Modifying data via the DbContext

DbContext SaveChanges always starts with transaction and will rollback if any transaction failed if not it will commit this is default behavior

Tracking Methods on DbSet and DbContext

context.Samurais.Add(…)

context.Samurais.AddRange(…)

context.Add(…)

context.AddRange(…)

Track via DbSet
DbSet indicates type

Track via DbContext
Context will discover type(s)

The approach that you adopt to modifying entities depends on whether the context is currently tracking the entity being modified or not.

In the following example, the entity is obtained by the context, so the context begins tracking it immediately. When you alter property values on a tracked entity, the context changes the `EntityState` for the entity to `Modified` and the ChangeTracker records the old property values and the new property values. When `SaveChanges` is called, an `UPDATE` statement is generated and executed by the database.

```
1. var author = context.Authors.First(a => a.AuthorId == 1);
2. author.FirstName = "Bill";
3. context.SaveChanges();
```

Since the ChangeTracker tracks which properties have been modified, the context will issue a SQL statement that updates only those properties that were changed:

```
1. exec sp_executesql N'SET NOCOUNT ON;
2. UPDATE [Authors] SET [FirstName] = @p0
3. WHERE [AuthorId] = @p1;
4. SELECT @@ROWCOUNT;
5. ',N'@p1 int,@p0 nvarchar(4000)',@p1=1,@p0=N'Bill'
```

Disconnected Scenario ^{link}

In a disconnected scenario such as an ASP.NET application, changes to an existing entity's property values can take place in a controller or service method, well away from the context. In these cases, the context needs to be informed that the entity is in a modified state. This can be achieved in several ways: setting the `EntityState` for the entity explicitly; using the `DbContext.Update` method (which is new in EF Core); using the `DbContext.Attach` method and then "walking the object graph" to set the state of individual properties within the graph explicitly.

Setting EntityState ^{link}

You can set the `EntityState` of an entity via the `EntityEntry.State` property, which is made available by the `DbContext.Entry` method.

```
1.  public void Save(Author author)
2.  {
3.      context.Entry(author).State = EntityState.Modified;
4.      context.SaveChanges();
5.  }
```

This approach will result in just the author entity being assigned the `Modified` state. Any related objects will not be tracked. Since the ChangeTracker is unaware of which properties were modified, the context will issue an SQL statement updating *all* property values (apart from the primary key value).

DbContext Update ^{link}

The `DbContext` class provides `Update` and `UpdateRange` methods for working with individual or multiple entities.

```
1.  public void Save(Author author)
2.  {
3.      context.Update(author);
4.      context.SaveChanges();
5.  }
```

As with setting the entity's `State`, this method results in the entity being tracked by the context as `Modified`. Once again, the context doesn't have any way of identifying which property values have been changed, and will generate SQL to update all properties. Where this method differs from explicitly setting the `State` property, is in the fact that the context will begin tracking any related entities (such as a collection of books in this example) in the `Modified` state, resulting

in UPDATE statements being generated for each of them. If the related entity doesn't have a key value assigned, it will be marked as Added, and an INSERT statement will be generated.

Attach ^{link}

When you use the Attach method on an entity, it's state will be set to Unchanged, which will result in no database commands being generated at all. All other reachable entities with key values defined will also be set to Unchanged. Those without key values will be marked as Added. However, now that the entity is being tracked by the context, you can inform the context which properties were modified so that the correct SQL to update just those values is generated:

```
1.  var context = new TestContext();
2.  var author = new Author {
3.      AuthorId = 1,
4.      FirstName = "William",
5.      LastName = "Shakespeare"
6.  };
7.  author.Books.Add(new Book {BookId = 1, Title = "Othello" });
8.
9.  context.Attach(author);
10. context.Entry(author).Property("FirstName").IsModified = true;
11. context.SaveChanges();
```

The code above will result in the author entity being marked as Modified, and SQL being generated to update just the FirstName property:

```
1.  exec sp_executesql N'SET NOCOUNT ON;
2.  UPDATE [Authors] SET [FirstName] = @p0
3.  WHERE [AuthorId] = @p1;
4.  SELECT @@ROWCOUNT;
5.  ',N'@p1 int,@p0 nvarchar(4000)',@p1=1,@p0=N'William'
```

