*Entity Relationship (ER) Modeling*

The **Entity-Relationship Model (ERM)** is a fundamental tool in database design that provides a way to visually represent the structure of a database. It focuses on the key elements such as entities, relationships, and attributes, which form the foundation of an **Entity-Relationship Diagram (ERD)**. Let's break down each component in detail, along with examples, to understand the ERD.

**1. Entity-Relationship Model (ERM) and ER Diagrams (ERD)**

* **ERM**: This model defines the database structure in terms of entities, their attributes, and the relationships between these entities.
* **ERD**: The ER diagram is a graphical representation of the conceptual database design as viewed by the end user. It shows how the entities relate to each other within the system.

The ERD has three main components:

1. **Entities**
2. **Attributes**
3. **Relationships**

**2. Entities**

* An **Entity** is something that exists in the real world and is represented as a collection of similar objects or items. For example, entities could be **CUSTOMER**, **EMPLOYEE**, **PRODUCT**, or **ORDER**.
* An entity refers to an **entity set** (a collection of similar objects) and not to a single occurrence of that entity. For example:
  + **CUSTOMER** refers to the entire set of customers, not just one specific customer.
* In database terms, an entity is similar to a **table** in a relational database, not to a single **row** (record).

**Representation in ERD**

* In **Chen notation** and **Crow’s Foot notation**, entities are represented by rectangles.
* The **name of the entity** (which is always a noun) is written in **capital letters** inside the rectangle.

**Example**:

* An entity named **EMPLOYEE** could be represented as:
  + In Chen notation: A rectangle with the word **EMPLOYEE** inside it.
  + In Crow’s Foot notation: Also a rectangle with the word **EMPLOYEE** inside it.

**3. Attributes**

* Attributes are the **characteristics or properties** of an entity that provide more details about it. For example:
  + An **EMPLOYEE** entity might have attributes like **Employee\_ID**, **First\_Name**, **Last\_Name**, and **Date\_of\_Hire**.

**Representation in ERD**

* In **Chen notation**:
  + Attributes are represented by **ovals** connected to the entity's rectangle with a line.
  + Each oval contains the **name of the attribute** it represents.
* In **Crow’s Foot notation**:
  + Attributes are listed in an **attribute box** below the entity rectangle.

**Attribute Types:**

1. **Required Attribute**: Must have a value (e.g., **Employee\_ID** must have a unique value).
2. **Optional Attribute**: May be left empty (e.g., **Middle\_Name** may or may not have a value).
3. **Domain**: The set of possible values for an attribute (e.g., the domain for **Age** might be 0-100).
4. **Identifiers**: One or more attributes that uniquely identify each entity instance.
   * **Composite Identifier**: A primary key composed of more than one attribute (e.g., **Order\_ID** and **Product\_ID** combined).
5. **Composite Attribute**: Can be subdivided into smaller attributes (e.g., **Full\_Name** can be divided into **First\_Name** and **Last\_Name**).
6. **Simple Attribute**: Cannot be subdivided (e.g., **Gender**).
7. **Single-value Attribute**: Has only one value for each entity instance (e.g., **Date\_of\_Birth**).
8. **Multivalued Attribute**: Can have many values for a single entity instance (e.g., **Phone\_Numbers**).

**4. Relationships**

* **Relationships** define how entities interact with one another in a database. They express the logical associations between different entities.
* For example, an **EMPLOYEE** might **WORKS\_IN** a **DEPARTMENT**, or a **CUSTOMER** might **PLACES** an **ORDER**.

**Representation in ERD**

* Relationships can be represented differently in Chen notation and Crow’s Foot notation:
  + **Chen notation**: Uses diamond shapes to represent relationships, with lines connecting the entities involved.
  + **Crow’s Foot notation**: Uses symbols to show the relationship's nature directly on the lines between entities.

**Example ERD using Chen and Crow’s Foot Notations**

**Example with EMPLOYEE entity and its attributes:**

* **EMPLOYEE** entity with attributes: **Employee\_ID**, **First\_Name**, **Last\_Name**, **Date\_of\_Hire**, and **Phone\_Numbers**.

