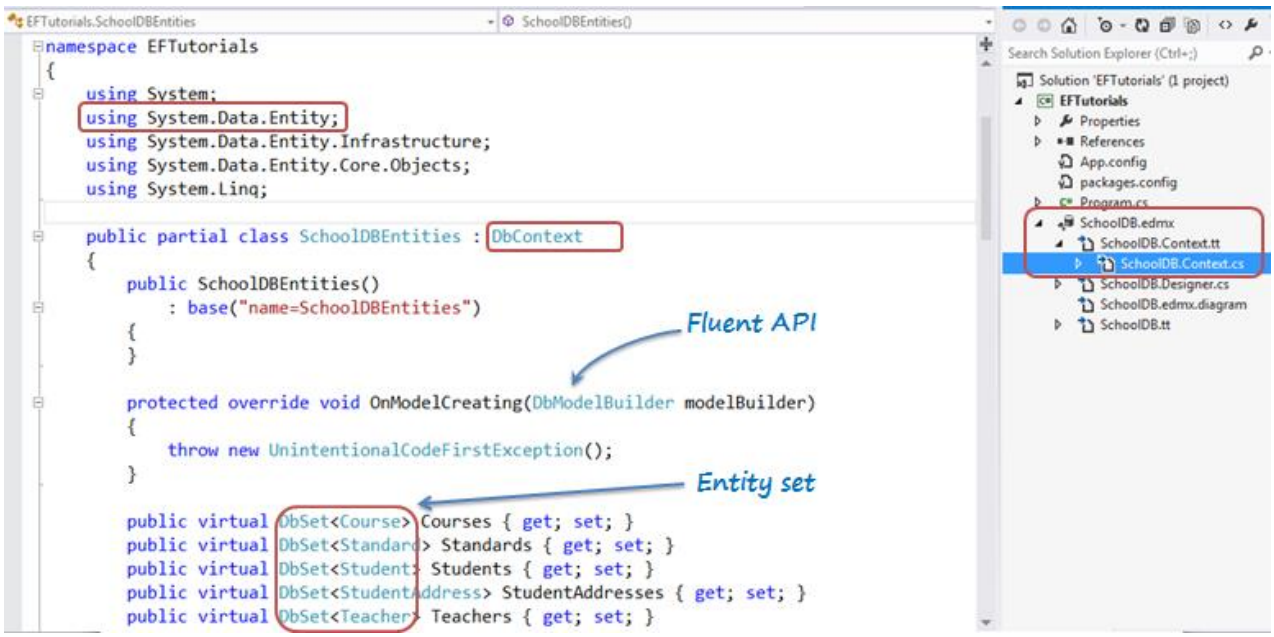
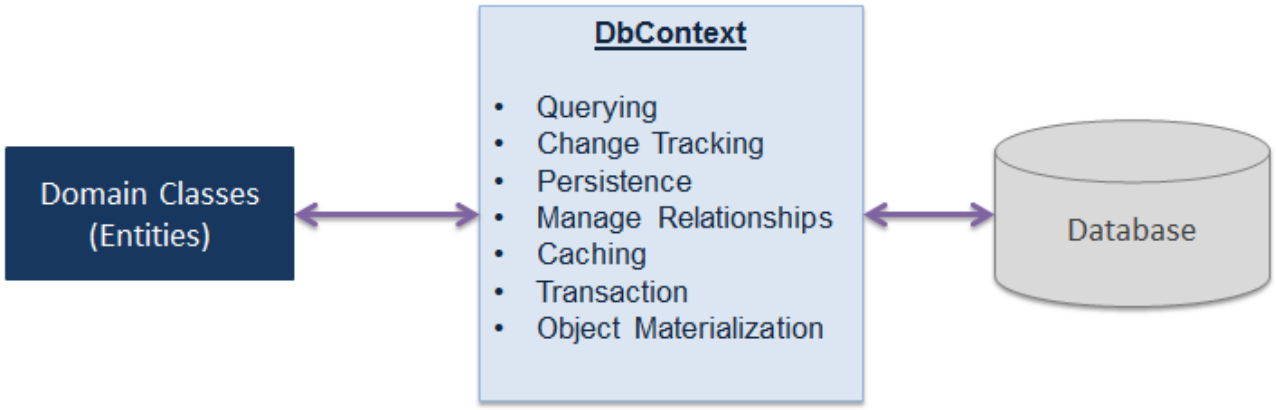
DbContext in Entity Framework 6:

The ApplicationDbContext class is derived from the *System.Data.Entity.DbContext* class. A class that derives DbContext is called a context class in entity framework.



DbContext is an important class in Entity Framework API. It is a bridge between your domain or entity classes and the database.



