Different between Entity and Domain constraints

Domain Constraints:

What are they? They are rules that define the allowed values that can be entered into a specific column in a database table. In other words, they define the "domain" of correct values for that column.

Their function: They ensure that the data entered into a column follows a specific type (like number, text, date) and is within a defined range (like age being greater than 18, or email having a specific format).

Examples:

- Specifying that the "Age" column must be a positive whole number.
- Specifying that the "Email" column must contain the "@" symbol and a dot ".".
- Specifying that the "Marital Status" column can only take the values "Single," "Married," "Divorced," "Widowed."
- Using the NOT NULL constraint to prevent empty values from being entered in a required column.
- Using the CHECK constraint to enforce specific conditions on the entered values.

Focus: Domain constraints focus on the validity of individual data values within each column.

Entity Constraints:

What are they? They are rules aimed at ensuring the integrity of the entities within a table. Here, "entity" refers to each row (record) in the table that represents a unique thing.

Their function: They ensure that each row in the table can be uniquely identified and does not contain empty values in the columns that define its identity.

Examples:

- Primary Key: This constraint ensures that one column or a group of columns contains unique and non-empty values for each row in the table. This guarantees no duplicate records and the ability to access each record directly.
- Unique Constraint: This constraint ensures that the values in a specific column are unique across all rows in the table, but it allows one null (empty) value (unlike the primary key, which does not allow any null values).
- NOT NULL Constraint: Although it can be used as a domain constraint, it plays a crucial role in entity integrity when applied to columns that form part of the primary key or important identifying columns for the entity.

Focus: Entity constraints focus on the rows (records) as unique units within the table.

The Core Difference:

- Domain Constraints: Deal with the validity of data values entered into each column individually.
- Entity Constraints: Deal with ensuring the integrity and uniqueness of each row (entity) within the table as a whole.

Insert values to tables priority

- **Ensuring Data Integrity and Consistency: Prioritizing Table Population Based on Relationships (Foreign Keys)**
- To guarantee the safety and interconnectedness of data in a database when entering values, the priority of populating tables primarily depends on the relationships between them, specifically **Foreign Keys**.
- Here are the steps and principles for determining the priority of table population:
- **1. Understanding Table Relationships:**
- **Identifying Parent Tables:** These tables do not rely on other tables for their definition. They often contain the primary keys that will be used as foreign keys in other tables.
- **Example:** A "Customers" table or a "Products" table.
- **Identifying Child Tables:** These tables depend on other tables through foreign keys. They contain columns that refer to existing records in the parent tables.
- **Example:** An "Orders" table that contains a foreign key referencing the "CustomerID" in the "Customers" table.
- **2. Determining Population Priority:**
- **Start with Parent Tables:** The parent tables must be populated with data first. The reason is that child tables rely on the existence of records in the parent tables to refer to them via foreign keys. If you try to enter data into a child table before the related data exists in the parent table, you will encounter a **Foreign Key Constraint Violation** because the value you are trying to reference won't be found.
- **Then Populate Child Tables:** After ensuring that the basic data exists in the parent tables, you can begin populating the child tables. When entering data into a child table, you must ensure that the foreign key values you enter match existing primary key values in the related parent tables.
- **Illustrative Example:**
- Let's assume you have two tables:
- **Customers:** Contains CustomerID (Primary Key), CustomerName, Email, etc.
- **Orders:** Contains OrderID (Primary Key), CustomerID (Foreign Key referencing CustomerID in the Customers table), OrderDate, etc.

- To populate these tables correctly:
- **Start by populating the "Customers" table:** Enter customer data first (customer IDs, names, and other information).
- **Then populate the "Orders" table:** When entering order data, ensure that the CustomerID value entered in the "Orders" table matches one of the CustomerID values that already exist in the "Customers" table.
- **Summary of Prioritization Rules:**
- Tables that do not contain foreign keys (or contain foreign keys that refer to themselves - self-referencing relationships) can be populated first.
- Tables that contain foreign keys must be populated after the tables containing the primary keys they refer to have been populated.
- **Tools and Techniques to Aid Population:**
- **Database Diagrams:** Help visualize the relationships between tables and identify dependencies.
- **Ordering Insertion Operations in Scripts:** If you are using SQL scripts to enter data, order the INSERT INTO operations so that data is inserted into parent tables first.
- **Using Database Management Tools (e.g., SQL Server Management Studio, MySQL Workbench):** These tools provide visual interfaces that help manage data and understand relationships.
- **Verifying Constraints:** Ensure that foreign key constraints are enabled in the database to prevent the entry of inconsistent data.

