

PROGRAMMING LANGUAGE

Kurs və əlavə məlumatlar



**KHAN**

**C#**

* **T-SQL**
* **ADO.NET**
* **WEB SERVICE**
* **MVC – ASP.NET**
* **HTML + CSS**
* **JAVASCRIPT**

**C#**

**ORXAN MƏMMƏDƏLİ**

Kurs və əlavə məlumatlar

# Types and Variables

Burada məlumatlandırma üçün ilkin olaraq console applicationdan istifadə ediləcəkdir.

namespace TypesAndVariables

{

class Program

{

static void Main(string[] args)

{

// Proqramin islek hissesi

Console.WriteLine("Hello Word");

// ekranda qalmasi ucun

Console.ReadLine();

}

}

}

* int tipi (type int, variable number1) = 32 bit

int number1 = 10; // int-un uzunluq +- 2 147 483 647 dir

Console.WriteLine("Number is {0}", number1);

Console.ReadLine();

* int tipindən daha uzun rəqəm lələb olunarsa, long tipindən istifadə edilir. Uzunluğu 19 rəqəmdən ibarət ədəddir. = 64 bit

long number1 = 10; // long-un uzunluq +- 9 223 372 036 854 775 807 dir

* Qeyd: Yaddaşda int-ə nisbətən 2 dəfə çox yer tutur.
* Short = 16 bit

short number1 = 10; // long-un uzunluq +- 32 767 dir

* Byte = 8 bit

byte number1 = 10; // long-un uzunluq 0 dan 255 -ə kimi

* Bool

bool condition = true; //condition -sert demekdir

bool condition2 = false;

* Char

char character = 'a';

* Qeyd: Console.WriteLine((int)character); // xususi hal her simvol bir edede uygundur.
* String

string city = "Baki"; // charlardan ibaret massiv tipi

* Double = 64 bit. Vergülden sonra 18 simvol.(əsasən hesablama işləmləri üçün)

double number5 = 10.4; // onluq saylari tutur

* Decimal Vergülden sonra 28 simvol.(əsasən money tipi işləmləri üçün)

decimal number6 = 10.4m;// double tipinin nisbeten daha uzun (for money)

decimal number7 = 10;

* Qeyd: tipində onluq saydan istifadə edən zaman “m” və ya “M” smvolundan istifadə olunmalıdır.
* Enum – magic string Sonradan təkrarlanacaq string məlumatların təkrar-təkrar yazmaq əvəzinə bir dəfə yazılaraq istifadə edilməsi üçün yararlıdır.

namespace TypesAndVariables

{

enum days

{

Mondey, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine(days.Friday); // əks olunan məlumat “Friday”

Console.WriteLine((int)days.Friday); // əks olunan məlumat “4” (index)

Console.ReadLine();

}

}

}

* Qeyd: Normalda Enum daxilində index “0”-dan başlayır. Lakin başlanğıc index dəyişdirilə bilər. Aşağıdakı nümunədə başlanğıc index = 10 olacaq. *(Xüsusi olaraq index verilməzsə, 1-1 artım baş verir)*

enum days

{

Mondey=10, Tuesday=20, Wednesday=30, Thursday, Friday, Saturday, Sunday

}

* Var keyword-ü bir dəyişən tipi deyildir, lakin dəyişən tiplərinin tutmaq üçün istifadə edilir. Bu keyword-ə hansı tip mənimsədələrsə onun dəyişən tipi həmin tipə mənimsədilmiş olur və həmin dəyərə sonradan başqa tipli məlumat mənimsədilə bilmir.

var number8 = 10; ~~number8 = "A"~~

# DƏRS 1. (OPERATORLAR)

## Arifmetik

(// + - \* \ %)

* UNUTMA: “OPERANTLAR” operatorları təşkil edən elementlərdir.

int a = 0;

int b = 0;

int c = a + b; // bu yazılış daxilində a və b operantlardır.

* Qeyd: %-mod qalığı göstərmək üçün istifadə olunur.

## Mənimsətmə

(// += - = \*= \= %=)

int d1 = 10;

int d2 = 12;

d1 += d2; // d1 = d1 + d2

* Qeyd: Əsasən dövri əməliyyatlarda istifadə olunur.

