# Entity-Relationship Diagram

An Entity-Relationship Diagram (ERD) is a visual representation used in database design to depict the structure and relationships between different entities (or tables) within a database system. ERDs are a critical tool for database developers, designers, and stakeholders to understand and communicate how data is organized and connected in a database.

Entities: These represent the main objects or concepts in the database, such as customers, products, employees, or orders. Each entity is typically depicted as a rectangle in the diagram.

Attributes: Attributes describe the properties or characteristics of each entity. For example, a "Customer" entity may have attributes like "CustomerID," "FirstName," "LastName," and "Email.";

Relationships: Relationships illustrate how entities are connected to one another. They show how data is related and can be used to establish links between different entities. Common types of relationships include one-to-one, one-to-many, and many-to-many.

# Entities

Entities can be categorized into different types based on their roles and characteristics within the database model.

**Strong Entities**: These are entities that can exist independently and have their attributes. They represent primary objects in the database. For example, in a university database, the "Student" entity is a strong entity with attributes like student ID, name, and date of birth.

**Weak Entities**: Weak entities depend on a related strong entity for their existence. They do not have a primary key attribute of their own but instead have a partial key that connects them to a strong entity. A common example is the "Address" entity, which may rely on a "Person" entity to exist.

**Associative Entities**: Also known as junction entities or linking entities, associative entities are used to represent many-to-many relationships between two other entities. They often contain attributes that are specific to the relationship itself. For instance, in a library database, an "Authorship" entity could link "Author" and "Book" entities, indicating which author(s) wrote which book(s).

**Derived Entities**: These entities contain attributes that can be calculated or derived from other attributes within the database. They are often used to store information that can be computed when needed, rather than being explicitly stored. An example might be a "Total Price" entity in an e-commerce system, which can be derived from the quantity and unit price of items in a shopping cart.

Subtypes and Supertypes: Subtype entities represent specific categories or subsets of a supertype entity. This is commonly used in situations where entities share common attributes but also have attributes unique to their subtype. For instance, "Employee" could be a supertype with subtypes "Manager" and "Clerk," each having their own set of attributes.

**Singleton Entities:** These entities represent a **single**, unique instance within the database. They are often used for settings or configuration data that should have only one record. For example, a "Settings" entity might contain global settings for an application.

**Historical Entities**: These entities are used to store historical data, such as records of past transactions or changes in status. Historical entities often have attributes like "Effective Date" and "End Date" to track when data was valid.

**External Entities:** In some database models, external entities represent entities from external systems or sources that interact with the database. They are often used in data modeling to illustrate how the database interfaces with external systems or data.

# **Types of Attributes**

**Attributes:** Attributes are **characteristics or properties** that describe entities. Attributes provide detailed information about the entities represented in a database.

**Simple Attributes:** These are atomic attributes that cannot be divided any further. For instance, "EmployeeID" or "FirstName" are simple attributes because they represent singular data elements.

**Composite Attributes**: Composite attributes can be divided into sub-parts that have independent meaning. For example, an "Address" attribute may consist of sub-attributes like "Street," "City," "State," and "ZIP Code."

**Derived Attributes:** Derived attributes are calculated or derived from other attributes within the database. They do not store data directly but instead are generated based on certain rules or calculations. For instance, the "Age" attribute can be derived from the "DateOfBirth" attribute by subtracting the birthdate from the current date.

**Multi-valued Attributes**: Multi-valued attributes can have multiple values associated with them for a single entity. For example, an "Interests" attribute for a "Person" entity could contain multiple hobbies or interests.

**Key Attributes**: Key attributes are used to uniquely identify each instance of an entity. In many cases, one of the attributes is designated as the primary key, which ensures its uniqueness within the entity.

**Composite Key**: A composite key consists of multiple attributes that, when combined, uniquely identify an entity. This is often used when a single attribute is insufficient for ensuring uniqueness.

**Null Attributes**: Null attributes are those that may not have a value for every instance of an entity. They are used to represent missing or unknown data.

# **Relationship**

**Relationship:** A relationship refers to the association or connection between two or more entities within a database. Relationships are used to model how data in different entities is related or connected to each other; They provide a way to represent the interactions and dependencies between entities in a structured and organized manner.

**Cardinality**: Cardinality in a relationship defines the number of instances (records or rows) in one entity that can be associated with the number of instances in another entity. It specifies the nature of the relationship;

One-to-One (1:1): Each instance in one entity is associated with exactly one instance in another entity;

One-to-Many (1:N): Each instance in one entity is associated with one or more instances in another entity, but each instance in the second entity is associated with only one instance in the first entity;

**Many-to-Many (N:N):** Many instances in one entity can be associated with many instances in another entity. To represent many-to-many relationships in a database;

# **Multiplicity**

The Multiplicity attribute of a relationship specifies the cardinality or number of instances of an EntityType that can be associated with the instances of another EntityType;

multiplicity refers to the numeric constraints or restrictions placed on the number of instances that can be associated between two entities in a relationship.

Common multiplicity constraints include;

**One (1):** This constraint indicates that one instance from one entity can be associated with one instance from another entity, it represents a **one-to-one (1:1)** relationship.

**Zero or One (0..1)**: This constraint means that an instance from one entity can be associated with either zero instances or one instance from another entity, and vice versa. It represents a one-to-one (1:1) relationship where the **association is optional for one or both entities**;

**One or Many (1..N)**: This constraint signifies that one instance from one entity can be associated with one or more instances from another entity, but each instance from the second entity can be associated with only one instance from the first entity. It represents a one-to-many (1:N) relationship.

**Zero or Many (0..N)**: This constraint indicates that an instance from one entity can be associated with zero or more instances from another entity, and vice versa. It represents a one-to-many (1:N) relationship where the association is optional for one or both entities.

**Exactly N (N)**: This constraint specifies that an instance from one entity must be associated with exactly N instances from another entity, and vice versa. It represents a **one-to-N (1:N) relationship with a fixed number of associations**.

**Many (N):** This constraint means that an instance from one entity can be associated with many instances from another entity, and vice versa. It represents a many-to-many (N:N) relationship;

# **Diagram Notations**