1. **Chen Notation**:
   * A rectangle labeled **EMPLOYEE**.
   * Ovals connected to the rectangle representing attributes: **Employee\_ID**, **First\_Name**, **Last\_Name**, **Date\_of\_Hire**, **Phone\_Numbers**.
2. **Crow’s Foot Notation**:
   * A rectangle labeled **EMPLOYEE**.
   * Attributes listed directly in an attribute box inside or below the rectangle.

In both cases:

* **Employee\_ID** would be a **required** and **identifier** attribute.
* **Phone\_Numbers** could be a **multivalued attribute**.

**Summary**

* **Entities** represent real-world objects or sets of objects and correspond to tables in a database.
* **Attributes** describe the properties of these entities and can be represented differently in various notations (Chen or Crow’s Foot).
* **Relationships** illustrate how entities are connected and interact with each other in the database.

This structured approach helps in designing a conceptual model that can later be converted into a logical model, and eventually into a physical database schema.

**Relationships in Entity-Relationship Models**

In the **Entity-Relationship Model (ERM)**, a **relationship** represents an association or link between two or more entities. Understanding how entities relate to each other is crucial for defining the logical structure of a database. Let's delve into the details of relationships, their classification, and some key concepts.

**Key Concepts of Relationships**

1. **Participants**:
   * **Participants** are the entities involved in a relationship. They are the elements that interact or are connected through this relationship.
   * For example, if we have an **EMPLOYEE** entity and a **DEPARTMENT** entity, these two entities can participate in a relationship called **WORKS\_IN**.
2. **Directionality of Relationships**:
   * Relationships between entities **always operate in both directions**, which means you can look at the relationship from either entity's perspective.
   * For example:
     + From the **EMPLOYEE** perspective, you might say: "An **EMPLOYEE** **works in** a **DEPARTMENT**."
     + From the **DEPARTMENT** perspective, you could say: "A **DEPARTMENT** **has** one or more **EMPLOYEES**."
3. **Classification of Relationships**: Relationships can be classified based on the **cardinality**, which defines the number of instances of one entity that can be associated with instances of another entity. The most common types are:
   * **One-to-Many (1**

**)**:

* + - In a 1

relationship, a single instance of one entity is related to multiple instances of another entity.

* + - **Example**: **DEPARTMENT** and **EMPLOYEE**:
      * One **DEPARTMENT** can have many **EMPLOYEES**, but each **EMPLOYEE** works in only one **DEPARTMENT**.
      * In this case, **DEPARTMENT** is on the "one" side, and **EMPLOYEE** is on the "many" side.
  + **Many-to-One (M:1)**:
    - This is simply the reverse perspective of a 1

relationship.

* + - From the **EMPLOYEE** to **DEPARTMENT** viewpoint, this relationship is M:1 because multiple **EMPLOYEES** work in one **DEPARTMENT**.
  + **One-to-One (1:1)**:
    - In this type of relationship, a single instance of one entity is related to a single instance of another entity.
    - **Example**: Suppose there is an entity called **PERSON** and another called **PASSPORT**. A person can have only one passport, and each passport is issued to only one person.
  + **Many-to-Many (M**

**)**:

* + - In an M

relationship, many instances of one entity can be related to many instances of another entity.

* + - **Example**: **STUDENT** and **COURSE**:
      * A student can enroll in many courses, and a course can have many students enrolled.

**Challenges in Establishing Relationships**

One of the challenges in establishing relationships is when you have information from only one side of the relationship. It becomes difficult to determine the correct classification without knowing how both entities interact with each other.

**Example Scenario**

* Suppose we know that an **EMPLOYEE** entity exists and that there is a relationship with a **PROJECT** entity, but we don't have information about the **PROJECT** side.
* Without knowing whether each **EMPLOYEE** works on one project or multiple projects, or if a project can have multiple employees, it would be hard to classify the relationship correctly.

To correctly classify a relationship, you must understand how both entities participate in that relationship. Knowing both sides of the relationship allows you to determine whether it is **1**

, **M:1**, **1:1**, or **M**

.