## Qarşilaşdirma

(// == != >= <= < >)

bool m = a > b; // True, False

## Məntiqi

(AND, OR)

## Xüsusi

(IF funksiyasının xüsusi halı)

string c = (a > b) ? "alma" : "armud";

***SADALANAN OPERATORLAR XƏTTİ ALQORİTMƏ SAHİBDİR.***

# DƏRS 2. (OPERATORLAR)

## Dəyişən tipləri

1. byte (0-255)
2. int [16, 32, 64]
3. short, int, long
4. double (100,5)
5. decimal
6. flout
7. bool
8. char
9. string

## Şərt Operatorları (IF, SWITCH)

if (true)

{

/.../

}

else if (true)

{

/ .../

}

....

else

{

/ .../

}

* Qeyd: Əgər IF funksiyası mörtərizəsiz yazılarsa şərt yanlız özündən sonraki 1 sətrə aid olur.

switch (switch\_on)

{

case sert:

/.../

break;

case sert:

/ .../

break;

case sert:

/ .../

break;

/.../

default:

break;

}

* Qeyd: Əgər SWITCH funksiyasıda “break” əmrindən istifadə edilməzsə hər bir “case” yoxlanır. Performansı artırmaq üçün əgər şərt ödənilərsə əməliyyat icra edildikdən sonra şərtdən çıxma təmin edilməlidir.
* UNUTMA1: SWITCH funksiyasıda “case”-lər daxilində “goto” əmrindən istifadə edərək “case”-ləe arası keçid etmək mümkündür.

* UNUTMA2: Əgər ardıcıl gələn iki və daha artıq case eyni əməliyyatı icra edərlərsə, bu zaman eyni əməliyyatı icra edən sonuncu case-ə “break” yazmaq tələb olunur

switch (switch\_on)

{

case sert1:

case sert2:

case sert3:

/ .../ // UNUTMA2 - eyni emeliyyati icra eden case-ler

break;

case sert4:

/ .../

break;

case sert5:

goto case sert4; // UNUTMA1 - bir case-den digerine yonlendirme

break;

/ .../

default:

break;

}

# DƏRS 3. (İÇ-İÇƏ IF FUNKSİYASİ)

* Qeyd: IF (şərt) funksiyasının iç-içə işlədilməsi ehtiyacının yaradılması bir növ performansın artırılması məqsədi güdür.

## CheckBox haqqında ilk məlumatlar

# DƏRS 4. (DÖVR OPERATORLARI)

## Null və Empty anlayışı

## Dövr operatorları ( For, Foreach, While, Do while)

int k = 0;

for (int i = 1; i < 10; i++)

{

k += i;

}

* Qeyd: Əgər FOR daxilində i = 0 olarsa, II şərt i < və ya i > formatında olmalıdır. i = 1 olarsa, i < = və ya i >= halları ola bilər. UNUTMA

## WHILE => Verilmiş aralıqdakı dəyərlərin cəmi

int netice = 0;

int d1 = Convert.ToInt32(textBox1.Text);

int d2 = Convert.ToInt32(textBox2.Text);

int i = d1;

while (i <= d2)

{

netice += i;

i++;

}

## Do WHILE

int netice = 0;

int i = 1;

do

{

netice += i;

i++;

} while (i <= 10);

* Qeyd: Do while dövr operatorundan istifadə edilən zaman birinci dövr istisnasız olaraq hər zaman icra edilir.

## ForEach

Random tesadufiEded = new Random();

int[] ededler = new int[10];

ededler[0] = tesadufiEded.Next(0,10);

ededler[1] = tesadufiEded.Next(0,10);

ededler[2] = tesadufiEded.Next(0,10); EKVİVALENT

ededler[3] = tesadufiEded.Next(0,10);

ededler[4] = tesadufiEded.Next(0,10); for (int i = 1; i < ededler.length; i++)

ededler[5] = tesadufiEded.Next(0,10); {

ededler[6] = tesadufiEded.Next(0,10); ededler[i] = tesadufiEded.Next(0,10);

ededler[7] = tesadufiEded.Next(0,10); }

ededler[8] = tesadufiEded.Next(0,10);

ededler[9] = tesadufiEded.Next(0,10);

foreach (var item in ededler)

{

listBox1.Items.Add(item);

}

* Qeyd: Foreach operatoru əsasən ARRAY və COLLECTİON-lar ilə istifadə edilir.
* Unutma: Foreach taxilində “~~item=10~~” yazmaq mümkün deyil

# DƏRS 5. (İÇ-İÇƏ DÖVR OPERATORU)

* Qeyd: İç-içə dövr qurulan zaman ümumi dövr sayı hər dövrün ayrılıqda olan saylarının bir-biri ilə hasilinə bərabər olur.