Untracked

Untracked Graph Behavior

Adding New Children to Pre-existing, Unmodified Parent

DbSet.Add(Parent)

All objects marked Added
All objects inserted into DB

DbSet.Attach(Parent)

Default: all objects marked Unchanged

Objects with no key value, marked Added

Added objects are inserted into DB

DbSet.Update(Parent)

Default: all objects marked Modified

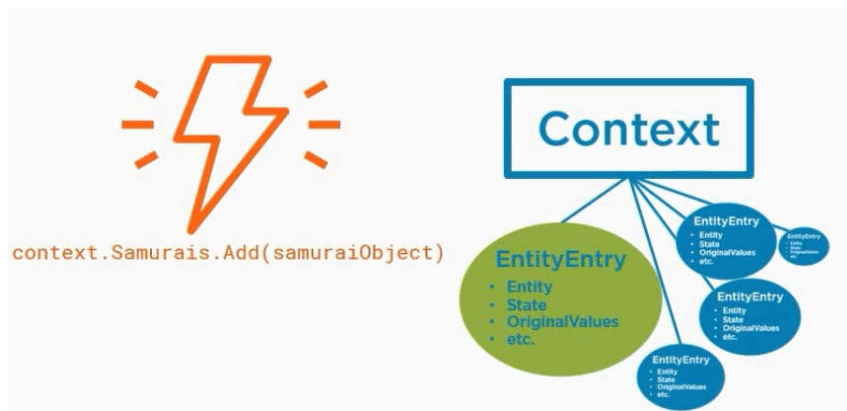Objects with no key value, marked Added

Modified objects are updated in DB

Added objects are inserted into DB

EF Core's Default Entity State of Graph Data

	Has Key Value	No Key Value
Add(graph)	Added*	Added
Update(graph)	Modified	Added
Attach(graph)	Unchanged	Added

Tracked

Update samo smenetoto property ke se update

Tracking is expensive

```
    ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
//will not track
            ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.TrackAll;
//default behavior
```


call it just before savechanges

and wirite appDbContext.ChangeTracker.Entries()

to add item to watch in Watch1

An Entity does not have state and are not tracked when retrived from front-end app

The `ChangeTracker` class in Entity Framework Core starts tracking of all the entities as soon as it is retrieved using `DbContext`, until they go out of its scope.

EF API maintains the state of each entity during its lifetime. Each entity has a state based on the operation performed on it via the context class. The entity state represented by an enum `System.Data.Entity.EntityState` in EF 6 and `Microsoft.EntityFrameworkCore.EntityState` in EF Core with the following values:

1. **Added:** The entity is marked as added.
2. **Deleted**: The entity is marked as deleted.
3. **Modified**: The entity has been modified.
4. **Unchanged**: The entity hasn't been modified
5. **Detached**: The entity isn't tracked.

The Context not only holds the reference to all the entity objects as soon as retrieved from the database, but also keeps track of entity states and maintains modifications made to the properties of the entity. This feature is known as *Change Tracking*.

The change in entity state from the Unchanged to the Modified state is the only state that's automatically handled by the context. All other changes must be made explicitly using proper methods of `DbContext` or `DbSet`.
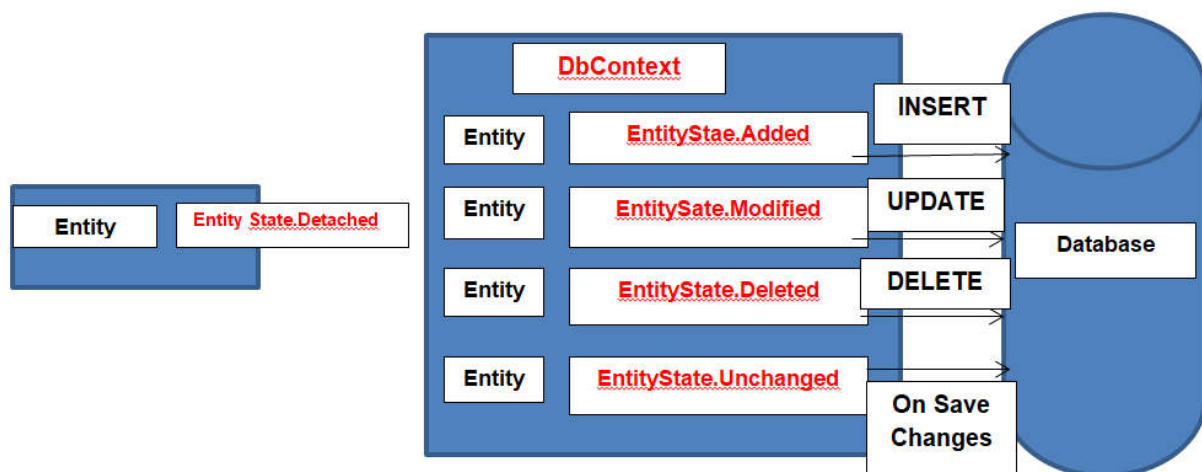
Entity State in Entity Framework