Case: In mapping Can not a Foreign Key Reference a Partial Key

In a **mapping diagram** (such as in **Entity-Relationship (ER) modeling** or **relational schema design**), whether a **foreign key** can reference a **partial key** depends on the context of the table being referenced.

Key Concepts:

- **Primary Key (PK):** A unique identifier for a row in a table.
- **Partial Key:** An attribute (or set of attributes) that **can uniquely identify a weak entity** but **only in the context of its owning entity**.
- **Foreign Key (FK):** An attribute (or set) that refers to the primary key of another table to enforce referential integrity.

📌 Can a Foreign Key Reference a Partial Key?

Not directly. A **foreign key must reference a candidate key (usually the primary key)** of another table. A **partial key alone is not sufficient** as a foreign key target because:

- A **partial key is not unique** on its own.
- It needs to be **combined with the primary key of the owner entity** to form a **composite primary key** in the weak entity.

✅ **However, in Practice:**

If a weak entity is transformed into a relation (table), then:

- The **weak entity's table** will have a **composite primary key** = {Primary key of owner + Partial key}.
- Another table can define a **foreign key referencing this composite key**.

Example:

Let's say we have:

- **Owner Table:** Building(building_id PK)
- **Weak Entity:** Room(building_id FK, room_number)
  Here, room_number is a **partial key**, and Room's **composite PK** is (building_id, room_number)

Now, suppose we have another table:

- Reservation(building_id, room_number, date, ...)

Then Reservation.building_id and room_number can be a **foreign key referencing** Room(building_id, room_number), which includes the partial key — **but not the partial key alone**.

✅ **Summary:**

- ❌ A foreign key **cannot reference a partial key alone**.
- ✅ A foreign key **can reference a composite key** that includes the partial key **plus** the owner's primary key.

Case: it is possible to make ternary relationship between 2 stronge entity and one week entity

It is absolutely possible (and sometimes necessary) to create a **ternary relationship** between **two strong entities and one weak entity** — but you need to carefully manage the **identification** and **referential integrity** of the weak entity.

✅ **Let's break this down:**

Definitions:

- **Strong entity**: Has its own primary key.
- **Weak entity**: Cannot be uniquely identified by its own attributes; relies on related strong entity/entities.
- **Ternary relationship**: A relationship involving **three entities**.

🧩 **Scenario Example:**

Let's imagine this scenario:

- **Doctor(doctor_id)** – strong entity
- **Patient(patient_id)** – strong entity
- **Treatment(treatment_id)** – weak entity

A treatment is defined **only when a specific doctor treats a specific patient**. So, the weak entity Treatment depends on both Doctor and Patient.

ER Design:

scss

CopyEdit

Doctor(doctor_id PK)

Patient(patient_id PK)

Treatment(

doctor_id FK,

patient_id FK,

treatment_number, -- Partial key (e.g., 1st treatment, 2nd treatment)

description,

PRIMARY KEY (doctor_id, patient_id, treatment_number)

)

- Here, Treatment is a **weak entity** that participates in a **ternary relationship** with Doctor and Patient.
- treatment_number is the **partial key**.
- The **composite key** of Treatment is (doctor_id, patient_id, treatment_number).

✅ **Summary:**

Feature	Allowed?	Notes
Weak entity in ternary relationship	✅ Yes	It depends on **two (or more)** strong entities.
Weak entity has partial key	✅ Yes	Identifies the weak entity within the relationship context.
Foreign keys from strong entities used in weak entity	✅ Yes	Required to ensure identification.

Case : Not need to add fourth table if we have ternary relationship between 2 strong entity and one week entity

To represent this ternary relationship in the mapping schema, we do **not** need to create a separate (fourth) relationship table, because the **weak entity itself acts as the relationship**. It already includes the foreign keys referencing the two strong entities and contains the identifying attributes. Therefore, the weak entity table captures both the relationship and the data.

💡 Explanation:

In a typical ternary relationship between **three strong entities**, you usually need a separate relationship table to capture the association (since no one entity can "own" the relationship).
But when **one of the entities is weak**, the weak entity:

- Depends on the other two (strong) entities.
- Has its own attributes.
- Serves as a **bridge** or **relationship holder**.

So, **no need for an extra fourth table** — the **weak entity table is enough**.

🧩 Example Mapping (ER to Relational Schema):

Entities:

- Doctor(doctor_id PK)
- Patient(patient_id PK)
- Treatment(doctor_id FK, patient_id FK, treatment_number, description, PRIMARY KEY(doctor_id, patient_id, treatment_number))

No need for a table like Doctor_Patient_Treatment(…) — because Treatment **already serves that role**.

---

parent table عادي فيه جدول يكون بدون المفتاح الرئيسي , مادام لا يوجد جدول يعتبر هذا الجدول ك