## \n açar sözü

“\n” açar sözü mənt ifadənin yeni sətrdən yazılmasını təmin edir. *(Nəzərə almaq lazımdır ki “Asim\nAnar” yazılan zaman iki dırnaq daxilində yazılmasına baxmayaraq öz funksiyasını icra edəcəkdir)*

## Dövr daxilində istifadə edilən keyword-lər ( CONTİUNE, BREAK, RETURN)

Contiune – dövr daxilində seçilmiş olan 1 dövründən atır.

Break – dövr daxilində təyin edilmiş hissədən sonra dövrdən çıxır.

Return – dövr daxilində təyin edilmiş hissədən sonra “event”-dən atır. *(event daxili kodlar işləmir)*

string netice = "";

for (int i = 0; i < 10; i++)

{

if (i == 6)

{

continue;

}

netice += i + " ";

}

MessageBox.Show("Test");

//Test 1 2 3 4 5 7 8 9

string netice = "";

for (int i = 0; i < 10; i++)

{

if (i == 6)

{

break;

}

netice += i + " ";

}

MessageBox.Show("Test");

//Test 1 2 3 4 5

string netice = "";

for (int i = 0; i < 10; i++)

{

if (i == 6)

{

continue;

}

netice += i + " ";

}

MessageBox.Show("Test");

// heç bir nəticə əks olunmur.

# DƏRS 6. (ARRAYS-MASSİVLƏR)

* Qeyd: ARRAY-lərdə index nömrəsi “0”-dan başlanılır.
* Qeyd: ARRAY yaratmaq üçün arraya daxil olacaq elemetləri tipini və sayını bilmək lazımdır. *(bu arraylarin çatışmazlığı hesab edilir.)*

string[] adlar = new string[7];

adlar[0] = "Anar";

adlar[1] = "Asim";

adlar[2] = "Aysel";

adlar[3] = "Amil";

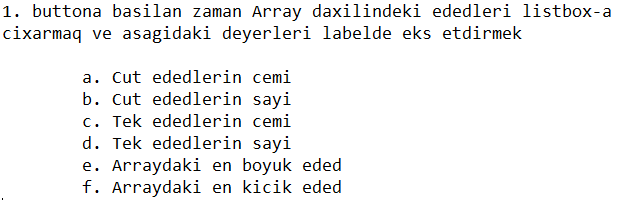
adlar[4] = "Aynur";

adlar[5] = "Ayten";

adlar[6] = "Aylin";

* Qeyd: ARRAY-in uzunluğu ilə max index-i arasında “-1” fərq var.

### TAPŞIRIQ



## Array yaratmağın növləri

class Program

{

static void Main(string[] args)

{

int[] massiv = new int[5]; // birinci yol

massiv[0] = 1;

massiv[1] = 2;

massiv[2] = 3;

massiv[3] = 4;

massiv[4] = 5;

foreach (var item in massiv)

{

Console.WriteLine(item);

}

int[] massiv2 = new[] { 1, 2, 3, 4, 5 }; // ikinci yol

int[] massiv3 = { 1, 2, 3, 4, 5 }; // ucuncu yol

int[] massiv4 = new int[5] { 1, 2, 3, 4, 5 }; // dorduncu yol

Console.ReadLine();

}

}

## Çox ölçülü array

int[,] coxOlculuMassiv = new int[3, 5]

{

{1,2,3,4,5 },

{1,2,3,4,5 },

{1,2,3,4,5 },

};

for (int i = 0; i <= coxOlculuMassiv.GetUpperBound(0); i++)

// setrlerin en boyuk indeksi

{

for (int j = 0; j <= coxOlculuMassiv.GetUpperBound(1); j++)

// sutunlarin en boyuk indeksi

{

Console.WriteLine(coxOlculuMassiv[i, j]);

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*");

}

Console.ReadLine();

* Qeyd: GetUpperBound metodundan istifadə edən zaman massivin deyerlərinin hər birinin əks edilməsi üçün dövr “0”-dan başlayaraq metodun qaytardığı dəyərdən “<=” olmalıdır.

# DƏRS 7. (ARRAY METHODS)

## IndexOf \*\*\*

(Axtarılan elementin index-ni qaytarır)

…

int index = Array.IndexOf(adlar,"Aynur");

* Qeyd: IndexOf Methodu axtarış edən zaman 0-cı indexdən yoxlamağa başlayır. *(Əgər array daxilində axtarılan dəyər tərkar olunarsa, bu zaman dəyərin ilk rastlandığı index əks olunacaqdır)*
* Qeyd: Axtarılan element ARRAY daxilində yoxdursa “-1“ dəyəri geriyə qayıdır.

