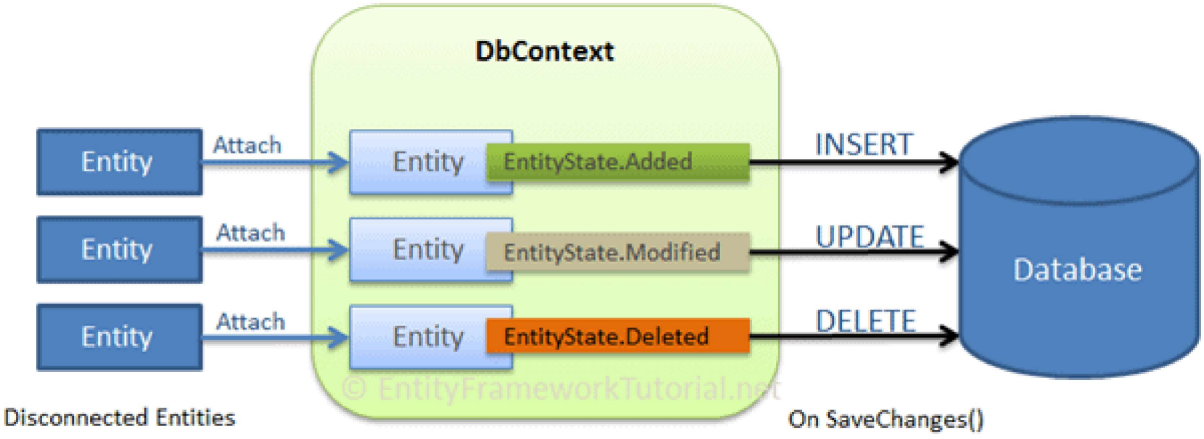
Previous Next

Insert Data in Disconnected Scenario in Entity Framework Core

You learned about how to save data in the connected scenario. Here, you will learn about saving data in the disconnected scenario.

Saving data in the disconnected scenario is a little bit different than in the connected scenario. In the disconnected scenario, the `DbContext` is not aware of disconnected entities because entities were added or modified out of the scope of the current `DbContext` instance. So, you need to attach the disconnected entities to a context with appropriate `EntityState` in order to perform CUD (Create, Update, Delete) operations to the database.

The following figure illustrates the CUD operations in disconnected scenario:



As per the above figure, disconnected entities (entities which are not being tracked by the `DbContext`) need to be attached to the `DbContext` with an appropriate `EntityState`. For example, Added state for new entities, Modified state for the edited

entities and Deleted state for the deleted entities, which will result in an INSERT, UPDATE, or DELETE command in the database when the `SaveChanges()` method is called.

The following steps must be performed in order to insert, update or delete records into the DB table using Entity Framework Core in disconnected scenario:

1. Attach an entity to `DbContext` with an appropriate `EntityState` e.g. Added, Modified, or Deleted
2. Call `SaveChanges()` method

The following example demonstrates inserting a new record into the database using the above steps:

```
//Disconnected entity
var std = new Student(){ Name = "Bill" };

using (var context = new SchoolContext())
{
    //1. Attach an entity to context with Added EntityState
    context.Add<Student>(std);

    //or the followings are also valid
    // context.Students.Add(std);
    // context.Entry<Student>(std).State = EntityState.Added;
    // context.Attach<Student>(std);

    //2. Calling SaveChanges to insert a new record into Students table
    context.SaveChanges();
}
```

In the example above, `std` is a disconnected instance of the `Student` entity. The `context.Add<Student>()` method attaches a `Student` entity to a context with an Added state. The `SaveChanges()` method builds and executes the following INSERT statement:

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

EF Core provides multiple ways to add entities with Added state. In the above example, `context.Students.Add(std);`, `context.Entry<Student>(std).State = EntityState.Added;` and `context.Attach<Student>(std);` will result in same INSERT statement as above.

Entity Framework Core provides the following `DbContext` and `DbSet` methods which attach disconnected entities with Added `EntityState`, which in turn will execute INSERT statements in the database.

DbContext Methods	DbSet Methods	Description
DbContext.Attach	DbSet.Attach	Attach an entity to DbContext. Set Unchanged state for an entity whose Key property has a value and Added state for an entity whose Key property is empty or the default value of data type.
DbContext.Add	DbSet.Add	Attach an entity to DbContext with Added state.
DbContext.AddRange	DbSet.AddRange	Attach a collection of entities to DbContext with Added state.
DbContext.Entry	-	Gets an `EntityEntry` for the specified entity which provides access to change tracking information and operations.
DbContext.AddAsync	DbSet.AddAsync	Asynchronous method for attaching an entity to DbContext with Added state and start tracking it if not. Data will be inserted into the database when SaveChangesAsync() is called.

DbContext Methods	DbSet Methods	Description
DbContext.AddRangeAsync	DbSet.AddRangeAsync	Asynchronous method for attaching multiple entities to DbContext with Added state in one go and start tracking them if not. Data will be inserted into the database when SaveChangesAsync() is called.

Note: The above `DbContext` methods are introduced in EF Core (they were not available in EF 6 or prior). Both `DbContext` and `DbSet` methods perform the same operation. Which one you use depends on your coding pattern and preference.

Insert Relational Data

In the previous chapter, we learned to create one-to-one, one-to-many and many-to-many relationships between two entities. Entity Framework API inserts all the relational data contained in related entities.

