

## Chapter Three

# Relational Database Model

### Properties of Relational Databases - Basic Concepts in Relational Database

- Each row of a table is uniquely identified by a primary key (can be composed of one or more columns).
- Each tuple in a relation must be unique.
- Group of columns, that uniquely identifies a row in a table is called a candidate key.
- Entity integrity rule of the model states that no component of the primary key may contain a NULL value.
- A column or combination of columns that matches the primary key of another table is called a foreign key. This key is used to cross-reference tables.
- The referential integrity rule of the model states that, for every foreign key value in a table there must be a corresponding primary key value in another table in the database or it should be NULL.
- All tables are logical entities.
- A table is either a base tables (named relations) or views (unnamed relations).
- Only base tables are physically stores.
- Views are derived from base tables with SQL instructions like:  
    [select .. from .. where .. order by].
- Relatioal database is the collection of tables.
  - Each entity in one table.
  - Attributes are fields (columns) in table.
- Order of rows and columns is immaterial or irrelevant.
- Entries with repeating groups are said to be un-normalized.
- Entries are single-valued.
- Each column (field or attribute) has a distinct name.

All values in a column represent the same attribute and have the same data format.

## Building Blocks of the Relational Database Model

The building blocks of the relational database model are:

- **Entities:** Real world physical or logical object.
- **Attributes:** Properties used to describe each Entity or real world object.
- **Relationship:** The association between the real world objects (i.e Entities.)
- **Constraints:** Rules that should be obeyed or followed while manipulating the data.

**Entities:** The entities (persons, places, things etc.) which the organization has to deal with.

Relations can also describe relationships. The name given to an entity should always be a singular noun descriptive of each item to be stored in it. E.g.: student, NOT students. Every relation has a schema, which describes the columns, or fields, the relation itself corresponds to our familiar notion of a table: A relation is a collection of tuples, each of which contains values for a fixed number of attributes.

An entity is an object that exists and which is distinguishable from other objects. An entity can be a person, a place, an object, an event, or a concept about which an organization wishes to maintain data. The following are some examples of entities:

**Person: STUDENT, EMPLOYEE, CLIENT**

**Object: COUCH, AIRPLANE, MACHINE**

**Place: CITY, NATIONAL PARK, ROOM, WAREHOUSE**

**Event: WAR, MARRIAGE, LEASE**

**Concept: PROJECT, ACCOUNT, COURSE**

It is important to understand the distinction between an entity type, an entity instance, and an entity set. An entity type defines a collection of entities that have same attributes. An entity instance is a single item in this collection. An entity set is a set of entity instances. The following example will clarify this distinction: STUDENT is an entity type; a student with ID number DMU201 is an entity instance; and a collection of all students is an entity set.

**Attributes** - The items of information which characterize and describe these entities. Attributes are pieces of information about entities. The analysis must of course identify those which are actually relevant to the proposed application.

We represent an entity with a set of attributes. An attribute is a property or characteristic of an entity type that is of interest to an organization. Some attributes of common entity types include the following:

**STUDENT = {Student ID, SSN, Name, Address, Phone, Email, DOB}**

**ORDER = {Order ID, Date of Order, Amount of Order}**

**ACCOUNT = {Account Number, Account Type, Date Opened, Balance}**

**CITY = {City Name, State, Population}**

Attributes will give rise to recorded items of data in the database. At this level we need to know such things as:

- Attribute name: Should be explanatory words or phrases.
- The domain: From which attribute values are taken (A domain is a set of values from which attribute values may be taken.) Each attribute has values taken from a domain. For example, the domain of Name is string and that for salary is real.
- Whether the attribute is part of the entity identifier (attributes which just describe an entity and those which help to identify it uniquely).
- Whether it is permanent or time-varying (which attributes may change their values over time).
- Whether it is required or optional for the entity (whose values will sometimes be unknown or irrelevant).

## **Types of Attributes**

### **1. Simple (atomic) Vs Composite attributes**

A simple or an atomic attribute, such as City or State, cannot be further divided into smaller components. A composite attribute, however, can be divided into smaller subparts in which each subpart represents an independent attribute. Name and Address are examples of composite attributes.