## LastIndexOf \*\*\*

(Axtarılan elementin index-ni qaytarır)

…

int index = Array.LastIndexOf(adlar,"Aynur");

* Qeyd: Last IndexOf Methodu axtarış edən zaman sonuncu indexdən yoxlamağa başlayır. *(Əgər array daxilində axtarılan dəyər tərkar olunarsa, bu zaman dəyərin ilk rastlandığı index əks olunacaqdır)*

## Contains

(Axtarılan elementin varlığını yoxlayır)

…

bool deyer = adlar.Contains("Aynur");

## Length

(Arrayın uzunluğunu tapır)

…

int uzunluq = adlar.Length;

int MaxIndex = adlar.Length - 1;

## Resize \*\*\*

(Arrayın ölçüsünü artırır)

…

Array.Resize(ref adlar, 8);

adlar[7] = "Ronaldo";

ref => referans, istinad etmək.

## CreateInstance \*\*\*

(Arrayın Method istifadə ederək yaradılması üçün ist edilir.)

Array ad = Array.CreateInstance(typeof(string), 5);

* Qeyd: Method ilə yaradılan massivə element daxil etmək üçün Methoddan istifadə etmək lazımdır.

…

ad.SetValue("adil", 4);

* Qeyd: Method ilə yaradılmış massivə yalnız Method ilə elemet daxil etmək mümkündür.

…

~~Ad[4] = "adil"; (XXX)~~

## SetValue

(Array-ə element daxil etmək üçün istifadə edilir)

…

ad.SetValue("adil", 4);

* Qeyd: Arrayın yaradılma Methodundan asılı olmayaraq SetValue Methodu ilə element daxil etmək mümkündür.

## GetValue

(Arrayın element xaric etmək üçün istifadə edilir)

…

label1.Text = ad.GetValue(4).ToString();

* Qeyd: Arrayın yaradılma Methodundan asılı olmayaraq GetValue Methodu ilə element xaric etmək mümkündür.
* Qeyd: Method ilə yaradılan massivə element daxil etmək və xaric etmək üçün Methoddan istifadə etmək lazımdır.

## Sort \*\*\*

(Massivin elementlərini sıralı hala gətirir)

…

Array.Sort(adlar);

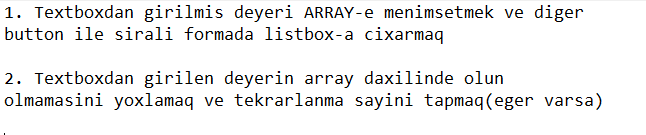
## Reserve \*\*\*

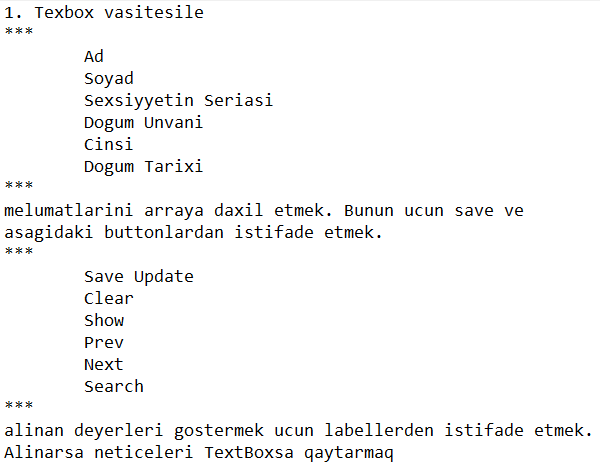
(Massivin elementlərini axırıncı index-dən 0-ya doğru tərsinə çevirir)