DbContext is the primary class that is responsible for interacting with the database. It is responsible for the following activities:

- **Querying:** Converts LINQ-to-Entities queries to SQL query and sends them to the database.
- **Change Tracking:** Keeps track of changes that occurred on the entities after querying from the database.
- **Persisting Data:** Performs the Insert, Update and Delete operations to the database, based on entity states.
- **Caching:** Provides first level caching by default. It stores the entities which have been retrieved during the life time of a context class.
- **Manage Relationship:** Manages relationships using CSDL, MSL and SSDL in Db-First or Model-First approach, and using fluent API configurations in Code-First approach.
- **Object Materialization:** Converts raw data from the database into entity objects.

The following code is an example. The `SchoolDBEntities` class (context class that derives from `DbContext`) is generated with EDM for the `SchoolDB` database in the previous section.

```
namespace EFTutorials
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;
    using System.Linq;

    public partial class SchoolDBEntities : DbContext
    {
        public SchoolDBEntities()
            : base("name=SchoolDBEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Course> Courses { get; set; }
        public virtual DbSet<Standard> Standards { get; set; }
        public virtual DbSet<Student> Students { get; set; }
        public virtual DbSet<StudentAddress> StudentAddresses { get; set; }
        public virtual DbSet<Teacher> Teachers { get; set; }
        public virtual DbSet<View_StudentCourse> View_StudentCourse { get; set;
}

        public virtual ObjectResult<GetCoursesByStudentId_Result> GetCoursesByS
tudentId(Nullable<int> studentId)
        {
            var studentIdParameter = studentId.HasValue ?
```

```
                new ObjectParameter("StudentId", studentId) :
                new ObjectParameter("StudentId", typeof(int));

            return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<
GetCoursesByStudentId_Result>("GetCoursesByStudentId", studentIdParameter);
        }

        public virtual int sp_DeleteStudent(Nullable<int> studentId)
        {
            var studentIdParameter = studentId.HasValue ?
                new ObjectParameter("StudentId", studentId) :
                new ObjectParameter("StudentId", typeof(int));

            return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction(
"sp_DeleteStudent", studentIdParameter);
        }

        public virtual ObjectResult<Nullable<decimal>> sp_InsertStudentInfo(Nul
lable<int> standardId, string studentName)
        {
            var standardIdParameter = standardId.HasValue ?
                new ObjectParameter("StandardId", standardId) :
                new ("StandardId", typeof(int));

            var studentNameParameter = studentName != null ?
                new ObjectParameter("StudentName", studentName) :
                new ObjectParameter("StudentName", typeof(string));

            return ((IObjectContextAdapter)this).ObjectContext
                    .ExecuteFunction<Nullable<decimal>>("sp_InsertStudentInfo",
                    standardIdParameter, studentNameParameter);
        }

        public virtual int sp_UpdateStudent(Nullable<int> studentId,
```

```
          Nullable<int> standardId, string studentName)
        {
            var studentIdParameter = studentId.HasValue ?
                new ObjectParameter("StudentId", studentId) :
                new ObjectParameter("StudentId", typeof(int));

            var standardIdParameter = standardId.HasValue ?
                new ObjectParameter("StandardId", standardId) :
                new ObjectParameter("StandardId", typeof(int));

            var studentNameParameter = studentName != null ?
                new ObjectParameter("StudentName", studentName) :
                new ObjectParameter("StudentName", typeof(string));

     return ((IObjectContextAdapter)this).ObjectContext
              .ExecuteFunction("sp_UpdateStudent",
              studentIdParameter, standardIdParameter,
               studentNameParameter);
        }
    }
}
```

As you can see in the above example, the context class (`SchoolDBEntities`) includes the entity set of type `DbSet<TEntity>` for all the entities. Learn about DbSet class here. It also includes functions for the stored procedures and views included in EDM.

The `OnModelCreating` method allows us to configure the model using `DbModelBuilder` Fluent API in EF 6.

DbContext Methods:

Method	Usage
Entry	Gets an `DbEntityEntry` for the given entity. The entry provides access to change tracking information and operations for the entity.
SaveChanges	Executes INSERT, UPDATE and DELETE commands to the database for the entities with Added, Modified and Deleted state.
SaveChangesAsync	Asynchronous method of SaveChanges()
Set	Creates a `DbSet<TEntity>` that can be used to query and save instances of `TEntity`.
OnModelCreating	Override this method to further configure the model that was discovered by convention from the entity types exposed in `DbSet<TEntity>` properties on your derived context.

DbContext Properties:

Method	Usage
ChangeTracker	Provides access to information and operations for entity instances that this tracking.
Configuration	Provides access to configuration options.
Database	Provides access to database related information and operations.