Let's discuss different states.

Unchanged State

The property values of the entity have not been modified since it was retrieved from the database. SaveChanges ignores this entity. This is the default state the entities will be in when we perform the query and also whenever we attach an entity to the context using Attach() method.

Detached State

Whenever we use Detach() method, the entity will be in the Detached state. Once the entity is in the Detached state, it cannot be tracked by the ObjectContext. We have to use Attach() method for the entity to be tracked by the ObjectContext. The Detached entity state indicates that the entity is not being tracked by the context.

Added State

Whenever we add a new entity to the context using the AddObject() method, the state of the entity will be in the Added state. Added entity state indicates that the entity exists in the context, but does not exist in the database. DbContext generates the INSERT SQL query and insert the data into the database when the **saveChanges** method is invoked. Once the saveChanges are successful the state of the entity is changed to Unchanged

Modified State:

The entity will be in a Modified state whenever we modify scalar properties. The Modified entity state indicates that the entity is modified but not updated in the database. It also indicates that the entity exists in the database. The Dbcontext generates the update SQL Query to remove the entity from the database. Once the saveChanges is successful the state of the entity is changed to Unchanged

In the Connected environment, the Entity framework also keeps track of the properties that have been modified. The Columns in the Update statement are set for only those columns, whose values are modified.

Deleted State

Whenever we call the DeleteObject() method, the entity will be deleted from the context and will be marked as "Deleted". When the SaveChanges method is called, the corresponding rows are deleted from the database. The Deleted entity state indicates that the entity is marked for deletion, but not yet deleted from the database. It also indicates that the entity exists in the database. The DbContext generates the delete SQL Query to remove the entity from the database. The entity is removed from the context once the delete operation succeeds after the saveChanges

errors

```
        //The instance of entity type 'Job' cannot be tracked because another instance
with the key value '{Id: 15}'
        //is already being tracked.When attaching existing entities, ensure that only
one entity instance
        //with a given key value is attached.
        //You don't need to call _dbSet.Update because as the error message
        //indicates the entity is already being tracked from your previous query.
```