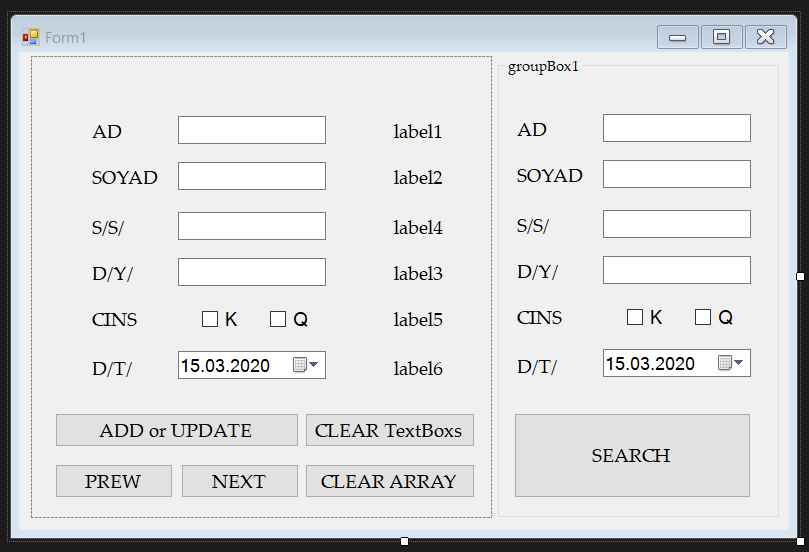
…

Array.Reserve(adlar);

### TAPŞIRIQ







# DƏRS 8. (COLLECTİONS)

ARRAY-lərdən istifadə edən zaman iki tip məhdudluq ( tipin və ölçünün öncədən məlum olması) meydana çıxırdı. Amma bundan fərqli olaraq kənardan gələn dəyərin hər zaman tipini və ölçüsünü müəyyən etmək olmur. Bu problemi aradan qaldırmaq üçün eyni məntiqə malik olan COLLECTİON-lar meydana çıxdı.

Collection-lar iki tipə bölünür: Generic, NonGenerik

## Generic

LİST - (System.Collection.Generic)

## NonGeneric

ARRAYLİST - (System.Collection)

# ARRAYLİST

ArrayList liste = new ArrayList();

liste.Add("Asim");

liste.Add(15);

liste.Add('t');

liste.Add(23.5);

liste.Add(true);

* Qeyd: ArrayList daxilində bütün elemetlər “object” tipində saxlanılır. *(Bütün tiplərin alına bilməsi üçün)* Nəticədə məlumatlar daxil edilən zaman BOXİNG, digər bir tipli dəyərə mənimsədilən zaman isə UNBOXİNG prosesi icra olunur.

object a = 15; // Boxing

int b = (int)a; // Unboxing

* Qeyd: ArrayList-lərin CAPACİTY-si “2n” şəklində hesablanır. ArrayListin başlanğıc Capacity-si “4” bərabərdir. *(Əgər arraylist daxilində 5 elemet olarsa, Count=5 – Capacity = 8 olur)*

# ARRAYLİST METHODS

## Add

(Arraylistə bir-bir element daxil edir.)

…

liste.Add(15);

## AddRange

(Arraylistə elementləri toplu şəkildə əlavə edir)

…

liste.AddRange(liste2);

* Qeyd: Bir ArrayList-dən digərinə məlumat əlavə edir.

## Remove

(İstənilən bir dəyəri silmək üçün)

…

liste.Remove("Asim");

* Qeyd: “void” Methoddur. Geriyə heç bir dəyər qaytarmır.

## RemoveAt

(İstənilən bir index-i silmək)

…

liste.RemoveAt(1);

## RemoveRenge

(Başlanğıc index-dən istənilən simvol qədər silmək)

…

liste.RemoveRange(1, 2);

* Qeyd: İndex-daxil olmaqla silir.

## Insert

(İstənilən index-ə bir element daxil etmək)

…

liste.Insert(1, "Aysel");

* Qeyd: Əgər “ınsert” əməliyatından öncə qeyd edilmiş və sonrakı index-lərdə dəyərlə mövcud olarsa, həmin elementlərin indexləri “+1” olaraq sürüşür.

## InsertRange

(İstənilən index-ə toplu element daxil etmək)

…

liste.InsertRange(1, liste2);

## Count

(Arraylistin elementlərinin sayı)

…

int say = liste.Count;

## Capacity

(Arraylistin tutumu)

…

int tutum = liste.Capacity;

## ToTrimSize

(Boşluqları silir)

…

liste.TrimToSize();

* Qeyd: Count = Capacity olur

## Clear

(Arraylistin elementlərini pozur)

…

liste.Clear();

## CopyTo

(Arraylistdən arraya kopyalayır)

…

liste.CopyTo(adlar);

* Qeyd: Bu əməliyatın icra olunması üçün ArrayList daxilindəki elementlərin eyni tipə malik olmalıdır.