- Simple : Contains a single value (not divided into sub parts) E.g. Age, Gender, etc.
- Composite: Divided into sub parts (composed of other attributes). E.g. Name, address, etc.

## 2. Single-valued Vs multi-valued attributes

Most attributes have a single value for an entity instance; such attributes are called single-valued attributes. A multi-valued attribute, on the other hand, may have more than one value for an entity instance. Languages, which stores the names of the languages that a student speaks. Since a student may speak several languages, it is a multi-valued attribute. All other attributes of the STUDENT entity type are single-valued attributes. For example, a student has only one date of birth and one student identification number. In the E-R diagram, we denote a multi-valued attribute with a double-lined ellipse. Note that in a multi-valued attribute, we always use a double-lined ellipse, regardless of the number of values.

- **Single-valued** : Have only single value (the value may change but has only one value at one time). E.g. Name, Sex, Id. No. color\_of\_eyes, etc.
- **Multi-Valued**: Type of attribute that can have more than one value at a time. E.g. Address, dependent-name, Person may have several college degrees, Language etc.

## 3. Stored vs. Derived Attributes

The value of a derived attribute can be determined by analyzing other attributes. For example, Age is a derived attribute because its value can be derived from the current date and the attribute DateofBirth. An attribute whose value cannot be derived from the values of other attributes is called a stored attribute. a derived attribute Age is not stored in the database. Derived attributes are depicted in the E-R diagram with a dashed ellipse.

- **Stored** : Not possible to derive or compute the values of stored attributes E.g. Name, Address, etc.
- **Derived**: The value of drived attribute may be derived (computed) from the values of other attributes E.g. Age (current year – year of birth). Length of employment (current date - start date). Profit (earning - cost). G.P.A (grade point/credit hours).

## 4. Null Values

- NULL applies to attributes which are not applicable or which do not have values.

- You may enter the value NA (meaning not applicable).

Value of a Primary key attribute can not be null.

Default value - Assumed value if no explicit value.

### Entity versus Attributes

When designing the conceptual specification of the database, one should pay attention to the distinction between an Entity and an Attribute.

- Consider designing a database of employees for an organization:
- Should address be an attribute of Employees or an entity (connected to Employees by a relationship)?
  - If we have several addresses per employee, address must be an entity (attributes cannot be set-valued/multi valued).
  - If the structure (City, Woreda, Kebele, etc) is important, e.g. want to retrieve employees in a given city, address must be modeled as an entity (attribute values are atomic).

**Relationships** :The relationships between entities which exist and must be taken into account when processing information. In any business processing one object may be associated with another object due to some event. Entities in an organization do not exist in isolation but are related to each other. Students take courses and each STUDENT entity is related to the COURSE entity. Faculty members teach courses and each FACULTY entity is also related to the COURSE entity. Consequently, the STUDENT entity is related to the FACULTY entity through the COURSE entity. E-R diagrams can also illustrate relationships between entities. We define a relationship as an association among several entities. Consider, for example, an association between customers of a bank. If customer Abebe has a bank account number 123, then the quality of ownership constitutes a relationship instance that associates the CUSTOMER instance Abebe with the ACCOUNT instance 123. We can think of the relationship instance as a verb that links a subject and an object: customer Abebe has an account; student Elias registers for a course; professor Fekadu teaches a course. A relationship set is a grouping of all matching relationship instances, and the term relationship type refers to the relationship between entity types.

- One external event or process may affect several related entities.
- Related entities require setting of links from one part of the database to another.
- A relationship should be named by a word or phrase which explains its function.
- Role names are different from the names of entities forming the relationship: one entity may take on many roles, the same role may be played by different entities.
- For each relationship, one can talk about the number of entities and the number of tuples participating in the association. These two concepts are called degree and cardinality of a relationship respectively.

An **entity** is an object that exists and that is distinguishable from other objects.