**Example Relationships and Their Representations**

**Example: One-to-Many (1) Relationship**

**Entities**: **CUSTOMER** and **ORDER**

* **Relationship**: **PLACES**
* **Explanation**: A **CUSTOMER** can place many **ORDERS**, but each **ORDER** is associated with only one **CUSTOMER**.

**Representation**:

* In **Chen notation**: The **PLACES** relationship is represented by a diamond shape, with lines connecting the **CUSTOMER** (one side) to **ORDER** (many side).
* In **Crow’s Foot notation**: A straight line connects **CUSTOMER** to **ORDER**, with a crow's foot symbol indicating the "many" side of the relationship.

**Example: Many-to-Many (M) Relationship**

**Entities**: **STUDENT** and **COURSE**

* **Relationship**: **ENROLLS\_IN**
* **Explanation**: A **STUDENT** can enroll in many **COURSES**, and each **COURSE** can have many **STUDENTS**.

**Representation**:

* In **Chen notation**: The **ENROLLS\_IN** relationship is represented by a diamond shape with a line to **STUDENT** (many side) and a line to **COURSE** (many side).
* In **Crow’s Foot notation**: The crow's foot symbol appears on both sides, indicating a many-to-many relationship.

Understanding **Connectivity**, **Cardinality**, **Existence Dependence**, and **Relationship Strength** is crucial in database design, as these concepts help define the precise interactions between entities. Let’s break down each term with examples to make these concepts clear.

**1. Connectivity and Cardinality**

**Connectivity**

* **Connectivity** describes the **type of relationship** between entities. It classifies the relationships based on how entities are connected to each other.
* Common relationship types include:
  + **One-to-One (1:1)**: A single occurrence of one entity is related to a single occurrence of another entity.
  + **One-to-Many (1)**: A single occurrence of one entity can be related to many occurrences of another entity.
  + **Many-to-Many (M)**: Multiple occurrences of one entity can be related to multiple occurrences of another entity.

**Example**:

* **EMPLOYEE** and **DEPARTMENT** entities:
  + If an **EMPLOYEE** can only work in one **DEPARTMENT**, but a **DEPARTMENT** can have many **EMPLOYEES**, this is a **One-to-Many (1)** relationship.

**Cardinality**

* **Cardinality** defines the **minimum and maximum number of instances** of one entity that can be associated with instances of another entity. It specifies how many entities can or must participate in a relationship.
* Cardinality is often defined using business rules that establish these limits.

**Example**:

* Consider the relationship between **STUDENT** and **COURSE**:
  + Minimum cardinality: A **STUDENT** must enroll in at least one **COURSE** (so the minimum is 1).
  + Maximum cardinality: A **STUDENT** can enroll in up to 5 **COURSES** (so the maximum is 5).

Cardinality is expressed in this form: (min, max). For example, (1, 5) indicates that a **STUDENT** must enroll in at least 1 course and can take up to 5 courses.

**2. Existence Dependence and Existence Independence**

**Existence Dependence**

* An entity is **existence-dependent** if it **cannot exist without** being associated with another entity. It must be linked to another entity occurrence to be valid.
* Such an entity is also called a **weak entity**.

**Example**:

* **ORDER** and **ORDER\_ITEM** entities:
  + An **ORDER\_ITEM** cannot exist without an **ORDER**. Each **ORDER\_ITEM** must be linked to an **ORDER** to be meaningful, making it **existence-dependent**.

**Existence Independence**

* An entity is **existence-independent** if it can exist on its own, without needing to be associated with another entity. These entities are often referred to as **strong entities** or **regular entities**.

**Example**:

* **CUSTOMER** entity:
  + A **CUSTOMER** can exist independently of any **ORDER**. Even if a customer has not placed any orders, they still exist as a record in the database.

**3. Relationship Strength**

Relationship strength depends on the **primary key (PK)** of the entities involved in the relationship. There are two types of relationships based on this:

**Weak (Non-Identifying) Relationships**

* A **weak relationship** (also called a non-identifying relationship) exists if the **primary key of the related entity** does not contain the **primary key component of the parent entity**.
* In this case, the child entity does not depend on the parent entity for its identification.

