ERD

What is an ER Diagram?🔎🔎

In simple terms, an **entity-relationship diagram (ERD)** is a way of illustrating how entities (like people, objects, or concepts) are related to each other in a system. It shows entities, their attributes, and the relationships between them, providing a clear picture of the system's structure.

Why ERD?👀

ER diagrams are essential in **database design**. Just like blueprints are crucial for constructing buildings🏢, ERDs serve as blueprints for databases. Imagine😇 how helpful visual aids are when someone is telling a story — ER diagrams play a similar role by providing a visual representation of how data is organized and connected. They help developers and stakeholders better understand the structure of the system and how different pieces of data interact.

Scenario: Designing a Hospital Management System (HMS)

1. Visual Representation of Data

Example:
In the HMS, an ER diagram visually displays entities like **patients**, **doctors**, **nurses**, **rooms**, and **appointments.** This visualization helps all stakeholders, including administrators and developers, understand how these entities are connected (e.g., a patient is admitted to a room, a doctor schedules appointments).

2. Improved Database Design

Example:
By using an ER diagram, you can design the HMS database to avoid redundancy. For instance, the **doctor** entity stores information only once, and **appointments** can reference the doctor using a foreign key, avoiding duplication of doctor details in each appointment record.

3. Easy to Identify Relationships

Example:
The ER diagram clearly shows the relationships between **patients** and **appointments.** It illustrates that a patient can have multiple appointments, but each appointment is linked to one doctor. This helps ensure that the data reflects real-world interactions accurately.

4. Prevents design errors

Example:
In the early design phase, the ER diagram for the HMS can help spot errors like missing relationships. For example, without an ER diagram, you might forget to link **Rooms** to **Patients**, leading to incomplete data about room assignments. The diagram ensures no entities or relationships are left out.

5. Facilitates Communication

Example:

When discussing the HMS with stakeholders, the ER diagram makes it easier to explain how data flows through the system. For instance, the diagram can clarify how a **patient** books an **appointment**, how the **doctor** is assigned, and how the **room** is linked during admission.

6. Foundation for Logical and Physical Database Design

Example:

The ER diagram serves as the blueprint for building the actual HMS database. It guides the developers in creating tables for **patients**, **doctors**, **appointments**, etc., as well as defining primary keys (e.g., **PatientID**) and foreign keys to maintain relationships between the entities.

7. Ensures Data Integrity

Example:

The ER diagram ensures that key constraints are enforced in the HMS database. For instance, the **DoctorID** in the **Appointments** table must reference a valid **doctor** in the **Doctors** table, preventing data inconsistencies like booking appointments with non-existent doctors.

8. Scalability and Flexibility

Example:

As the hospital grows, new entities like **pharmacies** or **insurance providers** can be added to the existing ER diagram. This flexibility allows the HMS to scale without needing to overhaul the entire database structure, ensuring it remains adaptable for future needs.

Key components of ER diagrams

Entities

An entity is any object in the system we want to represent, like a person, place, or concept.

1. Entity Types:

- **Physical Entities**: These represent real-world objects, such as **students**, **products**, or **employees.** For example, in a school database, **a student** is an entity that stores information like name, student ID, and date of birth.
- **Conceptual Entities**: These represent abstract concepts, such as **orders**, **courses**, or **projects.** In a company database, **Project** is an entity that tracks project name, start date, and assigned team members.

2. Entity Set:

An **entity set** is a collection of all instances of a particular entity in the database. For example, if the entity is **Student**, the entity set would be the collection of all students stored in the system.

3. Strong vs. Weak Entities:

- **Strong Entity**: A strong entity can exist independently and is not reliant on any other entity. For example, an **Employee** entity can exist on its own.

- **Weak Entity**: A weak entity depends on another entity for its existence and does not have a unique key by itself. For example, an **OrderItem** in an online store depends on an **Order** entity and cannot exist without

4. Entity keys:

Entity keys are attributes or sets of attributes that uniquely identify entities within an entity set in a database. They play a crucial role in maintaining the integrity and structure of the database. Keys help distinguish between different instances of an entity, ensuring that every record can be uniquely identified.