because Update starts Tracking the entity again and many more on the images

is the dbContext is already tracking an entity and you try to track it again with the following images
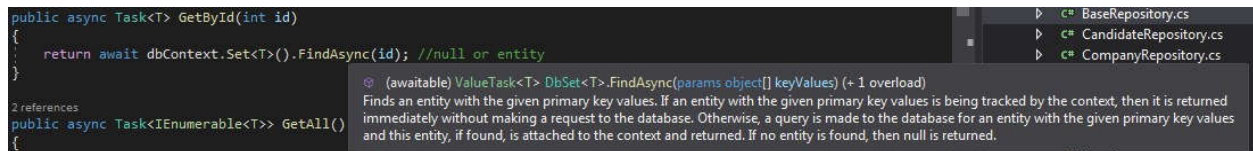
if is tracked it does not track again FindAsync
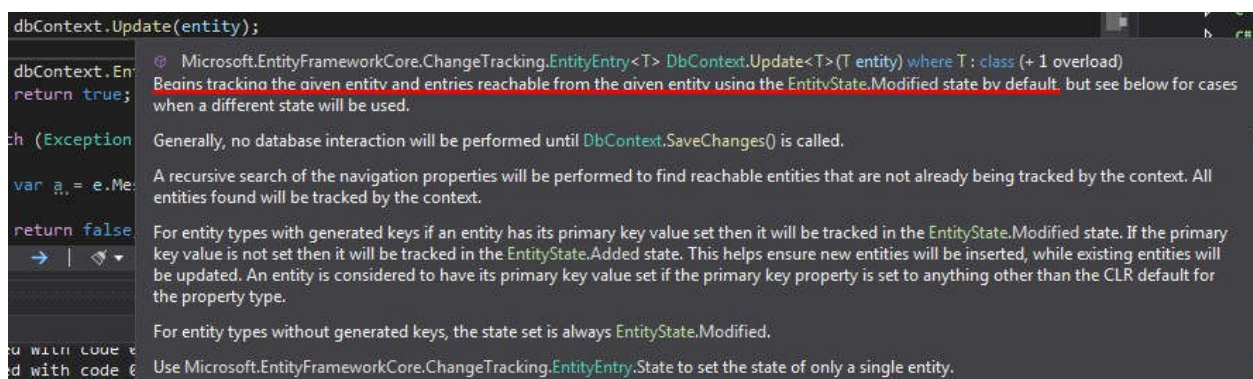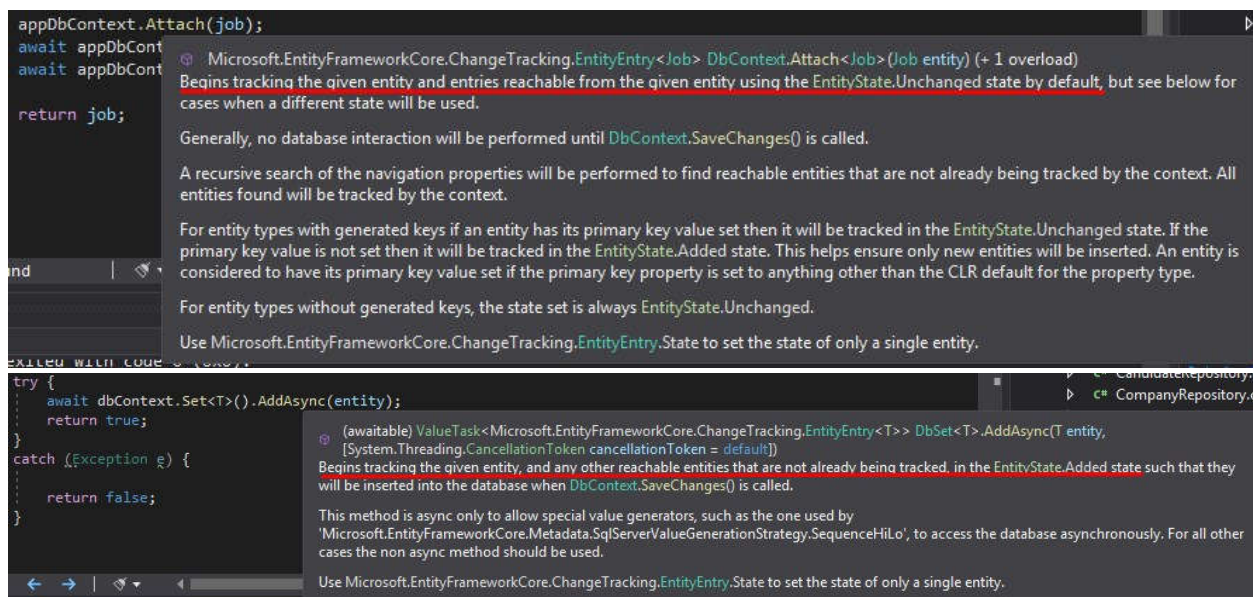
```
dbContext.Entry(entity).State = EntityState.Modified;//updates only the entety not his related ddata
return true;
                              EntityState.Modified = 3      will start tracking
                              The entity is being tracked by the context and exists in the database. Some or all of its property values have been modified.
h (Exception e)
```

you can see which are tracked the related of the current are not