Entity Framework: Why does the database get hit if the data is in the context?

Asked 15 years, 10 months ago Modified 12 years, 1 month ago Viewed 3k times

4

Why would the database be hit to find a record that is already represented in the ObjectContext?

So here's what I thought would happen when you query:

```
  SiteUser someUser = context.SiteUser.Where(role => role.UserID ==
1).ToList()[0];
```

Junk example but the idea is that I want to get a user from the table with the id of 1. Now assume this is the first time through so what I would guess is that it has to create the SiteUser list on the context, query the database, and then fill it's list. Using profiler I see this:

```
SELECT
 [Extent1].[UserID] AS [UserID],
 [Extent1].[MainEmail] AS [MainEmail],
 [Extent1].[Password] AS [Password],
 [Extent1].[UserName] AS [UserName]
FROM [TIDBA].[TI_USER] AS [Extent1]
WHERE 1 = [Extent1].[UserID]
```

Beautiful. It did what I expect and in the SiteUser list (if I dig far enough using Watch) I can see that there is one item in the context SiteUser list and it happens to be the hydrated object that represents this data row.

Next I want to change something without saving:

```
someUser.UserName = "HIHIHI";
```

Now say for some reason I want grab it again Using the same context (This is a weird example but it's actually a test so I could prove this happening) :

```
someUser = context.SiteUser.Where(role => role.UserID == 1).ToList()[0];
```

What I think would happen is it would look at the SiteUser list on the context since that's what the generated property says. (If not null, return list) Then it would look to see if it's there and return it. No database hit. Guess what profiler says...

```
SELECT
 [Extent1].[UserID] AS [UserID],
 [Extent1].[MainEmail] AS [MainEmail],
 [Extent1].[Password] AS [Password],
 [Extent1].[UserName] AS [UserName]
FROM [TIDBA].[TI_USER] AS [Extent1]
WHERE 1 = [Extent1].[UserID]
```

Hrm. Ok so I start thinking that maybe it's a gut check to see if anything has changed on that data item and update the SiteUser object ONLY on values I haven't changed on the client. (Sort of like context.Refresh(RefreshMode.ClientWins, context.SiteUser)) So I have it stopped at the :

```
someUser = context.SiteUser.Where(role => role.UserID == 1).ToList()[0];
```

Line and I change a value in the database (Password column) and let it hit the database. Nothing changed on the object.

Something doesn't seem right here. It hits the database to select the object I already have hydrated in the context yet it doesn't apply the change I manually made in the database. Why is it hitting the database at all then?

UPDATE Thanks to some links below, I was able to dig in a bit and find this:

Merge Option

Looks like there is an enumeration that is set to tell how to deal with loads. Now after reading that I saw this for MergeOption.AppendOnly:

> Objects that already exist in the object context are not loaded from the data source. This is the default behavior for queries or when calling the Load method on an EntityCollection<(Of <(TEntity>)>).

This would suggest that if I have it in the context, there should be no hit to the database. However, this doesn't seem to be true. It would make sense if OverwriteChanges or PreserveChanges were the defaults, but they are not. This seems to be contradictory to what is supposed to happen. Only thing I can think of is that "loaded" just means there are no overwrites. However, it doesn't mean there are no queries to the database.

entity-framework

Share edited Nov 20, 2012 at 17:50 asked Feb 12, 2009 at 21:58

Improve this question

Follow

Ryan
2,998 ● 3 ● 33 ● 41

Programmin Tool
6,527 ● 11 ● 52 ● 69

4 Answers

Sorted by: Highest score (default)

8

+100

context.SiteUser is an property of type ObjectQuery. When you execute an ObjectQuery, it will *always* hit the backing store. That's what they do. If you don't want to execute a database query, then don't use an ObjectQuery.

It sounds like what you really want is a function which says, "If the entity is already materialized in the context, then just return it. If it isn't, then go grab it from the database." As it happens, ObjectContext includes such a function, called **GetObjectByKey**

> GetObjectByKey tries to retrieve an object that has the specified EntityKey from the ObjectStateManager. If the object is currently not loaded into the object context, a query is executed in an attempt to return the object from the data source.

Share Improve this answer Follow

answered Feb 19, 2009 at 15:02

Craig Stuntz
127k ● 12 ● 256 ● 275

Hrm seems odd that it wouldn't do this by default. I guess the fact that the default MergeOption says not to look at the database for changes, but only holds that in the context of overwriting is annoying. Then again, it is called MergeOption... – Programmin Tool Feb 19, 2009 at 15:38

2

IMO, the reason that EF hits the database a second time is to make sure that there aren't any additional rows in the db that satisfy the query. It's possible that additional relevant rows have been inserted into the table since the first query was issued, and EF is seeing if any of those exist.

Share Improve this answer Follow

answered Feb 18, 2009 at 15:46

Sean Reilly
21.8k ● 4 ● 50 ● 63

If I understand your question, this should answer it:

1

> Actually, the way the entity framework does this by default is to require notifications of changes from the entity objects to a framework class called the state manager which then keeps track of which properties have been changed. The original values are copied only on demand. When updates happen, those original values are used in talking to the server only if the changed properties are marked as “concurrency tokens”. That is, for any concurrency token columns, when the framework is creating an update statement it will include a check to verify that the row in the database still has the original value, and if not it will raise an exception to notify the program that someone else has changed the row in the database. It’s also true that the entity framework doesn’t absolutely require notifications from the property setters, you can also determine what’s modified in the application code and call an explicit method on the framework to indicate which properties are changed (but then the framework will only have a record that the property is modified, it won’t have an original value).

Which comes from here. More can be read about it here, and here.

Edited to add:

It appears that with EF, there is an *ObjectStateManager* that tracks changes which never really allows for disconnected data. In order to have disconnected data, you'll have to call the ObjectContext.Detach method to disconnect your object. More can be found here and here.

Share
Improve this answer
Follow

edited Feb 16, 2009 at 15:43

answered Feb 16, 2009 at 15:28
GregD
7,000 ●5 ●36 ●61

The only thing that seems off is that this didn't involve an update. Now I can either be incredibly thick (Very possible) or I can assume that it by default ALWAYs checks the database regardless of a select, udpate, or whatever. Is that what I should assume? – Programmin Tool Feb 16, 2009 at 15:37

The links lead me to something that I have updated my original post with but unfortunately only add to my confusion of what's going on. – Programmin Tool Feb 16, 2009 at 16:35

What if you avoided the .ToList() and use .FirstOrDefault() ?

1

Share Improve this answer Follow

answered Feb 18, 2009 at 9:09
Davy Landman
15.4k ●6 ●53 ●76

Nope. First (Top) still hits it every time the linq "query" is run. – Programmin Tool Feb 19, 2009 at 14:54

did you do context.SiteUser.FirstOrDefault(role => role.UserID == 1) ? I'm wondering what would happen if you retrieved to otherUser and than compared the data in someUser to otherUser? – Davy Landman Feb 19, 2009 at 15:03

Values are still the same even if you somehow change them in the database between hydrating someUser and otherUser. I suspect this is because it knows it's in the context and the MergeOption by default says to ignore changes. – Programmin Tool Feb 19, 2009 at 15:41

If I'm not mistaken you'll get the same reference (because you're in the same context). – Davy Landman Feb 19, 2009 at 15:46

One can hope. If not, there might be something scary wrong with Entity Framework. – Programmin Tool Feb 19, 2009 at 18:01