## Sort, Reserve, IndexOf, LastIndexOf, Contains

* Qeyd: “SORT” Methodunun işləməsi üçün ArrayList daxilindəki elemetlər eyni tipə malik olmalıdr.

liste.Sort();

liste.Reverse();

int index = liste.IndexOf("Asim");

int index = liste.LastIndexOf("Asim");

liste.Contains("Asim");

# DƏRS 9 və 10. (LIST və STRING)

# LIST və METHODS

LIST-lər Generic Collection-lar sinifinə daxildir. LIST və ARRAYLIST Methodlar baxımından hər biri eyni məntiqə malikdir. Listləri Arraylistlərdən fərqləndirən başlıca amil gələn dəyərin tipinin öncədən bilinməsidir.

* Qeyd: LIST-lər əsasən class-lar ilə bazadan məlumat çekməsi zamanı geniş istifadə olunur.
* Qeyd: Performans baxımından dəyərləndilərsə, belə bir yanaşma sərgiləmək olar. Əgər toplu şəkildə gələn məlumatların eyni zamanda həm TİPİ, həm də SAYI məlum olarsa ARRAY, yalnız TİPİ müəyyən olarsa LIST, heç bir məlum olmazsa ARRAYLIST istifadə edilməsi daha məqsədə uyğun hesab edilir.

List<string> adlar = new List<string>();

…

adlar.Add("Asim");

…

adlar.AddRange(adlar2);

…

adlar.Remove("Asim");

…

adlar.RemoveAt(1);

…

adlar.RemoveRange(1, 2);

…

adlar.Insert(1, "Aysel");

…

adlar.InsertRange(1, adlar2);

…

int say = adlar.Count;

…

int tutum = adlar.Capacity;

…

adlar.TrimToSize();

…

adlar.Clear();

…

adlar.CopyTo(adlar);

…

adlar.Sort();

…

adlar.Reverse();

…

int index = adlar.IndexOf("Asim");

…

int index = adlar.LastIndexOf("Asim");

…

adlar.Contains("Asim");

# STRING METHODS

* Qeyd: string Referans tipdir.

## ToString

(İstənilən tipi string-ə çevirir)

…

ad.ToString();

## IndexOf

(String daxilində axtarılan char tipli simvolu 0-cı index-dən başlayaraq axtarır)

…

ad.IndexOf('a');

## LastIndexOf

(String daxilində axtarılan char tipli simvolu sonuncu index-dən başlayaraq axtarır))

…

ad.LastIndexOf('a');

## Contains

(Axtarılan simvolun varlığını yoxlayır)

…

ad.Contains('a');

## Length

(Uzunluğu tapır)

…

int uzunluq = ad.Length;

## Trim

(String məlumatın ön və arxa hissəsində olan boşluqları silir)

…

ad.Trim();

* Nümunə: User, Password hissəsində istifadə oluna bilər

## TrimStart

(String məlumatın ön hissəsində olan boşluqları silir)

…

ad.TrimStart();

## TrimEnd

(String məlumatın arxa hissəsində olan boşluqları silir)

…

ad.TrimEnd();

## IsNullOrEmpty \*\*\*

(Daxil olan dəyərin null və ya "" – boş olmasını yoxlayır)

…

string.IsNullOrEmpty(ad);

* UNUTMA: Baza ilə məlumat mübadiləsi zamanı çox vacib funksiyadır.

## IsNullOrWhiteSpace \*\*\*

(Daxil olan dəyərin null və ya " " – boşluq olmasını yoxlayır)

…

string.IsNullOrWhiteSpace(ad);

* Qeyd: Əgər IsNullOrEmpty funksiyası işləyən zaman geriyə qayıdan dəyər (" ")-olarsa, FALSE nəticə verir. Fərqli olaraq əgər IsNullOrWhiteSpace funksiyası işləyən zaman geriyə qayıdan dəyər (" ", null və ya "") olarsa, TRUE nəticə verir.

## ToUpper

(bütün hərflər böyük)

…

ad.ToUpper();

## ToLower

(bütün hərflər kiçik)

…

ad.ToLower();

## Compare \*\*\*

(stringləri müqayisə edir)

Eyni söz olarsa – “0” ;

Eyni uzunluq fərqli sözlər və ya fərqli sözlərdən I söz II sözdən uzun olarsa – “1”

Fərqli sözlərdən II söz I sözdən uzun olarsa – “-1“ dəyəri geri qayıdır.