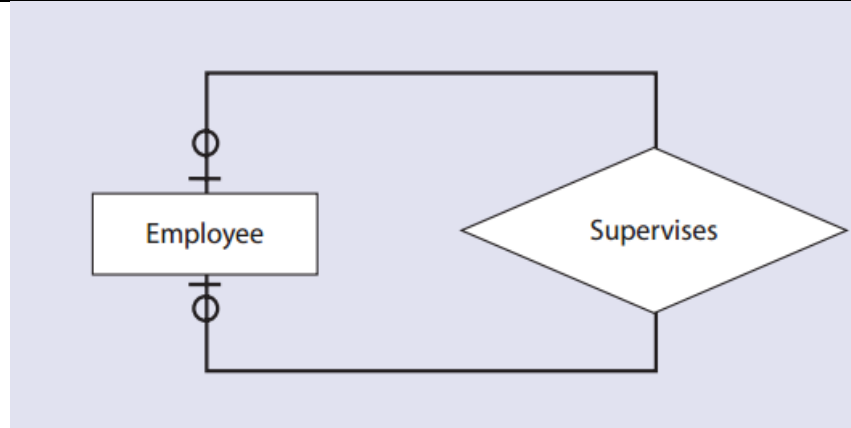
An **attribute** is a property or characteristic of an entity type that is of interest to an organization.

A **relationship** is an association among several entities.

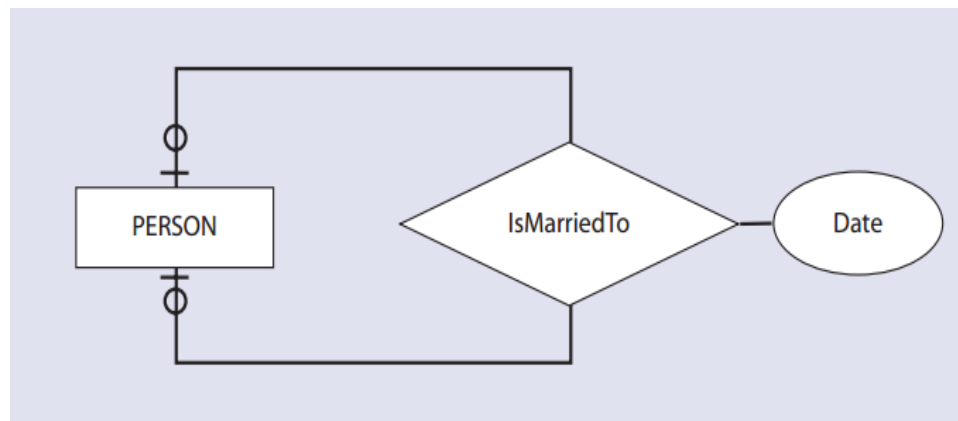
### **Degree of a Relationship**

Degree of relationship is an important point about a relationship which concerns how many entities are participate in it. The number of entities participating in a relationship is called the degree of the relationship. Among the Degrees of relationship, the following are the basic:

**Unary/Recursive Relationship:** Tuples/records of a single entity are related with each other. Unary Relationship A unary relationship R is an association between two instances of the same entity type. For example, two students are roommates and stay together in an apartment. The following relationship instance exists whenever an employee supervises another employee. The relationship Supervises is a one-to-many relationship since an employer can supervise many employees but a supervisee can have only one supervisor. The minimum cardinality for supervising is zero (an employee may not supervise anyone) but the minimum cardinality of being supervised is one (every employee must be supervised).



