3. Entities: Objects with Identity

An entity is something that remains the same thing over time, even when its data changes. This is the simplest useful definition. A customer can change email. A machine can change status. A unit can change power output. The values change, but the thing is still the same thing.

Most developers already use entities because most database tables have primary keys. But DDD asks a slightly deeper question: does this object have a life of its own in the domain, or is it only a collection of values? If it has identity and a lifecycle, it is probably an entity.

PLAIN EXPLANATION

An entity is not important because it has many properties. It is important because the system must recognize it as the same thing across time. Identity is the point.

A simple business example is Customer. The customer may change address, phone, email, or status, but the same CustomerId tells the system that this is still the same customer. Equality is not based on all current values. Equality is based on identity.

```
public sealed class Customer
{
    public int CustomerId { get; private set; }
    public string Name { get; private set; } = null!;
    public string Email { get; private set; } = null!;

    public void ChangeEmail(string newEmail)
    {
        if (string.IsNullOrWhiteSpace(newEmail))
            throw new InvalidOperationException("Email is required.");

        Email = newEmail;
    }
}
```

The object is not just a bag of setters. It owns a small rule: the email cannot be empty. In richer models, entities protect more important rules, but the principle is the same. An entity should not only hold data; it should protect the meaning of its own change.

Candidate	Entity?	Why
Unit	Yes	It has stable identity and many changing states: power, status, alarms, twin state, and history.
Signal	Yes	It has a stable tag and is referenced by readings, alarms, and twin bindings.
AlarmEvent	Yes	It has lifecycle: raised, acknowledged, cleared, grouped into a flood, used as evidence.
RootCauseCase	Yes	It has an investigation lifecycle: opened, enriched, hypotheses tested, verdict recorded.
Temperature 323 C	No	This is a value. It matters by value and unit, not by identity.

NEXUS-1 EXAMPLE

The same UnitId can appear in ReactorFleet, AlarmManagement, RootCause, DigitalTwin, ReinforcementLearning, Audit, and Reporting. That does not mean all those contexts own the Unit. ReactorFleet owns the physical unit identity. Other contexts reference it.

This distinction is very important. A context may reference an entity owned by another context without becoming the owner of that entity. RootCause may investigate Unit NX1-U1, but RootCause does not own the definition of NX1-U1. It borrows the identity through a passport such as UnitId.

```
public sealed class Unit
{
    public int UnitId { get; private set; }
    public string Code { get; private set; } = null!; // NX1-U1
    public string Name { get; private set; } = null!;
    public UnitStatus Status { get; private set; }

    public void MarkOnline()
    {
        if (Status == UnitStatus.Decommissioned)
            throw new InvalidOperationException(
                "A decommissioned unit cannot be marked online.");

        Status = UnitStatus.Online;
    }
}
```

The example is intentionally small. The important point is that the entity owns a rule about its own lifecycle. The rule is not hidden in a controller. The controller may request the action, but the entity protects the meaning of the action.

COMMON MISTAKE

A common mistake is to call every EF Core class an entity in the DDD sense. EF Core uses the word entity for mapped classes. DDD uses the word entity for a domain concept with identity. Many mapped classes are only lookup rows, history rows, or value-like records. Do not confuse ORM vocabulary with domain vocabulary.

HONEST BOUNDARY

Not every NEXUS-1 table deserves rich entity behavior. Some tables are reference data, audit shadows, read models, or generated lookup tables. DDD does not require making every row into a behavioral object. It requires knowing which concepts own important rules.