**Example**:

* **DEPARTMENT** and **EMPLOYEE** entities:
  + If the **EMPLOYEE** entity does not have a primary key that includes the **DEPARTMENT\_ID** from the **DEPARTMENT** entity, then it is a **weak relationship**.
  + The **EMPLOYEE** entity can still be uniquely identified by its own **Employee\_ID**, which does not depend on the **DEPARTMENT\_ID**.

**Strong (Identifying) Relationships**

* A **strong relationship** (also called an identifying relationship) exists when the **primary key of the related entity** contains the **primary key component of the parent entity**.
* This relationship indicates a tighter connection between entities.

**Example**:

* **ORDER** and **ORDER\_ITEM** entities:
  + If the **ORDER\_ITEM** primary key includes the **Order\_ID** (from the **ORDER** entity), it indicates that an **ORDER\_ITEM** is uniquely identified only when combined with the **Order\_ID**, making it a **strong relationship**.

**4. Weak Entities**

**Weak Entity Characteristics**

A **weak entity** must satisfy two conditions:

1. **Existence-dependent**: It cannot exist without a parent entity.
2. Its **primary key is partially or totally derived** from the parent entity in the relationship.

The database designer determines whether an entity is weak based on **business rules** that specify how these entities interact.

**Example**:

* **ORDER\_ITEM** as a weak entity:
  + The **ORDER\_ITEM** entity is existence-dependent on the **ORDER** entity.
  + The primary key of **ORDER\_ITEM** might be composed of both **Order\_ID** and **Product\_ID**, where **Order\_ID** is derived from the **ORDER** entity.

This makes **ORDER\_ITEM** a **weak entity** because it depends on the **ORDER** to exist and is identified by the combination of its own key and a foreign key.

**Summary**

* **Connectivity** describes the type of relationship between entities, such as **1:1**, **1**, or **M**.
* **Cardinality** expresses the minimum and maximum number of instances that can or must be associated with entities in a relationship.
* **Existence Dependence** means an entity cannot exist without being related to another entity, making it a **weak entity**.
* **Existence Independence** allows an entity to exist without depending on another, making it a **strong entity**.
* **Relationship Strength** is determined by whether the primary key of the related entity contains the primary key component of the parent entity:
  + **Weak relationship**: Does not include the parent entity’s primary key.
  + **Strong relationship**: Includes the parent entity’s primary key.
* A **weak entity** must be both existence-dependent and have a primary key derived from the parent entity.

These concepts help define the structure of the database and ensure that entities and their relationships are correctly modeled to reflect the business rules and real-world scenarios.

Understanding **Relationship Participation** and **Relationship Degree** is crucial in database design, as these concepts define how entities interact within a database schema. Let's explain these terms with detailed examples.

**1. Relationship Participation**

**Relationship Participation** describes whether the presence of an entity in a relationship is optional or mandatory.

**a. Optional Participation**

* In an **optional participation** relationship, one entity occurrence does **not require** a corresponding occurrence in another entity for the relationship to exist.
* This means that an entity can exist without being linked to another entity.

**Example**:

* Consider the relationship between **EMPLOYEE** and **PROJECT**:
  + An **EMPLOYEE** may be working on a **PROJECT** (or many projects), but it is also possible that an **EMPLOYEE** is not assigned to any **PROJECT** at all.
  + In this case, the participation of **EMPLOYEE** in the **PROJECT** relationship is **optional**.

**b. Mandatory Participation**

* In a **mandatory participation** relationship, one entity occurrence **requires** a corresponding occurrence in another entity for the relationship to exist.
* This means that the entity cannot exist in the database unless it is associated with another entity.

**Example**:

* Consider the relationship between **ORDER** and **ORDER\_ITEM**:
  + Each **ORDER\_ITEM** must be linked to a specific **ORDER**; otherwise, the **ORDER\_ITEM** cannot exist.
  + In this scenario, the **ORDER\_ITEM** entity's participation in the **ORDER** relationship is **mandatory**.

