how to configure and use SQL Server with entity framework core.  
  
  
When using Entity Framework Core, one of the important things that we need to configure is the database provider that we plan to use. Entity Framework Core supports a wide variety of databases including non-relational databases. The following MSDN link has the list of all supported databases.  
<https://docs.microsoft.com/en-us/ef/core/providers/>

public class Startup

{

    private IConfiguration \_config;

    public Startup(IConfiguration config)

    {

        \_config = config;

    }

    public void ConfigureServices(IServiceCollection services)

    {

        services.AddDbContextPool<AppDbContext>(

            options => options.UseSqlServer(\_config.GetConnectionString("EmployeeDBConnection")));

        services.AddMvc().AddXmlSerializerFormatters();

        services.AddTransient<IEmployeeRepository, MockEmployeeRepository>();

    }

    // Rest of the code

}

* We want to configure and use Microsoft SQL Server with entity framework core.
* We usually specify this configuration in *ConfigureServices()* method in *Startup.cs*file.

**Difference between AddDbContext() and AddDbContextPool() methods**

* We can use either *AddDbContext()* or *AddDbContextPool()* method to register our application specific DbContext class with the ASP.NET Core dependency injection system.
* The difference between *AddDbContext()* and *AddDbContextPool()* methods is, *AddDbContextPool()*method provides DbContext pooling.
* With DbContext pooling, an instance from the DbContext pool is provided if available, rather than creating a new instance.
* DbContext pooling is conceptually similar to how connection pooling works in ADO.NET.
* From a performance standpoint *AddDbContextPool()* method is better over *AddDbContext()*method.
* AddDbContextPool() method is introduced in ASP.NET Core 2.0. So if you are using ASP.NET Core 2.0 or later use *AddDbContextPool()* method over *AddDbContext()*method.

**UseSqlServer() Extension Method**

* *UseSqlServer()* extension method is used to configure our application specific *DbContext*class to use Microsoft SQL Server as the database.
* To connect to a database, we need the database connection string which is provided as a parameter to *UseSqlServer()* extension method

services.AddDbContextPool<AppDbContext>(

    options => options.UseSqlServer(\_config.GetConnectionString("EmployeeDBConnection")));

**Database Connection String in ASP.NET Core**  
  
Instead of hard-coding the connection string in application code, we store it appsettings.json configuration file.

{

  "ConnectionStrings": {

    "EmployeeDBConnection": "server=(localdb)\\MSSQLLocalDB;database=EmployeeDB;Trusted\_Connection=true"

  }

}

In classic asp.net we store application configuration in web.config file which is in XML format. In asp.net core, there are different configuration sources. One configuration source is *appsettings.json* file and it is in JSON format.  
  
To read connection string from *appsettings.json*file we use *IConfiguration*service *GetConnectionString()* method.  
  
We are using SQL Server localdb which is automatically installed along with Visual Studio. If you want to use a full blown SQL Server instead of localdb, simply change the connection string in *appsettings.json*configuration file to point to your instance of SQL Server.

"server=(localdb)\\MSSQLLocalDB;database=EmployeeDB;Trusted\_Connection=true"

What is the difference between the following in a database connection string

* Trusted\_Connection=True;
* Integrated Security=SSPI;
* Integrated Security=true;

All the above 3 settings specify the same thing, use Integrated Windows Authentication to connect to SQL Server instead of using SQL Server authentication.  
  
At the moment our application is still using MockEmployeeRepository which is an in-memory collection of employees. In our next Session we will implement SQLRepository which stores and retrieves employees from sql server localdb that we have just configured.