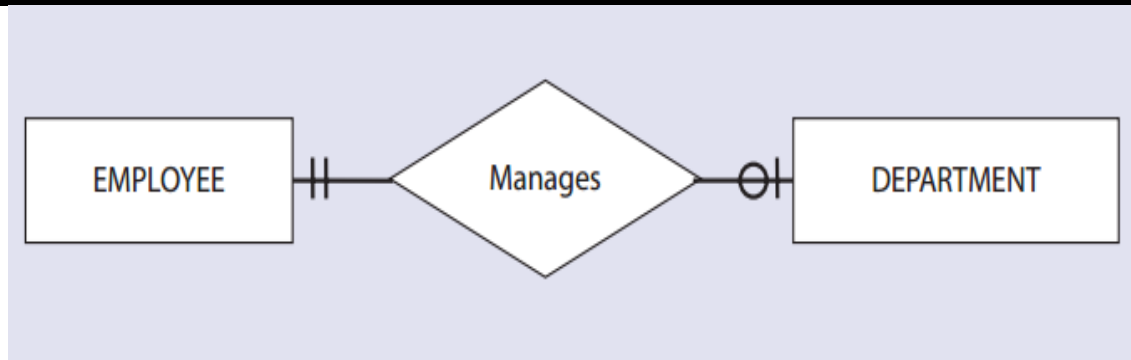
Whenever two people in the entity type PERSON get married, the relationship instance IsMarriedTo is created. The Date of marriage is an attribute of this relationship. Since a person can only be married to one other person, marriage is a one-to-one relationship. Furthermore, since a person can be unmarried, the minimum cardinality of the IsMarriedTo relationship is zero.



**Binary Relationships:** Tuples/records of two entities are associated in a relationship.

**Binary Relationship** A binary relationship R is an association between two instances of two different entity types. For example, in a university, a binary relationship exists between a student (STUDENT entity) and an instructor (FACULTY entity) of a single class; an instructor teaches a student.

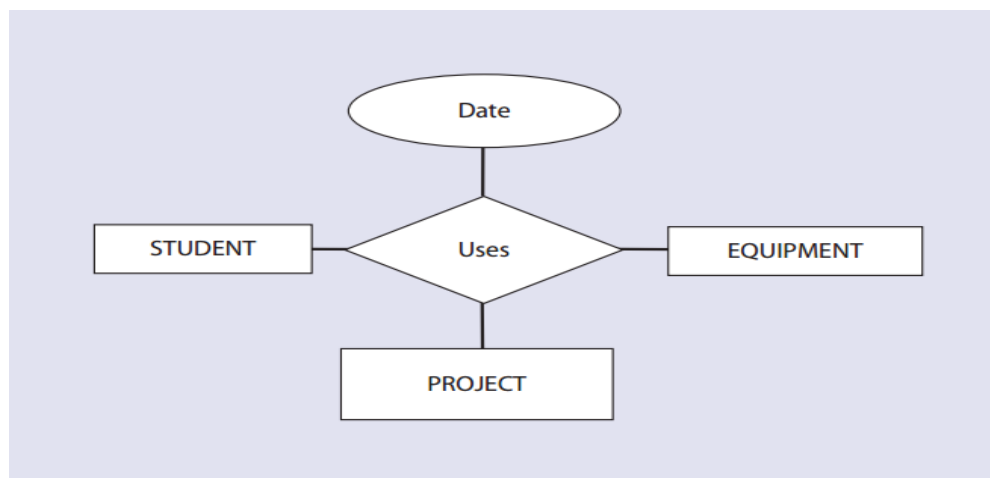
The following EMPLOYEE and DEPARTMENT relationship is a one-to-one relationship since we assume that an employee can manage at most one department and each department is managed by at most one employee. The minimum cardinality can be determined since each department must be managed by an employee, but not all employees manage department.



**Ternary Relationship:** Tuples/records of three different entities are associated.

students use equipment to work on projects; each instance of Uses involves an instance of STUDENT, PROJECT, and EQUIPMENT. If a student uses two pieces of equipment to work on a project, there are two instances of the relationship Uses. A campus lab may use the attribute in this ternary relationship, the Date of use, to log the equipment usage.

A ternary relationship R is an association between three instances of three different entity types. For example, consider a student using certain equipment for a project. In this case, the STUDENT, PROJECT, and EQUIPMENT entity types relate to each other with ternary relationships: a student checks out equipment for a project.



**n-nary relationship:** A generalized degree of relationship in which tuples from arbitrary number of entity sets are participating in a relationship.

