Till this day :-

1. Basics of c#, Framework, .NET core.
2. Datatypes, input / output op.
3. loops
4. class, .ctor, Overlodding of .ctor
   base keyword., Inheritance [:]
5. Inheritance - diff. overridding & shadowing
   - virtual - override
   - new keyword.
   - Sealed classes / sealed methods
6. Why Interface & Why Abstract -
   - Factory Pattern
     - class
     - method
   - using Interface how to establish a contract bet.n client - Server.
   - using Interface how to achieve template / structure in your appl.n.
7. Abstract :-
   why -: when base wants entire responsibility for application execution flow.
8. class library
   - access modifiers.
9. Logger - static
   - Singleton Design
   - object Pooling.
10. single Responsibility Principle
11. Dependency Injection - .ctor level

12. Events & Delegates, Generic delegates.
- Click : button class
- delegates
  : += → coupling operation
  : -= → decoupling operation
- Predicate
  Action
  Action<>
  Func<>

13 Collection, Generic Class, methods
Generic collection
- Boxing, Unboxing, Type casting
- dynamic type

14. File IO & Serialization
- Binary
- JSON
- XML
- SOAP ✓
- using block ✓

15. Reflection — : intellicense
: working of ORM APIs

: Attributes
: Custom Attributes: Table, Column

16. C Sharp features

—: LINQ

var result = (from : in

where

select) ;

17. ADO.Net

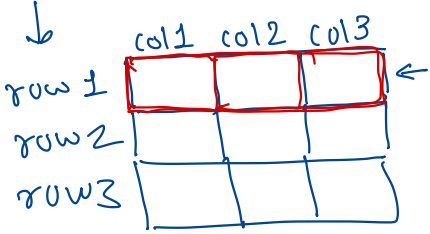
—: Connected Archi

—: Disconnected Arch.

—: Entity Framework

Dataset → coll.n → DataTable → DataRow collection → coll.n → Data colum

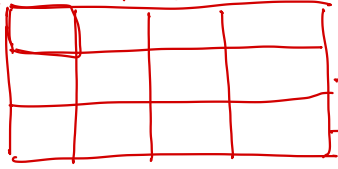
DataTable Emp. = collection of ROWS



collection of DataColumn

← 1 DataROW

DataTable Cust = row1 [col1 col2 col3 col4]



→ customer 1 obj

→ customer 2 obj

→ customer 3 obj

Connected Arch

- manually open & close conn.
- SqlConnection
- SqlCommand
- ExecuteReader()
- ExecuteNonQuery() – insert
  – Update
  – delete
- DataSet, DataTable, DataRow, DataColumn
- Manually write / Create – insert, update, delete queries for SqlCommand Obj. – Developer's responsibility

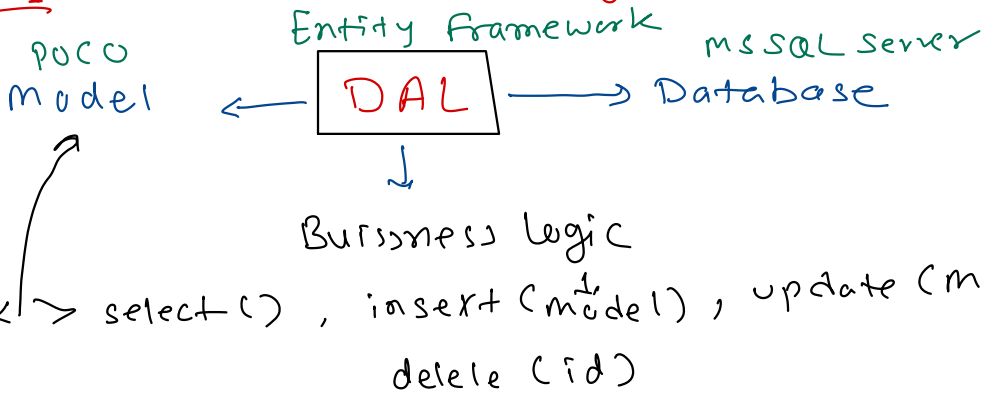
Disconnected Arch

- Outsource to DataAdapter
- SqlConnection
- [SqlDataAdapter
  SqlCommandBuilder]
- da.Fill()
- da.Update() – insert
  – Update
  – delete
- DataSet, DataTable, DataRow, DataColumn
  [Default Implementation] [da, dt]
- SqlCommandBuilder with the help of SqlDataAdapter, observes and creates queries accordingly – insert, Update, delete

- Entity framework :—

ORM [Object Relational Mapping Fw]

DAL :- Data Access Layer



Entity f/w – based on Connected Arch.

– uses LINQ FIW internally.

– around collections.

↑ Observe.

↓ Change

Insert
Update
Delete
} Queries generate

1. Create Application [console, web]
   ↓
   MVC core.

2. Install 4 Packages – Nuget package manager.
   1. Microsoft.EntityFrameworkCore [6.0.36] version
      :- initializes DbContext.class
   2. Microsoft.EntityFrameworkCore.Tools [6.0.36]
      :- primarily used to manage migrations and to scaffold a DbContext.
   3. Microsoft.EntityFrameworkCore.Design [6.0.36]
      :- contains all the design-time logic for Entity
   4. Microsoft.EntityFrameworkCore.SqlServer [6.0.36]
      : Database provider

3. Step 3 :-

1. Create Model Class

2. Create DbContext Class.

↳ inherit from DbContext which belongs to EntityFramework. Package.

↳ DbSet<Model> models { get; set; } create this property.

↳ Override OnConfiguring()

→ read con. string from appsettings.json file.

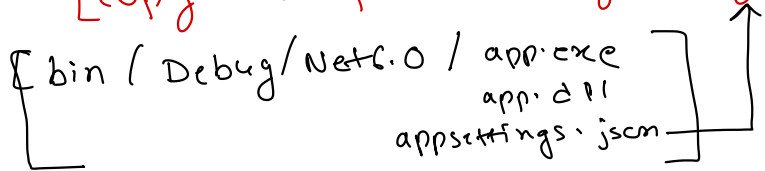
↳ add json file in your project. save it with name "appsettings.json"

↳ write connection string

↳ click on appsettings.json

change settings thr' property window

[Copy to Output Directory = Copy Always]



4. Add migrations [Code First Approach]

Finitely Infinite

model ⟶ Database Table.

1st command: `Add-migration Employee migration`

- [checks if table already exists]
- : Create Table Query.

2nd command: `update-database`

- : fires create table query on Database
- [generates table Employee if not there]

5. Using DbContext write Buissness Logic.

e.g:- DbContext.employees.ToList()

for configuration import

Microsoft.Extensions.Configuration.Json