What difference does .AsNoTracking() make?

189

47

I have a question regarding the `.AsNoTracking()` extension, as this is all quite new and quite confusing.

I'm using a per-request context for a website.

A lot of my entities don't change so don't need to be tracked, but I have the following scenario where I'm unsure of what's going to the database, or even whether it makes a difference in this case.

This example is what I'm currently doing:

```
context.Set<User>().AsNoTracking()
// Step 1) Get user
context.Set<User>()
// Step 2) Update user
```

This is the same as above but removing the `.AsNoTracking()` from Step 1:

```
context.Set<User>();
// Step 1) Get user
context.Set<User>()
// Step 2) Update user
```

The Steps 1 & 2 use the same context but occur at different times. What I can't work out is whether there is any difference. As Step 2 is an update I'm guessing both will hit the database twice anyway.

Can anyone tell me what the difference is?

c# .net entity-framework entity-framework-4.3

edited Aug 31 '12 at 8:43 asked Aug 31 '12 at 8:35

154

The difference is that in the first case the retrieved user is not tracked by the context so when you are going to save the user back to database you must attach it and set correctly state of the user so that EF knows that it should update existing user instead of inserting a new one. In the second case you don't need to do that if you load and save the user with the same context instance because the tracking mechanism handles that for you.

edited Aug 17 '15 at 15:08
Jess
13.7k 10 86 120

answered Aug 31 '12 at 8:39
Ladislav Mrnka
330k 55 620 644

Can we get the same benefits for anonymous classes in select query, such as context.Users.Select(u=> new { Name = u.Name })? Thanks. – Dilhan Jayathilake Feb 6 '17 at 23:33

2 @DilhanJayathilake: Anonymous classes don't represent the entity itself so they don't have tracking. – Ladislav Mrnka Feb 13 '17 at 13:42

1 Since EF6 infers the entity key incorrectly on a View sometimes, does AsNoTracking() ignore the key and therefore be an alternative to manually fixing the key (assuming other benefits of the key are not needed). – crokusek Nov 7 '17 at 22:51

2 Also note, the biggest effect AsNoTracking has is that lazy loading will not work – Douglas Gaskell May 23 '18 at 20:05

141

see this page Entity Framework and AsNoTracking

What AsNoTracking Does

Entity Framework exposes a number of performance tuning options to help you optimise the performance of your applications. One of these tuning options is `.AsNoTracking()` . This optimisation allows you to tell `Entity Framework` not to track the results of a query. This means that `Entity Framework` performs no additional processing or storage of the entities which are returned by the query. However, it also means that you can't update these entities without reattaching them to the tracking graph.

there are significant performance gains to be had by using AsNoTracking

edited Nov 13 '18 at 11:24

answered Apr 2 '14 at 9:26
Moji

my performance gain with a complex query loading a parent-child relationship with include in one step was about 50%

this should be the correct answer! – wenn32 Mar 29 at 8:25

39

No Tracking LINQ to Entities queries

Usage of AsNoTracking() is recommended when your query is meant for read operations. In these scenarios, you get back your entities but they are not tracked by your context.This ensures minimal memory usage and optimal performance

> **Pros**
>
> 1. Improved performance over regular LINQ queries.
> 2. Fully materialized objects.
> 3. Simplest to write with syntax built into the programming language.
>
> **Cons**
>
> 1. Not suitable for CUD operations.
> 2. Certain technical restrictions, such as: Patterns using DefaultIfEmpty for OUTER JOIN queries result in more complex queries than simple OUTER JOIN statements in Entity SQL.
> 3. You still can’t use LIKE with general pattern matching.

More info available here:

Performance considerations for Entity Framework

Entity Framework and NoTracking

edited Nov 9 '18 at 10:13
Bernard Vander Beken
2,837 5 36 58

answered Jun 13 '16 at 13:06
NullReference
1,608 20 29

- [How to avoid memory overflow when querying large datasets with Entity Framework and LINQ]
- [Entity framework large data set, out of memory exception]

edited May 23 '17 at 11:54
Community ♦
1 1

answered Dec 7 '13 at 2:03
Ronnie Overby
24.6k 62 232 326

9

AsNoTracking() allows the "unique key per record" requirement in EF to be bypassed (not mentioned explicitly by other answers).

This is extremely helpful when reading a View that does not support a unique key because perhaps some fields are nullable or the nature of the view is not logically indexable.

For these cases the "key" can be set to any non-nullable column but then AsNoTracking() must be used with every query else records (duplicate by key) will be skipped.

answered Feb 6 '18 at 23:09
crokusek
3,315 1 29 49

2 Just to reiterate the importance of this with Views, I have a query from a view which returns 7 unique records when run via SSMS. When run via EF, without the AsNoTracking modifier, I get the first record, three copies of the second and three copies of the third. This took a lot of incredulous head-scratching to fix, and it was using AsNoTracking which fixed it! – Ade Mar 22 '18 at 14:26

I had this exact same issue when using Linq to Entities while querying a View with no primary keys. Only found out about AsNoTracking after half a day of head-scratching. This ASP.Net forum post eventually led me to it. forums.asp.net/t/… – red_dorian Sep 14 '18 at 15:35

6

If you have something else altering the DB (say another process) and need to ensure you see these changes, use `AsNoTracking()` , otherwise EF may give you the last copy that your context had instead, hence it being good to usually use a new context every query:

http://codethug.com/2016/02/19/Entity-Framework-Cache-Busting/