# Difference between an entity and an aggregate in domain driven design

Asked 4 years, 6 months ago Active 1 month ago Viewed 10k times

11

9

Please what is the main difference between entities and aggregate roots in domain driven design. For example in entity framework, what is the use of aggregates if I can ensure data integrity entities?

c# design-patterns domain-driven-design

asked Sep 2 '15 at 12:57

D_Edet

191 ● 1 ● 4 ● 11

## 3 Answers

Active Oldest Votes

42

From domain driven design perspective `DbContext` is the implementation of UnitOfWork and a `DbSet<T>` is the implementation of a repository.

This is the point where DDD and EntityFramework contrast. DDD suggests to have a Repository per aggregate root but EntityFramework creates one per Entity.

So, what is an aggregate root?

Assume that we have a social network and have **entities** like Post, Like, Comment, Tag. (I believe you can imagine the relations

edited Jan 7 '16 at 5:38

answered Sep 2 '15 at 13:43

Mehmet Ataş
9,008 ● 5 ● 38 ● 65

1 Good answer and definition, a much more functional definition rather than my theory definition. – Greg Sep 2 '15 at 13:44

4 'DDD suggests to have a Repository per aggregate...' an excellent, and under-utilised concept, IMHO. – David Osborne Sep 2 '15 at 14:41

@Mehmet Nice description. Exactly the picture I had of aggregate roots. I thought this is possible with database table entities generated using entity framework, with regards to foreign keys and primary keys. Would it be wise to let entity framework generate entities for a domain driven design project? or manually build one?.....Finally how can I persist aggregate roots to a database? – D_Edet Sep 2 '15 at 14:42

5 "For instance, Like or Comment cannot live without a Post" That's a very naïve way of composing aggregate boundaries. Following this thinking will lead to large cluster aggregates, even though it's completely unnecessary. Do you really need to load all comments made on a Post in memory just to add a new one? What invariant are you trying to protect by doing so? You may need this in some domains, but that design would be wrong for most blogging applications. – plalx Sep 2 '15 at 15:25

1 This is very late though...but this article helped my understanding further. [link] (codeproject.com/Articles/1020932/…) – D_Edet Sep 9 '15 at 14:06

14

The definition is fairly straight forward:

- Aggregate: Basically a cluster of objects, that create a clear reference to the root aggregate, to so when you reference the root, you can ensure integrity of the aggregates as a whole.

Aggregate is a pattern in Domain-Driven Design. A DDD aggregate is a cluster of domain objects that can be treated as a single unit. An example may be an order and its line-items, these will be separate objects, but it's useful to treat the order (together with its line items) as a single aggregate.

An aggregate will have one of its component objects be the aggregate root. Any references from outside the aggregate should only go to the aggregate root. The root can thus ensure the integrity of the aggregate as a whole.

Aggregates are the basic element of transfer of data storage - you request to load or save whole aggregates. Transactions should not cross aggregate boundaries.

DDD Aggregates are sometimes confused with collection classes (lists, maps, etc). DDD aggregates are domain concepts (order

- Entity: In a data model context, describes the structure of data regardless of the stored form.

> The EDM addresses the challenges that arise from having data stored in many forms. For example, consider a business that stores data in relational databases, text files, XML files, spreadsheets, and reports. This presents significant challenges in data modeling, application design, and data access. When designing a data-oriented application, the challenge is to write efficient and maintainable code without sacrificing efficient data access, storage, and scalability. When data has a relational structure, data access, storage, and scalability are very efficient, but writing efficient and maintainable code becomes more difficult. When data has an object structure, the trade-offs are reversed: Writing efficient and maintainable code comes at the cost of efficient data access, storage, and scalability. Even if the right balance between these trade-offs can be found, new challenges arise when data is moved from one form to another. The Entity Data Model addresses these challenges by describing the structure of data in terms of entities and relationships that are independent of any storage schema. This makes the stored form of data irrelevant to application design and development. And, because entities and relationships describe the structure of data as it is used in an application (not its stored form), they can evolve as an application evolves.

The definition may vary, those are defined by Martin Fowler and Microsoft. Hopefully that clarifies the difference though.

answered Sep 2 '15 at 13:21

Greg
**10.2k** ● 2 ● 37 ● 72

2 The definition's for both are sadly used interchangeably, which can be confusing but that is what each actually means. – Greg Sep 2 '15 at 13:43

0

**Entity**: An object defined primarily by its identity is called an ENTITY that has significance (e.g. Customer) in the sales system is an Entity and can change over time.

**Value Object**: Value Objects are objects that are known only by their properties and values. For example, "Customer Address" can be designed as a Value Object. Value Objects can be assigned to different Entities and are usually implemented as Immutable (e.g. date, address)

**Aggregate**: a collection of entities or value objects that are related to each other through a Aggregate Root object

**Aggregate Root**: Each Aggregate has a root and a boundary, **Aggregate Root** owns an Aggregate and serves as a gateway for all

1,385 ● 5 ● 16