Use the `DbContext.Add` or `DbSet.Add` method to add related entities to the database. The `Add` method attaches entities to a context and sets the Added state to all the entities in an entity graph whose Id (Key) properties are empty, null or the default value of data type. Consider the following example.

```
var stdAddress = new StudentAddress()
{
    City = "SFO",
    State = "CA",
    Country = "USA"
};

var std = new Student()
{
    Name = "Steve",
    Address = stdAddress
};
using (var context = new SchoolContext())
{
    // Attach an entity to DbContext with Added state
    context.Add<Student>(std);

    // Calling SaveChanges to insert a new record into Students table
    context.SaveChanges();
}
```

In the example above, `context.Add<Student>(std)` adds an instance of `Student` entity. EF Core API reaches the `StudentAddress` instance through the reference navigation property of `Student` and marks `EntityState` of both the entities to Added, which will build and execute the following two INSERT commands on `SaveChanges()`.

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

exec sp_executesql N'SET NOCOUNT ON;
INSERT INTO [StudentAddresses] ([Address], [City], [Country], [State], [StudentId])
VALUES (@p5, @p6, @p7, @p8, @p9);
SELECT [StudentAddressId]
FROM [StudentAddresses]
WHERE @@ROWCOUNT = 1 AND [StudentAddressId] = scope_identity();
',N'@p5 nvarchar(4000),@p6 nvarchar(4000),@p7 nvarchar(4000),@p8 nvarchar(4000),
@p9 int',@p5=NULL,@p6=N'SFO',@p7=N'USA',@p8=N'CA',@p9=1
Go
```

Insert Multiple Records

Use the `DbContext.AddRange` or `DbSet.AddRange` method to add multiple entities in one go. You don't need to call `DbContext.Add` method multiple times.

AddRange Methods	Description
`void AddRange(IEnumerable<Object> entities)`	Adds a list of same or different types of entities to the DbContext with Added state.
`void AddRange(param object[] entities)`	Adds an array of same or different types of entities to the DbContext with Added state.
`void AddRangeAsync(IEnumerable<Object>, CancellationToken)`	Asynchronous method to add a list of same or different types of entities to the DbContext with Added state.

The following example demonstrates adding a list of `Student` entity objects using AddRange.

```
var studentList = new List<Student>() {
    new Student(){ Name = "Bill" },
    new Student(){ Name = "Steve" }
};

using (var context = new SchoolContext())
{
    context.AddRange(studentList);
    context.SaveChanges();
}
```

The above example will insert two new records in the `Students` table.

You can also add a list of different types of entities, as shown below.

```
var std1 = new Student(){ Name = "Bill" };

var std2 = new Student(){ Name = "Steve" };

var computer = new Course() { CourseName = "Computer Science" };

var entityList = new List<Object>() {
    std1,
    std2,
    computer
};

using (var context = new SchoolContext())
{
    context.AddRange(entityList);

    // or
    // context.AddRange(std1, std2, computer);

    context.SaveChanges();
}
```

In the above example, entityList is a type of `List<Object>`. So, it can contain any type of entities. The `AddRange()` method adds all the specified entities to a context and `SaveChanges()` will build and execute INSERT statements for all in one go.

EF Core improves the performance by executing INSERT statements for all the above entities in a **single database round trip**. The above example will execute the following statements in the database.

```
exec sp_executesql N'SET NOCOUNT ON;
INSERT INTO [Courses] ([CourseName], [Description])
VALUES (@p0, @p1);
SELECT [CourseId]
FROM [Courses]
WHERE @@ROWCOUNT = 1 AND [CourseId] = scope_identity();

DECLARE @inserted1 TABLE ([StudentId] int, [_Position] [int]);
MERGE [Students] USING (
VALUES (@p2, 0),
(@p3, 1)) AS i ([Name], _Position) ON 1=0
WHEN NOT MATCHED THEN
INSERT ([Name])
VALUES (i.[Name])
OUTPUT INSERTED.[StudentId], i._Position
INTO @inserted1;

SELECT [t].[StudentId] FROM [Students] t
INNER JOIN @inserted1 i ON ([t].[StudentId] = [i].[StudentId])
ORDER BY [i].[_Position];
',N'@p0 nvarchar(4000),@p1 nvarchar(4000),@p2 nvarchar(4000),@p3 nvarchar(4000)',
@p0=N'Computer Science',@p1=NULL,@p2=N'Steve',@p3=N'Bill'
go
```

Insert Data Using DbSet

As mentioned before, you can use DbSet to save an instance of an entity which will be translated into INSERT/UPDATE/DELETE command in the database, in the same way as EF 6.x.

Use the `DbSet<TEntity>.Add()` method to attach an entity with Added state or the `DbSet<TEntity>.AddRange()` method to attach a collection of entities with Added state, as shown below.

```
var std = new Student()
{
    Name = "Bill"
};

using (var context = new SchoolContext())
{
    context.Students.Add(std);

    // or
    // context.Students.Attach(std);

    context.SaveChanges();
}
```

In the above example, the type of `context.Students` is `DbSet<Student>` type. So, we can add the `Student` entity only. The `context.Students.Add(std)` attaches the `Student` entity to the context with the Added state, which will result in the INSERT statement when the `SaveChanges()` method is called.

Learn how to update disconnected entities in the next chapter.

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

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.