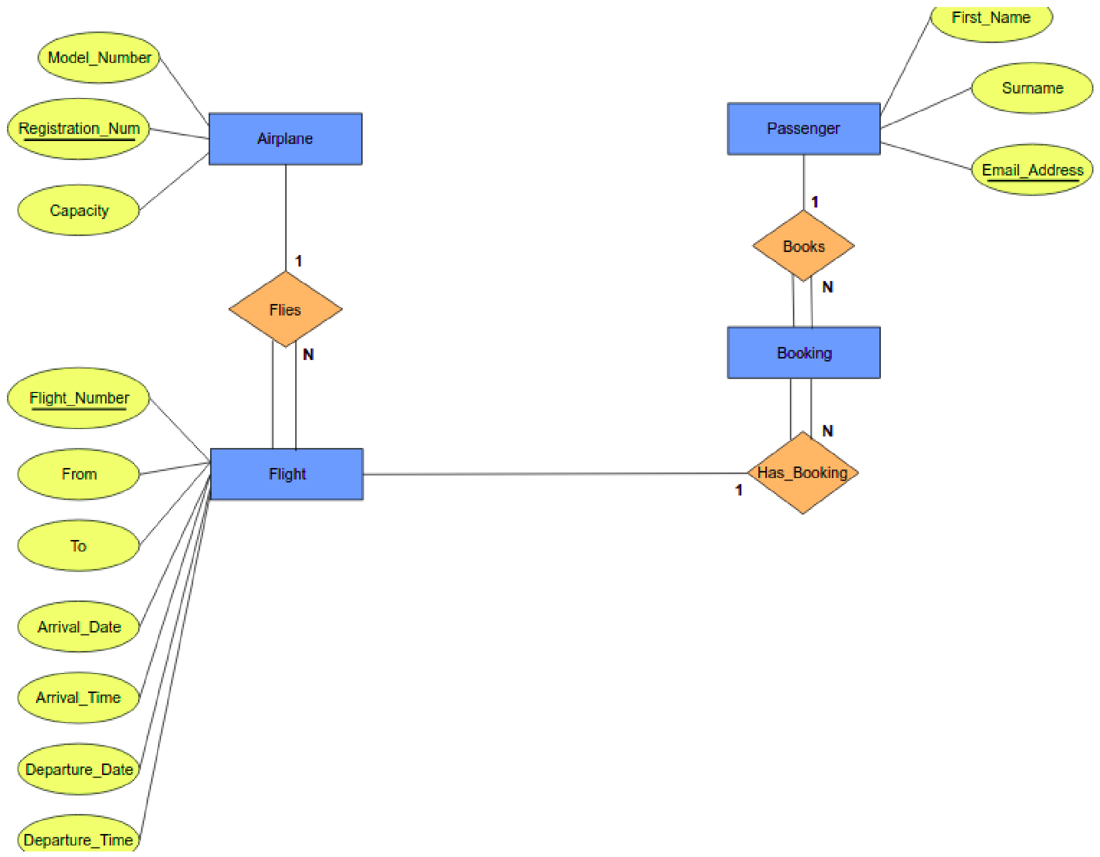
Relationships:

- Each flight is carried out by a single airplane.

- The airline has one or more airplanes.
- A passenger can book a seat on a flight.

Try to design the ER diagram on your own.

▼ Solution



▼ Explanation

In our design:

- An AIRPLANE is uniquely identified by its `Registration_Num`, so we use this as the primary key.
- A FLIGHT is uniquely identified by its `Flight_Number`, so we use the flight number as the primary key. The departure and destination airports are

captured in the `From` and `To` attributes, and we have separate attributes for the departure and arrival date and time.

- Because no two passengers will share an email address, we can use the `Email_Address` as the primary key for the PASSENGER entity.
- An airplane can be involved in any number of flights, while each flight uses exactly one airplane, so the FLIES relationship between the AIRPLANE and FLIGHT entities has cardinality 1:N; because a flight cannot exist without an airplane, the FLIGHT entity participates totally in this relationship.
- A passenger can book any number of flights, while a flight can be booked by any number of passengers. We capture this by creating the entity BOOKING which has 1:N relationships between it and the PASSENGER and FLIGHT entities.

Relational Data Model

▼ Relational Model Concepts

▼ Why use relational data model?

The relational data model was introduced by C.F. Codd in 1970. Currently, it is the most widely used data model.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint.
- Numerous database design methodologies.
- The standard database access language called Structured Query Language (SQL).
- Almost all modern commercial database management systems.

The relational data model describes the world as "a collection of inter-related relations (or tables)."

▼ What is relational model?

The relational model represents the database as a collection of relations. A **relation** is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship. The table and column names are helpful to interpret the meaning of values in each row.