**2. Relationship Degree**

**Relationship Degree** refers to the number of entities involved in a relationship. It indicates how many participants (entities) are associated with that relationship. There are three common degrees of relationships: **Unary**, **Binary**, and **Ternary**.

**a. Unary Relationship (Degree 1)**

* A **unary relationship** (or recursive relationship) occurs when the association is maintained within a **single entity**.
* In this type of relationship, an entity is related to itself.

**Example**:

* **EMPLOYEE** entity with a **MANAGES** relationship:
  + An **EMPLOYEE** can manage other **EMPLOYEES**.
  + The **MANAGES** relationship links an **EMPLOYEE** to another **EMPLOYEE** within the same entity.

**b. Binary Relationship (Degree 2)**

* A **binary relationship** involves the association between **two different entities**.
* This is the most common degree of relationship in databases.

**Example**:

* **CUSTOMER** and **ORDER** entities:
  + A **CUSTOMER** can place an **ORDER**.
  + The relationship **PLACES** connects these two entities (**CUSTOMER** and **ORDER**), making it a binary relationship.

**c. Ternary Relationship (Degree 3)**

* A **ternary relationship** involves the association of **three different entities**.
* It represents a more complex relationship that involves multiple entities interacting together.

**Example**:

* Entities **SUPPLIER**, **PRODUCT**, and **WAREHOUSE**:
  + A **SUPPLIER** supplies a **PRODUCT** to a **WAREHOUSE**.
  + The ternary relationship describes the interaction between these three entities in a single association.

**Summary**

1. **Relationship Participation**:
   * **Optional Participation**: An entity can exist without being linked to another entity in the relationship (e.g., **EMPLOYEE** may or may not work on a **PROJECT**).
   * **Mandatory Participation**: An entity must be linked to another entity to exist in the relationship (e.g., **ORDER\_ITEM** must be associated with an **ORDER**).
2. **Relationship Degree**:
   * **Unary Relationship (Degree 1)**: Involves one entity (e.g., **EMPLOYEE** managing another **EMPLOYEE**).
   * **Binary Relationship (Degree 2)**: Involves two entities (e.g., **CUSTOMER** placing an **ORDER**).
   * **Ternary Relationship (Degree 3)**: Involves three entities (e.g., **SUPPLIER** supplying **PRODUCT** to **WAREHOUSE**).

These concepts help in defining the structure of the database more precisely by understanding the participation requirements of entities and the degree of their associations.

Let's dive into the concepts of **Recursive Relationships** and **Associative (Composite) Entities** and explain them with examples.

**1. Recursive Relationships**

**Definition**

* A **recursive relationship** is a relationship that exists between occurrences of the **same entity set**.
* This type of relationship is naturally found within **unary relationships**, where an entity relates to itself.

**Example**

* **EMPLOYEE** Entity with a **SUPERVISES** Relationship:
  + Suppose we have an entity called **EMPLOYEE**, and within the organization, some employees manage or supervise other employees.
  + In this scenario, we can have a **SUPERVISES** relationship that connects one **EMPLOYEE** to another **EMPLOYEE**.
  + The recursive relationship indicates that an employee can supervise other employees within the same entity.

**Diagram Representation**:

* In ER diagrams, a recursive relationship is usually represented with a line that loops back to the same entity.
* For example, if we have a rectangle labeled **EMPLOYEE**, the **SUPERVISES** relationship will connect back to the **EMPLOYEE** entity, forming a loop.

**Real-World Use Case**

* **Company Hierarchy**:
  + A company might have multiple levels of management. In this case, a manager (employee) might supervise other employees, who in turn might supervise more employees.
  + This kind of hierarchical structure is a classic example of a recursive relationship.

**2. Associative (Composite) Entities**

**Definition**

* An **associative entity** (also known as a **bridge entity** or **composite entity**) is used to **implement many-to-many (M) relationships** between entities.
* It serves as a link between two or more entities, converting the M

relationship into two one-to-many (1) relationships.

* The associative entity is typically composed of the **primary keys** from the entities it connects.
* It may also have additional attributes that describe aspects of the relationship itself.