### Cardinality of a Relationship

Another important concept about relationship is the number of instances/tuples that can be associated with a single instance from one entity in a single relationship. The number of



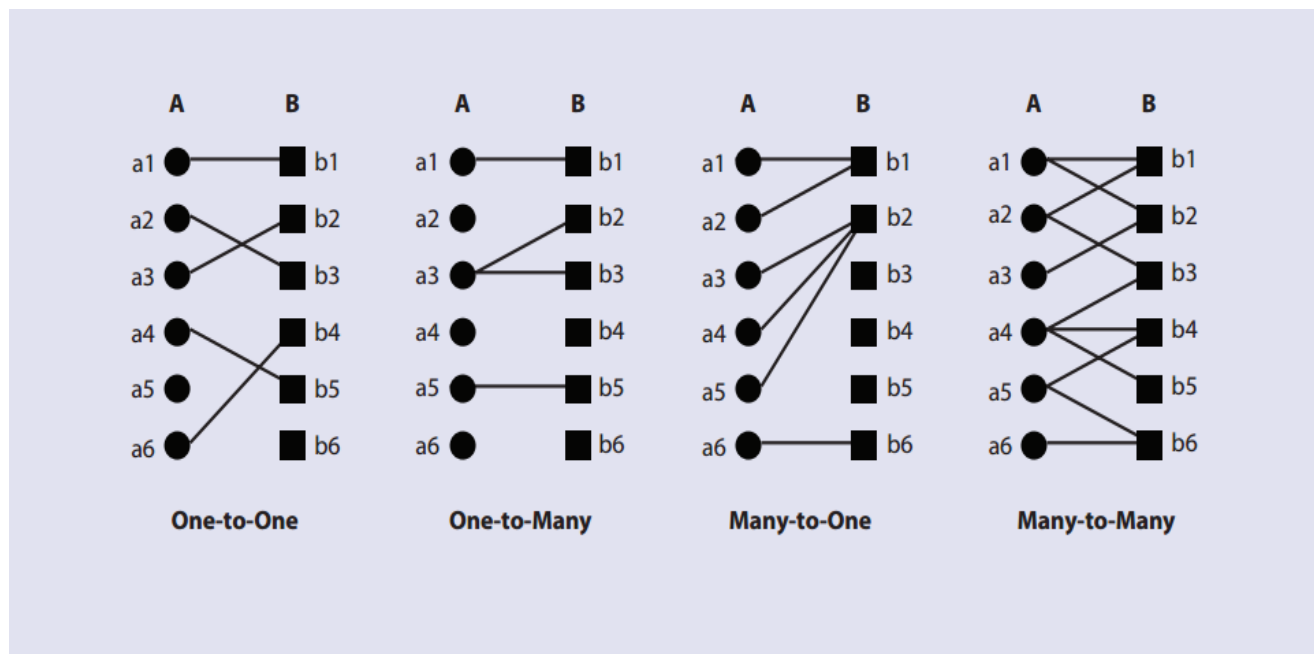
instances participating or associated with a single instance from an entity in a relationship is called the cardinality of the relationship. The major cardinalities of a relationship are:

**One-to-One:** one tuple is associated with only one other tuple. E.g. Building -to- Location, as a single building will be located in a single location and as a single location will only accommodate a single Building.

**One-to-Many:** one tuple can be associated with many other tuples, but not the reverse. E.g. Department-to-Student, as one department can have multiple students.

**Many-to-One:** many tuples are associated with one tuple but not the reverse. E.g. Employee -to-Department: as many employees belong to a single department.

**Many-to-Many:** one tuple is associated with many other tuples and from the other side, with a different role name one tuple will be associated with many tuples. E.g. Student-to-Course, as a student can take many courses and a single course can be attended by many students.