Some popular relational database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

▼ Fundamental concepts of the relational data model

1. **Attribute**: Each column in a table. Attributes are the properties that define a relation, e.g., `Roll_No`, `Name`, etc.
2. **Tuple**: It is nothing but a single row of a table, which contains a single record. The above relation contains 4 tuples.
3. **Relation schema**: A relation schema represents the name of the relation with its attributes. e.g; STUDENT (`Roll_No`, `Name`, `Address`, `Phone`, and `Age`) is a relation schema for the STUDENT relation.
4. **Degree**: The number of attributes in the relation is known as the degree of the relation. The STUDENT relation defined above has a degree of 5.
5. **Cardinality**: The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has a cardinality of 4.
6. **Relation instance**: The set of tuples in a relation at a particular instance in time is called a relation instance. It changes whenever we insert, delete or update the database.
7. **NULL values**: The value which is not known or unavailable is called a NULL value. In the STUDENT relation, we see that the phone number of a STUDENT whose `Roll_No` is 4 is set to NULL.
8. **Domain**: A domain is the original set of atomic values used to model data. By **atomic value**, we mean that each value in the domain is indivisible as far as the relational model is concerned. In other words, a

domain is a set of acceptable values that a column is allowed to contain. For example:

- Name: The set of character strings that represent the name of a person.
- Age: Possible ages of students in a university; each must be an integer value between 17 and 27.

▼ Properties of a Table

1. Each row is unique

   This property ensures that no two rows in a relational table are identical; there is at least one column, or set of columns, whose values uniquely identify each row in the table. Such columns are called primary keys.

2. Values are atomic

   An atomic value is one that can not be broken down into smaller pieces. In other words, the table does not contain repeating groups or multivalued attributes.

3. Column values are of the same kind

   In relational terms, this means that all values in a column come from the same domain based on their data type including:

   - number (numeric, integer, float, smallint,...)
   - character (string)
   - date
   - logical (true or false)

   This property simplifies data access because developers and users can be certain of the type of data contained in a given column. It also simplifies data validation.

4. The sequence of columns is insignificant

   This property states that there is no specific sequence of columns, i.e., columns can be retrieved in any order and various sequences. The benefit of this property is that it enables many users to share the same relational table without concern of how the table is organized. It also permits the physical structure of the database to change without affecting the relations.

So, according to this property, even if we change the order of the columns, it will not have any impact on the workings of the relational model.

5. The sequence of rows is insignificant

   This property is analogous to the one above but applies to rows instead of columns. The main benefit is that the rows of a relational table can be retrieved in any order or sequence. Adding information to a relational table is simplified and does not affect existing queries.

   Similar to the above property, even if the rows are ordered differently, it will not affect the way the relational model works.

6. Each column has a unique name

   Because the sequence of columns is insignificant, columns must be referenced by name and not by position. In general, a column name need not be unique within an entire database but only within the relation to which it belongs.

▼ Database Keys

Keys are a very important part of the relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table. A key can be a single attribute or a group of attributes, where the combination may act as a key.

▼ Why do we need a key?

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

Also, tables store a lot of data. Tables generally extend to thousands of records, unsorted and unorganized.

Now, to fetch any particular record from such a dataset, you will have to apply some conditions.

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

- ▼ Super Key

  A super key is defined as a set of attributes within a table that can uniquely identify each record within a table.

- ▼ Candidate Key

  Candidate keys are defined as the minimal set of fields that can uniquely identify each record in a table. There can be more than one candidate key.

- ▼ Primary Key

  There can be more than one candidate key in a relation, out of which one can be chosen as the primary key.

- ▼ Composite Key

  A key that consists of two or more attributes that uniquely identify any record in a table is called a composite key. But the attributes which together form the composite key are not a key independently or individually.

- ▼ Alternate Key

  The candidate key other than the primary key is called an alternate key.

- ▼ Foreign Key

  A foreign key is a column or group of columns in a relational database table that provides a link between the data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

▼ Integrity Rules and Constraints

Constraints are a very important feature in a relational mode.

- allows a designer to specify the semantics of data in the database.
- rules that force DBMSs to check the data satisfies the semantics.

Relational Integrity Constraint - are the conditions that much be present for a valid reason. They are derived in the mini-world that the database represents.

`Domain Constraints` - specify that within each tuple, the value of each attribute must appear in the corresponding domain (it should belong to the appropriate data type.)

`Entity Integrity` - To ensure entity integrity, it is required that every relation has a primary key. Neither the primary key nor any part of it can contain NULL values.

`Referential Integrity Constraint` - is enforced when a foreign key references the primary key of the relation. It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be NULL.

> **Rule 1** - We cannot insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.
>
> **Rule 2** - We cannot delete or update the record of the referenced relation if the corresponding record exists in the referencing relation.

`Key Constraints` - The primary key must have unique values.