**Example**

* **STUDENT** and **COURSE** Entities with an Associative Entity called **ENROLLMENT**:
  + Suppose we have a **M**

relationship where a **STUDENT** can enroll in many **COURSES**, and a **COURSE** can have many **STUDENTS**.

* + To handle this M relationship, we introduce an associative entity called **ENROLLMENT**.
  + The **ENROLLMENT** entity will include the primary keys of both **STUDENT** and **COURSE** as foreign keys.
  + The primary key of **ENROLLMENT** might be a composite key that includes **Student\_ID** and **Course\_ID**.

**Additional Attributes in the Associative Entity**

* The **ENROLLMENT** entity may also contain other attributes that are relevant to the relationship but not directly related to connecting the entities.
* For example:
  + **Enrollment\_Date**: The date when the student enrolled in the course.
  + **Grade**: The grade the student received in that specific course.

**Diagram Representation**:

* In ER diagrams, the associative entity is often represented as a rectangle with rounded corners.
* It is linked to the entities it connects with lines, converting the M

relationship into two 1

relationships.

**Real-World Use Case**

* **Online Shopping System**:
  + In an e-commerce platform, there could be a many-to-many relationship between **CUSTOMER** and **PRODUCT**.
  + A **CUSTOMER** can buy many **PRODUCTS**, and a **PRODUCT** can be bought by many **CUSTOMERS**.
  + The associative entity called **ORDER** or **ORDER\_DETAILS** can be used to track each purchase, linking **CUSTOMER** and **PRODUCT** entities.

**Summary**

* **Recursive Relationships**:
  + A recursive relationship is a self-referencing relationship within the same entity.
  + Example: An **EMPLOYEE** entity having a **SUPERVISES** relationship with itself to represent a hierarchical structure.
* **Associative (Composite) Entities**:
  + Associative entities are used to break down many-to-many (M

) relationships into two one-to-many (1

) relationships.

* + They contain the primary keys from the entities they connect and may also have additional attributes that describe the relationship.
  + Example: An **ENROLLMENT** entity that links **STUDENT** and **COURSE** entities and tracks information like **Enrollment\_Date** and **Grade**.

These concepts help in accurately modeling complex relationships in a database, ensuring the data structure aligns with real-world scenarios.

Developing an **Entity-Relationship Diagram (ERD)** is a structured process that involves several steps to accurately model the database according to the needs of the organization. Let's go through each step in this process with examples to illustrate how it works.

**Steps in Developing an ER Diagram**

1. **Create a Detailed Narrative of the Organization’s Description of Operations**
   * The first step in designing an ERD is to understand how the organization operates. You gather information by speaking to stakeholders, reviewing documents, and understanding business processes.
   * This narrative describes what the organization does, what data it handles, and how different pieces of information relate to each other.

**Example**:

* + Suppose we are designing a database for a university. The narrative might include details like:
    - "The university has multiple departments. Each department offers several courses. Students enroll in these courses, and professors teach them. Professors may teach in multiple departments, and each student is assigned to an academic advisor."

1. **Identify Business Rules Based on Description of Operations**
   * Business rules are specific rules that define how data should be created, stored, and maintained. They dictate relationships between different entities.
   * The rules are derived from the organization's narrative and serve as the foundation for creating the ERD.

**Example**:

* + From the university's description, some business rules might be:
    - A **Department** can offer many **Courses**.
    - A **Course** must belong to exactly one **Department**.
    - A **Student** can enroll in many **Courses**, and each **Course** can have many **Students** (M relationship).
    - A **Professor** can teach multiple **Courses**, but each **Course** is taught by only one **Professor**.

1. **Identify Main Entities and Relationships from Business Rules**
   * After understanding the business rules, you identify the main entities involved in the database and the relationships among them.
   * Entities are usually nouns (like **Student**, **Course**, **Professor**, **Department**), and relationships describe how these entities interact (like **enrolls in**, **teaches**, **belongs to**).