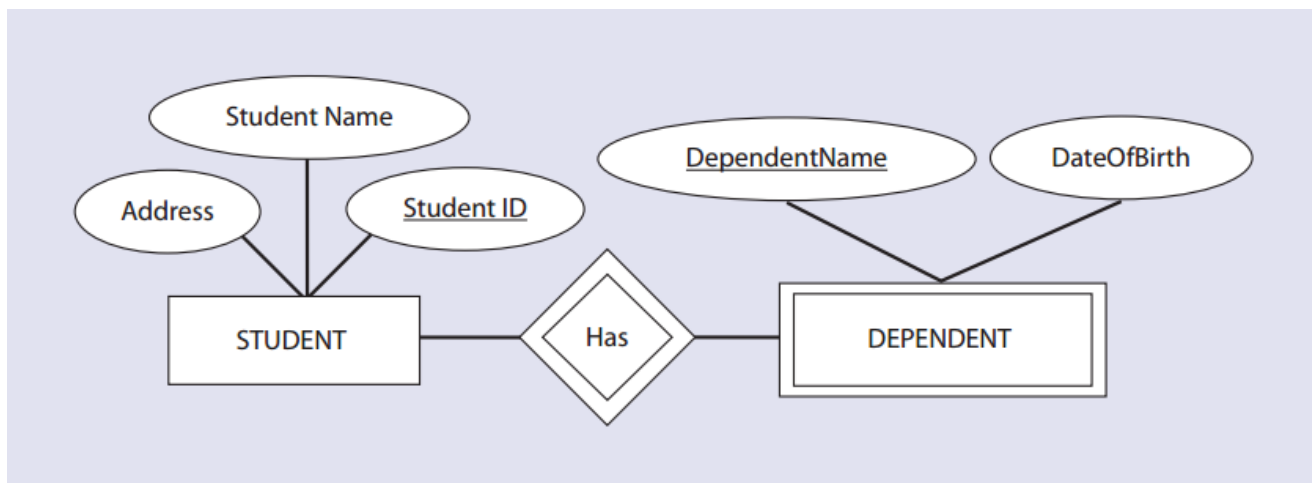


The number (unary, binary, or ternary) of entity sets that participate in a relationship is called the degree of relationship.

The cardinality of relationship represents the minimum/maximum number of instances of entity B that must/can be associated with any instance of entity A.

## Types of Entities

Entity types can be classified into two categories: strong entity types and weak entity types. A strong entity type exists independent of other entity types, while a weak entity type depends on another entity type. Consider a student database that includes an entity STUDENT. Suppose that we also record data about each student's dependents, such as a spouse or children, in this database. To do so, we must create the entity type DEPENDENT. The entity type DEPENDENT does not exist on its own and owes its existence to the STUDENT entity type. When students graduate and their records are removed from the STUDENT entity set, the records of their dependents are also removed from the DEPENDENT entity set. In the E-R diagram, a weak entity is indicated by a double-lined rectangle. The corresponding relationship diamond is also double-lined. The entity type on which a weak entity type depends is called the identifying owner (or simply owner), and the relationship between a weak entity type and its owner is called an identifying relationship.



**Fig. The weak entity in an E-R diagram**

STUDENT is the owner and Has is the identifying relationship. For a weak entity, we define a partial key attribute that, when combined with the key attribute of its owner, provides the full identifier for the weak entity. DependentName is the partial identifying attribute. When we combine it with the StudentID, it uniquely identifies the dependent. Of course, in this example, we make an implicit assumption that people do not give the same name to more than one of their children.

An associative entity is an entity type that connects the instances of one or more entity types and contains attributes particular to this association.

A strong entity type exists independent of other entity types, while a weak entity type depends on another entity type.

### **Relational Constraints/Integrity Rules**

Relational Integrity:

**Domain integrity:** No value of the attribute should be beyond the allowable limits.

**Entity integrity:** In a base relation, no attribute of a Primary Key can assume a value of NULL.

**Referential integrity:** If a Foreign Key exists in a relation, either the Foreign Key value must match a Candidate Key value in its home relation or the Foreign Key value must be NULL.

**Enterprise integrity:** Additional rules specified by the users or database administrators of a database are incorporated.

### **Keys and Constraints**

If tuples are need to be unique in the database, and then we need to make each tuple distinct.

To do this we need to have relational keys that uniquely identify each relation.

**A super key :** A super key also know as super set is then a set of one or more attributes that in group (collectively) can identify an entity uniquely from the entity set.

Example: Consider the “EMPLOYEES” entity set, then

- “EmpId”, “EmpId, Name”, “NationalId”, “NationalId, BDate”, ... are super keys
- “Name”, “BDate” are NOT super keys

Super Key: an attribute or set of attributes that uniquely identifies a tuple within a relation.