…

string.Compare(ad, ad2);

## Copy \*\*\*

(string məlumatı başqa dəyərə kopyalamaq)

…

ad2 = string.Copy(ad);

## ToCharArray

(stringi char array-ə çevirmək)

…

ad.ToCharArray();

## Replace

(Simvolu və yaxud simvol toplusunu əvəzləyir)

…

ad.Replace('a', 'b');

## Remove

(Verilən index daxil olmaqla sona kimi və ya index-dən sonra aralıqla silə bilir.)

string SozBirlesmesi = "OxuyanUsaq";

string s = SozBirlesmesi.Remove(5); // Overload 1. Indexden baslayaraq sona kimi silib atir

string s2 = SozBirlesmesi.Remove(5, 2); // Overload 2. Indexden baslayaraq *(index daxil olmaqla)* 2 simvol silib atir

* Qeyd: Remove funksiyası sildiyi hissəni atır. *(Digər hissə mənimsədir.)*

## Concat \*\*\*

(Stringləri birləşdirir.)

…

string.Concat(s3, " ", s4);

## SubString

(Verilən index daxil olmaqla sona kimi və ya index-dən sonra aralıqla kəsib götürür.)

string SozBirlesmesi = "OxuyanUsaq";

string s3 = SozBirlesmesi.Substring(5); // Overload 1. Indexden baslayaraq sona kimi kesib goturur

string s4 = SozBirlesmesi.Substring(5, 2); // Overload 2. Indexden baslayaraq *(index daxil olmaqla)* 2 simvol kesib goturur

* Qeyd: SubString funksiyası sildiyi hissəni özündə mənimsədir digər hissəni atır.

## Split

(String məlumatı verilmiş simvola əsasən bölərək yeni bir massiv əmələ gətirir.)

…

string[] massiv = SozBirlesmesi.Split('u'); // Overload 1.

string[] massiv2 = SozBirlesmesi.Split('u', 'a'); // Overload 2.

## Join \*\*\*

(Verilmiş massivin elementlərini simvola əsasən birləşdirərək vahid string əmələ gətirir.)

string[] adlar = new string[4];

adlar[0] = "Anar";

adlar[1] = "Asim";

adlar[2] = "Aysel";

adlar[3] = "Amil";

string VahidString = string.Join("", adlar);

### TAPŞIRIQ

1. TextBoxdan gələn dəyərin Label-ə tərsinə yazılması

2.

a. TextBoxa AD-SOYAD yazılıb OK tıklananda SOYAD ÇIXSIN

b. TextBoxa AD-SOYAD-ATA ADİ yazılıb OK tıklananda SOYAD ÇIXSIN

3. OK tıklananda massiv yaratsin ve sirali formada listBox yazsin. Ikinci OK tiklayanda birlesmis sekilde yazilsin.

4. TextBox-a daxil edilmiş 3+5 və s. ifadəsinin nəticəsini hesabladıqdan sonra 3+5=8 olaraq nəticəni əks etdirsin.

# DƏRS 11. (RANDOM və TİMER)

## Random

RANDOM – təsadüfi ədələr yaradan bir Methoddur. Romdolar - RANDOM class - nın NEXT Methodundan istidafə edərək verilmiş şərtə uyğun dəyərlər sintez edir.

* UNUTMA: RANDOM-lar hər zaman int dəyər sintez edir.

Random tesadufieded = new Random();

int deyer = tesadufieded.Next();

//overload1 - int tipinin aldigi araliqda deyer qaytarir

int deyer2 = tesadufieded.Next(280);

//overload2 - 0 ve 279 daxil olmaqla deyer qaytarir

int deyer3 = tesadufieded.Next(120, 300);

//overload3 - 120 ve 299 daxil olmaqla deyer qaytarir

* Qeyd: Next Methodunda verilən yuxarı sərhəddi dəyər olaraq geri qaytarmır.

## Timer

Təkrarlanan əməliyyatları zamanla təkrarlanması üçün timerlərdən istifadə edilir*. (İstənilən kodu daxil etmək mümkündür)*

* Qeyd: Timer PASSİV komponentdir. *(İnterfeys daxilində visual olaraq görsənmir)*

int i = 0;

private void button2\_Click(object sender, EventArgs e)

{

timer1.Interval = 1000; // 1 saniyəyə ekvivalentdir

timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

i++;

if (i%2==0)

{

this.BackColor = Color.AliceBlue;

}

else

{

this.BackColor = Color.Cyan;

}

if (i == 60)

{

timer1.Stop();