**Example**:

* + Identified entities from the university example:
    - **STUDENT**, **COURSE**, **PROFESSOR**, **DEPARTMENT**
  + Identified relationships:
    - **STUDENT** enrolls in **COURSE**
    - **PROFESSOR** teaches **COURSE**
    - **COURSE** belongs to **DEPARTMENT**

1. **Develop the Initial ER Diagram**
   * At this stage, you create the initial ER diagram using the identified entities and relationships. You sketch out how the entities are connected using lines to show relationships.
   * In the initial diagram, you focus on correctly representing the relationships between entities and their cardinality (e.g., one-to-many, many-to-many).

**Example**:

* + In the initial ERD for the university, you would draw rectangles for each entity (**STUDENT**, **COURSE**, **PROFESSOR**, **DEPARTMENT**) and connect them with lines indicating relationships, such as:
    - **STUDENT** ↔ **COURSE** (Many-to-Many, shown with a bridge entity like **ENROLLMENT**)
    - **PROFESSOR** → **COURSE** (One-to-Many)
    - **COURSE** → **DEPARTMENT** (Many-to-One)

1. **Identify Attributes and Primary Keys That Adequately Describe Entities**
   * In this step, you define attributes for each entity and determine the primary key (unique identifier) for each entity.
   * Attributes are properties or characteristics of entities. The primary key is a unique attribute that identifies each instance of the entity.

**Example**:

* + Attributes for the **STUDENT** entity might include:
    - **Student\_ID** (Primary Key)
    - **Name**
    - **Date\_of\_Birth**
    - **Email**
  + Attributes for the **COURSE** entity might include:
    - **Course\_ID** (Primary Key)
    - **Course\_Name**
    - **Credits**

1. **Revise and Review the ER Diagram**
   * The ERD design process is iterative. After creating the initial diagram and defining attributes, you review and refine the ERD based on feedback and a deeper understanding of the organization’s needs.
   * You may add, remove, or modify entities, attributes, and relationships to make sure the ERD accurately represents the data requirements.

**Example**:

* + After reviewing the initial ERD, you might decide to add an attribute like **Hire\_Date** to the **PROFESSOR** entity or introduce a new entity like **DEGREE\_PROGRAM** to better capture the university's structure.

**Full Example: Developing an ERD for a University**

Let's summarize the process with a complete example based on the university database scenario:

1. **Narrative**: The university has departments, courses, students, and professors. Each student enrolls in courses, which are taught by professors. Professors can belong to multiple departments.
2. **Business Rules**:
   * A department can have many courses, but each course belongs to one department.
   * Students can enroll in multiple courses, and each course can have multiple students.
   * Each course is taught by one professor, but a professor can teach multiple courses.
3. **Entities and Relationships**:
   * Entities: **STUDENT**, **COURSE**, **PROFESSOR**, **DEPARTMENT**, **ENROLLMENT**.
   * Relationships:
     + **STUDENT** enrolls in **COURSE** (M

, with **ENROLLMENT** as an associative entity).

* + - **COURSE** is taught by **PROFESSOR** (1

).

* + - **COURSE** belongs to **DEPARTMENT** (M:1).

1. **Initial ERD**:
   * Create rectangles for each entity and connect them with relationship lines:
     + **ENROLLMENT** entity links **STUDENT** and **COURSE**.
     + Arrow from **PROFESSOR** to **COURSE** indicating one professor can teach many courses.
2. **Attributes and Primary Keys**:
   * **STUDENT**: **Student\_ID** (PK), **Name**, **Email**.
   * **COURSE**: **Course\_ID** (PK), **Course\_Name**, **Credits**.
   * **ENROLLMENT**: **Student\_ID**, **Course\_ID** (Composite PK), **Enrollment\_Date**.
3. **Revise and Review**:
   * After feedback, refine relationships, add attributes if needed, and ensure all business rules are accurately represented.

**Summary**

Developing an ERD is an **iterative** process that involves the following steps:

1. Create a narrative of the organization's operations.
2. Identify business rules based on this description.
3. Identify entities and relationships from the business rules.
4. Develop an initial ERD.
5. Identify attributes and primary keys for the entities.
6. Revise and review the ERD based on feedback and refinement.

This step-by-step process ensures that the database design accurately reflects the data requirements and business processes of the organization.