**Note:** If K is a super set (super key) then a set consisting of K is also a super set.

The more interesting super set is the minimal super set that is referred to as the candidate key.

The candidate key is the sufficient and the necessary set of attributes to distinguish an entity set.

Example: In the “EMPLOYEES” entity set {EmpId}, {NationalId}, {Name, BDate} (assuming that there is no coincidence that employees with the same name may born on the same day) are candidate keys.

The designer of the database is the one that makes the choice of the candidate keys for implementation, but the choice has to be made carefully. Primary key is a term used to refer to the candidate key that is selected by the designer for implementation.

**Candidate Key:** an attribute or set of attributes that uniquely identifies individual occurrences of an entity type or tuple within a relation.

The candidate key is the sufficient and the necessary set of attributes to distinguish an entity set. Are individual columns in a table that qualifies for uniqueness of each row/tuple.

Example: Employee(EmpID, EmpName, SSN, DeptID, DOB) = EmpID, SSN

- For a Primary Key and thus are Candidate keys.
- Are super keys for which no proper subset is a super key
- In other words candidate keys are minimal super keys

A candidate key has two properties:

1. Uniqueness
2. Irreducibility

Candidate Key: a super key such that no proper subset of that collection is a Super Key within the relation.

**Alternative Key:** Candidate column other the Primary column, like if EmployeeID is set for a PK then SSN would be the Alternate key.

Example: Employee(EmpID, EmpName, SSN, DeptID, DOB) = SSN

**Composite key:** A candidate key that consists of two or more attributes. If a table do have a single column that qualifies for a Candidate key, then you have to select 2 or more columns to make a row unique. Like if there is no EmployeeID or SSN columns, then you can make

Example: EmployeeName + DOB as Composite Primary Key.

But still there can be a narrow chance of duplicate rows

Employee(EmpID, EmpName, SSN, DeptID, DOB)

**Primary key:** the candidate key that is selected to identify tuples uniquely within the relation.

The entire set of attributes in a relation can be considered as a primary case in a worst case.

In another way, an entity type may have one or more possible candidate keys, one of which is selected to be a primary key.

Properties of Primary key

- No two rows can have the same primary key value
- Every row must have a primary key value
- The primary key field cannot be null
- Value in a primary key column can never be modified or updated, if any foreign key refers to that primary key.

**Foreign key:** an attribute, or set of attributes, within one relation that matches the candidate key of some relation. A foreign key is a link between different relations to create the view or the unnamed relation.

Employee
EmployeeID
EmployeeName
SSN
<u>DeptID</u>
DOB

Department
<u>DeptID</u>
DeptName

**Unique Key:** Unique key is same as primary with the difference being the existence of null.

Unique key field allows one value as NULL value.

Employee
EmployeeID
EmployeeName
SSN
<u>EmailID</u>
DOB

## Relational Views

Relations are perceived as a table from the users' perspective. Actually, there are two kinds of relation in relational database. The two categories or types of relations are Base (Named) and View (Unnamed) Relations. The basic difference is on how the relation is created, used and updated:

**Base Relation:** A named relation corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.

**View (Unnamed Relation):** A View is the dynamic result of one or more relational operations operating on the base relations to produce another virtual relation that does not actually exist as presented. So a view is virtually derived relation that does not necessarily exist in the database but can be produced upon request by a particular user at the time of request. The virtual table or relation can be created from single or different relations by extracting some attributes and records with or without conditions.

### **Purpose of a view**

- Hides unnecessary information from users: since only part of the base relation (Some collection of attributes, not necessarily all) are to be included in the virtual table.
- Provide powerful flexibility and security: since unnecessary information will be hidden from the user there will be some sort of data security.
- Provide customized view of the database for users: each users are going to be interfaced with their own preferred data set and format by making use of the Views.
- A view of one base relation can be updated.
- Update on views derived from various relations is not allowed since it may violate the integrity of the database.
- Update on view with aggregation and summary is not allowed. Since aggregation and summary results are computed from a base relation and does not exist actually.