Types of Entity Keys

1. **Primary Key (PK)**:
    - A **primary key** is the main attribute (or a combination of attributes) that uniquely identifies each record in an entity.
    - **Example**: In a **student** entity, StudentID could be the primary key because it uniquely identifies every student in the system.
    - **Characteristics**:
        - It must be unique for every record.
        - It cannot contain null values (must always have a value).
        - Each entity should have only one primary key.
2. **Candidate Key**:
    - A **candidate key** is any attribute (or set of attributes) that could potentially act as a primary key. In a table, there may be multiple candidate keys, but only one is chosen as the primary key.
    - **Example**: In a **Student** entity, both StudentID and Email might uniquely identify a student, making them both candidate keys. However, StudentID would be chosen as the primary key, while Email remains a candidate key.
    - **Characteristics**:
        - It must be unique and not null.
        - An entity can have more than one candidate key.
3. **Composite Key**:
    - A **composite key** is a key made up of more than one attribute to uniquely identify a record when a single attribute is not enough.
    - **Example**: In an **OrderItems** entity, a composite key may consist of OrderID and ProductID because each combination of these two attributes is unique.
    - **Characteristics**:
        - It consists of two or more attributes.
        - The combination of these attributes must be unique across all records.
4. **Foreign Key (FK)**:
    - A **foreign key** is an attribute in one entity that links to the **primary key** in another entity. It establishes a relationship between two entities.
    - **Example**: In an **Orders** entity, CustomerID could be a foreign key that references the CustomerID primary key in the **Customer** entity. This shows which customer placed the order.
    - **Characteristics**:
        - It creates relationships between entities.
        - It ensures referential integrity, meaning the foreign key value must exist in the related entity.
5. **Super Key**:

   - A **super key** is a set of attributes (one or more) that can uniquely identify a record in an entity. A super key may include additional attributes that are not necessary for uniqueness.
   - **Example**: In the **Employee** entity, the set {EmployeeID, PhoneNumber} can be a super key because this combination can still uniquely identify an employee, even though EmployeeID alone is sufficient.
   - **Characteristics**:
      - It may contain extra attributes beyond what is needed for uniqueness.

6. **Alternate Key**:
   - An **alternate key** is any candidate key that is not chosen as the primary key.
   - **Example**: In the **Student** entity, if StudentID is the primary key, the Email can be an alternate key, since it could also have been chosen as the primary key.
   - **Characteristics**:
      - An entity can have multiple alternate keys.
7. **Unique Key**:
   - A **unique key** is similar to a primary key in that it ensures uniqueness of the records, but unlike the primary key, it can contain null values.
   - **Example**: In a **User** entity, the Email can be defined as a unique key to ensure that no two users share the same email address, but it may allow null values if email is optional.
   - **Characteristics**:
      - It guarantees uniqueness but allows null values.

Attributes

Attributes are properties or characteristics of an entity. Each entity can have multiple attributes.
Example Attributes for Entity 'Customer':

- Name
- Email
- Addres

1. Simple Attribute

- **Definition**: An attribute that cannot be further divided into smaller components.
- **Example**: FirstName in a **Person** entity, where FirstName is a single value and cannot be broken down further.

2. Composite Attribute

- **Definition**: An attribute that can be divided into smaller sub-parts, each representing a more basic attribute with a meaningful value.
- **Example**: FullName in a **Person** entity can be divided into FirstName and LastName.

3. Derived Attribute

- **Definition**: An attribute whose value can be derived from other attributes or entities. It is not stored directly in the database but computed as needed.
- **Example**: Age in a **Person** entity can be derived from the DateOfBirth attribute. It is calculated as the difference between the current date and the DateOfBirth.

4. Multi-Valued Attribute

- **Definition**: An attribute that can have multiple values for a single entity instance.
- **Example**: PhoneNumbers in a **Person** entity, where a person may have multiple phone numbers. This requires a separate table or a complex structure to store the multiple values.

5. Key Attribute

- **Definition**: An attribute that uniquely identifies an instance of an entity. Key attributes are used to ensure that each entity instance can be uniquely identified.
- **Example**: StudentID in a **Student** entity. Each student has a unique StudentID that distinguishes them from others.

6. Foreign Key Attribute

- **Definition**: An attribute that refers to the primary key of another entity. It is used to establish relationships between entities.
- **Example**: DepartmentID in an **Employee** entity, which refers to the primary key DepartmentID in the **Department** entity.

7. Relationship Attribute

- **Definition**: Attributes that describe the relationship between two or more entities.
- **Example**: EnrollmentDate and Grade in an **Enrollment** relationship between **Student** and **Course**.

8. Complex Attribute

- **Definition**: An attribute that is a combination of multiple simple or composite attributes. It is not easily divisible into individual attributes.
- **Example**: An address might be considered a complex attribute if it includes Street, City, State, and ZipCode all combined into a single attribute.

Relationships

These represent how entities are related to each other.

Types of Relationships:

  - One-to-One (1:1)
  - One-to-Many (1:M)
  - Many-to-Many (M:1)

Example:

"In an ER Diagram, **entities** are the objects we want to represent—like customers, products, or orders. These entities have **attributes** (properties) and are connected through **relationships** like one-to-one or many-to-many.

Cardinality in Relationships

Cardinality defines the number of instances in one entity that can be related to instances in another entity.

Types of Cardinality:

    - **1:1:** A student has one student ID.
    - **1:N:** A customer can place many orders.
    - **M:N:** A product can belong to many categories, and categories can have many products.

Participation constraints

In simple terms, they specify whether the participation of an entity in a relationship is mandatory or optional.

