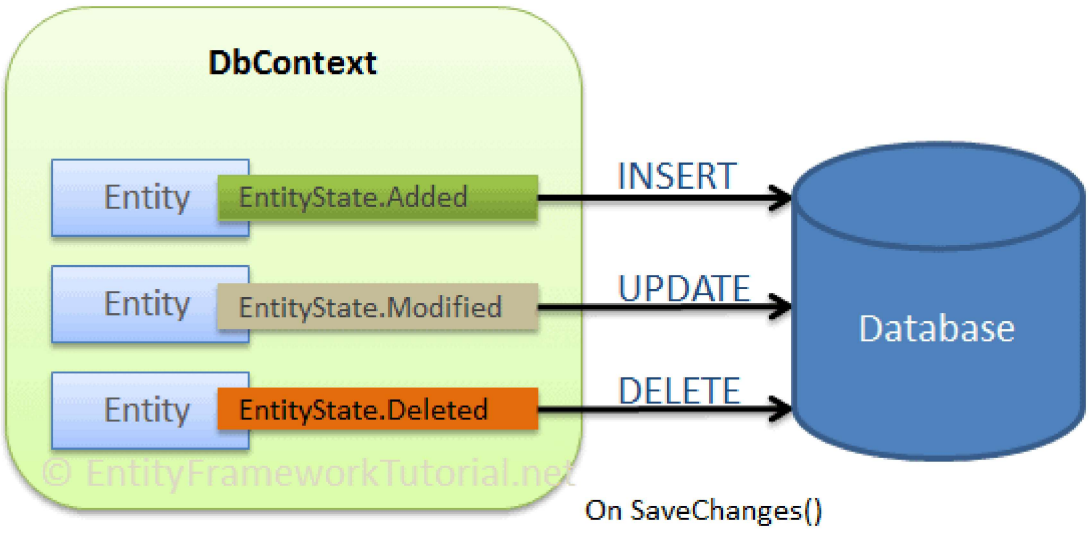
Previous Next

Entity Framework Core: Saving Data in Connected Scenario

Entity Framework Core provides different ways to add, update, or delete data in the underlying database. An entity contains data in its scalar property will be either inserted or updated or deleted based on its `EntityState`.

There are two scenarios to save an entity data: connected and disconnected. In the connected scenario, the same instance of `DbContext` is used in retrieving and saving entities, whereas this is different in the disconnected scenario. In this chapter, you will learn about saving data in the connected scenario.

The following figure illustrates the CUD (Create, Update, Delete) operation in the connected scenario.



As per the above figure, Entity Framework builds and executes INSERT, UPDATE, or DELETE statements for the entities whose `EntityState` is Added, Modified, or Deleted when the `DbContext.SaveChanges()` method is called. In the connected scenario, an

instance of `DbContext` keeps track of all the entities and so it automatically sets an appropriate `EntityState` of each entity whenever an entity is created, modified, or deleted.

Insert Data

The `DbSet.Add` and `DbContext.Add` methods add a new entity to a context (instance of DbContext) which will insert a new record in the database when you call the `SaveChanges()` method.

```
using (var context = new SchoolContext())
{
    var std = new Student()
    {
        FirstName = "Bill",
        LastName = "Gates"
    };
    context.Students.Add(std);

    // or
    // context.Add<Student>(std);

    context.SaveChanges();
}
```

In the above example, `context.Students.Add(std)` adds a newly created instance of the `Student` entity to a context with `Added` EntityState. EF Core introduced the new `DbContext.Add` method, which does the same thing as the `DbSet.Add` method. After this, the `SaveChanges()` method builds and executes the following INSERT statement to the database.

```
exec sp_executesql N'SET NOCOUNT ON;
INSERT INTO [Students] ( [FirstName], [LastName])
VALUES (@p0, @p1);
SELECT [StudentId]
FROM [Students]
WHERE @@ROWCOUNT = 1 AND [StudentId] = scope_identity();',N
'@p0 nvarchar(4000), @p1 nvarchar(4000) ',@p0=N'Bill',@p1=N'Gates'
go
```

Updating Data

In the connected scenario, EF Core API keeps track of all the entities retrieved using a context. Therefore, when you edit entity data, EF automatically marks `EntityState` to `Modified`, which results in an updated statement in the database when you call the `SaveChanges()` method.

```
using (var context = new SchoolContext())
{
    var std = context.Students.First<Student>();
    std.FirstName = "Steve";
    context.SaveChanges();
}
```

In the above example, we retrieve the first student from the database using `context.Students.First<student>()`. As soon as we modify the `FirstName`, the context sets its `EntityState` to `Modified` because of the modification performed in the scope of the `DbContext` instance (context). So, when we call the `SaveChanges()` method, it builds and executes the following Update statement in the database.

```
exec sp_executesql N'SET NOCOUNT ON;
UPDATE [Students] SET [FirstName] = @p0
WHERE [StudentId] = @p1;
SELECT @@ROWCOUNT;
',N'@p1 int,@p0 nvarchar(4000)',@p1=1,@p0=N'Steve'
Go
```

In an update statement, EF Core API includes the properties with modified values, the rest being ignored. In the above example, only the `FirstName` property was edited, so an update statement includes only the `FirstName` column.

Deleting Data

Use the `DbSet.Remove()` or `DbContext.Remove` methods to delete a record in the database table.

```
using (var context = new SchoolContext())
{
    var std = context.Students.First<Student>();
    context.Students.Remove(std);

    // or
    // context.Remove<Student>(std);

    context.SaveChanges();
}
```

In the above example, `context.Students.Remove(std)` or `context.Remove<Students>(std)` marks the `std` entity object as `Deleted`. Therefore, EF Core will build and execute the following DELETE statement in the database.

```
exec sp_executesql N'SET NOCOUNT ON;
DELETE FROM [Students]
WHERE [StudentId] = @p0;
SELECT @@ROWCOUNT;
',N'@p0 int',@p0=1
Go
```

Thus, it is very easy to add, update, or delete data in Entity Framework Core in the connected scenario.

Further Reading

- Saving data in the disconnected scenario in EF Core.

Previous

Next

ENTITYFRAMEWORKTUTORIAL

Learn Entity Framework using simple yet practical examples on EntityFrameworkTutorial.net for free. Learn Entity Framework DB-First, Code-First and EF Core step by step. While using this site, you agree to have read and accepted our terms of use and privacy policy.

---

feedback@entityframeworktutorial.net

TUTORIALS

- EF 6 DB-First
- EF 6 Code-First

E-MAIL LIST

Subscribe to EntityFrameworkTutorial email list and get EF 6 and EF Core Cheat Sheets, latest updates, tips & tricks about Entity Framework to your inbox.

Email address GO

We respect your privacy.