### **Schemas and Instances**

When a database is designed using a relational data model, all the data is represented in a form of a table. In such definitions and representation, there are two basic components of the database. The two components are the definition of the relation or the table and the actual

data stored in each table. The data definition is what we call the Schema or the skeleton of the database and the relations with some information at some point in time is the Instance or the flesh of the database.

## **Schemas**

Schema describes how data is to be structured, defined at setup/design time (also called "metadata"). Since it is used during the database development phase, there is rare tendency of changing the schema unless there is a need for system maintenance which demands change to the definition of a relation.

**Database Schema (Intension):** specifies name of relation and the collection of the attributes (specifically the Name of attributes).

- Refer to a description of database (or intention)
- Specified during database design
- Should not be changed unless during maintenance

**Schema Diagrams:** convention to display some aspect of a schema visually.

**Schema Construct:** refers to each object in the schema (e.g. STUDENT)

E.g.: STUNEDT (FName,LName,Id,Year,Dept,Sex)

## **Instances**

Instance: is the collection of data in the database at a particular point of time (snap-shot).

- Also called State or Snap Shot or Extension of the database.
- The actual data stored in a
- database at a particular moment in time. Also called database state (or occurrence).
- State of database is changed any time we add, delete or update an item.
- **Valid state:** the state that satisfies the structure and constraints specified in the schema and is enforced by DBMS.

Since instance is actual data of database at some point in time, changes rapidly. To define a new database, we specify its database schema to the DBMS (database is empty). Database is initialized when we first load it with data.

## **ENTITY - RELATIONSHIP DIAGRAMS (E-RD)**

As one important aspect of E-Relationship data modeling, database designers represent their data model by E-R diagrams. These diagrams enable designers and users to express their understanding of what the planned database is intended to do and how it might work, and to communicate about the database through a common language. Each organization that uses E-R diagrams must adopt a specific style for representing the various components.

### Graphical Representations in ER Diagramming

- Entity is represented by a rectangle containing the name of the entity.

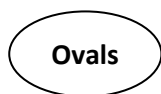


**Strong Entity**



**Weak Entity**

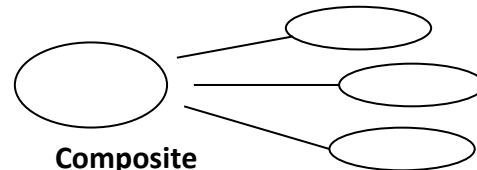
- Connected entities are called relationship participants
- Attributes are represented by ovals and are connected to the entity by a line.





**Attribute**

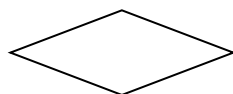


**Multi-valued  
Attribute**

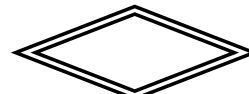


**Composite  
Attribute**

- A derived attribute is indicated by a dotted line. (.....) 
- Primary Keys are underlined. 
- Relationships are represented by Diamond shaped symbols



**Strong Relationship**



**Weak Relationship**

**An entity-relationship model (ERM)** is a model that provides a high-level description of a conceptual data model. Data modeling that provides a graphical notation for representing such data models in the form of entity-relationship diagrams (ERD).



The whole purpose of ER modeling is to create an accurate reflection of the real world in a database. The ER model doesn't actually give us a database description. It gives us an intermediate step from which it is easy to define a database.

The E-R data model is based on a perception of a real world that consists of a set of basic objects called **entities**, and of **relationships** among these objects. It was developed to facilitate database design by allowing the specification of an enterprise schema, which represents the overall logical structure of a database.

The E-R data model is one of several semantic data models; the semantic aspect of the model lies in the attempt to represent the meaning of the data. The E-R model is extremely useful in mapping the meanings and interactions of real-world enterprises onto a conceptual scheme.

Because of this utility, many database design tools draw on concepts from the E-R model.

A data model in which information stored in the database is viewed as sets of entities and sets of relationships among entities. There are four basic notions that the ER Model employs: entity sets/type, relationships, attributes, and constraints.