**When should store a derived attribute**

**Should You Store a Derived Attribute?**

**A *derived attribute* is an attribute that can be calculated from other stored data, such as calculating Age from DateOfBirth.**

**While derived attributes are usually not stored directly in databases, there are cases where storing them can be beneficial.**

**Yes – Store It When:**

* **Performance is a concern: Calculating the value repeatedly may slow down the system.**
* **The value is used frequently: Storing avoids recalculating in every query.**
* **The calculation is costly or complex: Especially when data is large or involves multiple joins or functions.**

**When to Store a Derived Attribute**

1. **Complex or Time-Consuming Calculation  
   *Example:* Generating financial reports or statistical summaries.**
2. **Frequent Usage in Queries  
   Improves system efficiency and response time.**
3. **Infrequent Change of Value  
   *Example:* Age — it can be stored and updated once per year.**
4. **Auditing or Legal Requirements  
   *Example:* Storing a discounted price at the time of purchase, even if product prices later change.**

**When NOT to Store a Derived Attribute**

1. **Easily Calculated from Existing Data  
   No need to store if the value is simple to derive on demand.**
2. **Frequent Changes in Value  
   Storing can lead to outdated or inconsistent data.**
3. **Risk of Data Inconsistency  
   If the source data changes and the derived attribute isn’t updated, this can cause incorrect results.**

**Conclusion**

**Store a derived attribute only if it improves performance, supports business needs (e.g., auditing), or avoids expensive computations. Otherwise, derive it at runtime to keep your data clean and consistent.**

**Mapping unary M:N**

**What is a Unary M:N Relationship?**

A unary relationship (also called a recursive relationship) occurs when an entity is related to itself.

A many-to-many (M:N) unary relationship means that:

Each instance of the entity can be related to many other instances of the same entity, and vice versa.

**Example Scenario**

**Consider an entity Employee, and a relationship called Mentors, where:**

* **An employee can mentor many other employees.**
* **An employee can also be mentored by many others.**

**This forms a unary M:N relationship on the Employee entity.**

**Mapping Unary M:N to a Relational Model**

**To represent this relationship in a relational database, follow these steps:**

**1. Create the Main Table**

**2. Create a Relationship (Associative) Table**

**This new table will represent the self-referencing many-to-many relationship.**

**Notes**

* **Both MentorID and MenteeID reference the same primary key in the Employee table.**
* **If the relationship has attributes (e.g., StartDate, Status), you can add them to the Mentorship table.**
* **Data constraints may be added to prevent invalid relationships (e.g., an employee mentoring themselves).**

**Another example:**

A diagram of a function

AI-generated content may be incorrect.

A diagram of a component

AI-generated content may be incorrect.