There are two types of participation constraints:

Total Participation (Mandatory Participation)

- **Definition**: Every instance of the entity must participate in the relationship.
- **Example**: In a **student-enrolls-Course** relationship, if every student **must enroll** in at least one course, then the **student** entity has total participation in the **enrolls** relationship. This means that each student must have at least one associated course.
- **Notation**: In ER diagrams, total participation is usually represented by a double line connecting the entity to the relationship.

Partial Participation (Optional Participation)

- **Definition**: Only some instances of the entity may participate in the relationship, meaning participation is optional.
- **Example**: In an **employee-manages-Department** relationship, not every employee needs to manage a department. Some employees might not have any management responsibilities. In this case, the **employee** entity has partial participation in the **managed** relationship.
- **Notation**: In ER diagrams, partial participation is shown by a single line connecting the entity to the relationship.

Degree of Relationships

The degree of a relationship in an entity-relationship diagram (ERD) refers to the number of entities involved in a relationship. It defines how many entity types are participating in a relationship set.

Types of relationship degrees

1. **Unary Relationship (Degree 1):**
    - In a unary relationship, an entity is related to itself. It involves only one entity type.
    - Example: In an organizational structure, the employee entity may have a relationship "supervises" with itself, where an employee supervises other employees. This is a unary relationship because it involves only the employee entity.
2. **Binary Relationship (Degree 2):**
    - A binary relationship involves two entities.
    - Example: In a university system, the relationship between student and course is binary because it involves two entities—students enroll in courses.
3. **Ternary Relationship (Degree 3):**
    - A ternary relationship involves three entities.
    - Example: In a hospital, a ternary relationship could be represented as Doctor, Patient, and Treatment. A doctor provides a specific treatment to a patient, linking all three entities.
4. **N-ary Relationship (Degree N):**

  - When a relationship involves more than three entities, it is called an n-ary relationship. It generalizes the concept of ternary relationships to include n entities.
  - Example: In a project management system, a relationship between project, employee, role, and department can be considered an n-ary relationship because it involves four entities.

Migrate Attributes

In entity-relationship (ER) diagrams, sometimes it's efficient to **migrate (or move) attributes** of a relationship into one of the participating entities, rather than keeping them in the relationship. This can be done especially in **1:1** (one-to-one) and **1**

Here's how the attributes can be migrated in each type of relationship:

1. One-to-One (1:1) Relationship:

- In a **1:1 relationship**, each instance of one entity is related to at most one instance of another entity, and vice versa.
- Since there's a unique relationship between the two entities, the attributes of the relationship can often be migrated into **either** of the entities.

Example:

- **Entities**: **Person** and **Passport**
- **Relationship**: **Owns** (A person owns a passport)
- **Attribute of the Relationship**: IssueDate

Migration:

The attribute IssueDate can be moved to either the **Person** entity or the **Passport** entity because each person has exactly one passport and vice versa. Typically, it would be moved to the entity that is more relevant for that attribute (in this case, **Passport**).

2. One-to-Many (1) Relationship:

- In a **1:M relationship**, one instance of an entity can be related to multiple instances of another entity, but the reverse is not true.
- The attributes of the relationship can be migrated to the **"many"** side of the relationship because multiple instances of the entity on that side can have unique relationship attributes.

Example:

- **Entities**: **Department** and **Employee**
- **Relationship**: **WorksIn** (An employee works in one department, but a department has many employees)
- **Attribute of the Relationship**: HireDate

Migration:

In this case, the relationship attribute HireDate can be moved to the **Employee** entity. Since each employee works in one department and has a specific hire date, it makes sense to store HireDate with the **Employee** entity.

3. Many-to-Many (M) Relationship:

- In a **many-to-many relationship**, multiple instances of one entity can be associated with multiple instances of another entity.
- Because both sides of the relationship have multiple associations, attributes related to the relationship are typically placed in a junction table or a bridge entity that connects the two entities.

Example:

- **Entities**: **Student** and **Course**
- **Relationship**: **EnrolledIn** (A student can enroll in multiple courses, and a course can have multiple students)
- **Attribute of the Relationship**: EnrollmentDate, Grade

Migration:

To handle the attributes EnrollmentDate and Grade, you would typically create a new entity (junction table) called **Enrollment** that connects **Student** and **Course**. This entity will include the attributes of the relationship.

- **Junction Table**: **Enrollment**
  - **Attributes**: EnrollmentDate, Grade
  - **Foreign Keys**: StudentID, CourseID

4. Many-to-One (N:1) Relationship:

- In a **many-to-one relationship**, multiple instances of one entity are related to a single instance of another entity.
- Since there is only one instance on one side of the relationship, attributes related to the relationship are generally moved to the "many" side of the relationship.

Example:

- **Entities**: **Employee** and **Department**
- **Relationship**: **BelongsTo** (Many employees work in one department)
- **Attribute of the Relationship**: DateHired

Migration:

The attribute DateHired, which is specific to the employee’s association with a department, should be included in the **Employee** entity.

- **Employee Entity**:
  - **Attributes**: EmployeeID, Name, DateHired, DepartmentID (Foreign Key)