}

}

### TAPŞIRIQ

1. User-Password (telefon ucun.) ok duymesini basan zaman daxil edilmis melumatlardan hansi sehv olarsa hemin melumatin sehvliyi barede ekrana melumat gelsin. Sehv sayi maksimal 3 olsun. Eger dogru melumat girilerse yeni form acilsin ve yeniden sehv sayi 3 olsun. Eks halda ekrandaki tool-lar enable =false olsun. 20 saniyeden sonar yeniden melumatlar aktiv olsun. Eger 3 defeden artiq 20 san blokuna duserse form.exit olsun. (timer tapsiriq)

2. her gun dukana gedecek adamin adini ekrana cixarsin (random tap)

3. her gun apteke gedecek iki ferqli adamin adini ekrana cixarsin (random tap)

# DƏRS 12. (Try Catch və Dialog Windows)

## Try Catch - Exception

TryCatch bloku xətaların idarə edilməsi mümkün olur. *(Xətaların idarə edilməsi).* Sistem işləyən zaman hər hansı bir xəta baş verərsə, sistemin sönməsi əvəzinə XƏTANIN ÖYRƏNILMƏSI və ya hər hansı bir IŞIN ICRA OLUNMASINI təmin etmək faydalıdır.

try

{

int d1 = Convert.ToInt32(textBox1.Text);

int d2 = Convert.ToInt32(textBox2.Text);

int result = d1 / d2;

label1.Text = result.ToString();

}

catch

{

}

// eger textboxdan inte cevrilmesi mumkun olmayan simvol toplusu gelerse bu zaman FORMAT-EXCEPTİON bas verir. Amma CATCH blokuna hec bir kod yazilmadigi ucun sistem aktiv qalir

* Qeyd: TRY bloku daxilində error olan zaman CATCH blokundakı kod işləməyə başlayır.
* Qeyd: Əgər sistemdə baş verən xəta CATCH-dən sonra yerləşən kod kombinasiyasının işləməsində hər hansı bir həta yaranmasına səbəb ola bilərsə, bu zaman CATCH bloku daxilində RETURN əmrindən istifadə edilə bilər.

try

{

int d1 = Convert.ToInt32(textBox1.Text);

int d2 = Convert.ToInt32(textBox2.Text);

int result = d1 / d2;

label1.Text = result.ToString();

}

catch (Exception ex) //hansi tip xetanin oyrenilmesi ucun “ex” yaxalanir

{

MessageBox.Show(ex.Message);

return; //Error verən zaman TEST yazısı ekrana çıxmayacaqdır.

}

MessageBox.Show("Test");

* UNUTMA: Kod daxilində birdən çox error olarsa, bu zaman ən birinci baş verən erron yaxalanır və həmin error həll olunmamış digər errorları yaxalaya bilmir.
* Qeyd: Əgər qeyd edilərsəki, hər hansı bir EXCEPTİON-nı tutmasın Bu zaman sadecə olaraq yaxalanması lazım olan EXCEPTİON-nun adını bir başa yazaraq onun tutulmasını təmin etmək olar.

TryCatch bloku əsəs hissələrindən biridə FİNALY blokudur. FİNALY bloku daxilində qeyd edilmiş kod hər zaman işləyir. *(Nümunə olaraq əgər ADO.NET -in Connected üsulu ilə məlumat mübadiləsi edilərsə, bu zaman Finaly blokuna connection-nun bağlanılması əmrini yazmaq olar.)*

try

{

int d1 = Convert.ToInt32(textBox1.Text);

int d2 = Convert.ToInt32(textBox2.Text);

int result = d1 / d2;

label1.Text = result.ToString();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return;

}

finally

{

MessageBox.Show("her zaman aktiv qalir");

}

MessageBox.Show("Test");

* Qeyd: FİNALY blokundan əvvəl Return əmri olmağına baxmayaraq ekranda "her zaman aktiv qalir" mesajı əks olunur.

TryCatch bloku Catch hissəsində THROW əmri işlənə bilər. THROW əmrinin məntiqi ondan ibarətdirki, kod daxilində yazılmış səhvi yaxaladıqdan sonra sistem errora düşür və sönür. *(Əsasən ilkin kod yazılışı zamanı sistemdəki səhvləri taparaq aradan qaldırmaq üçün istifadə edilir)*

…

catch (Exception ex)

{

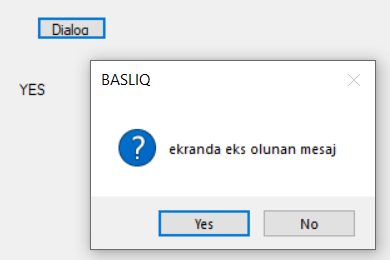
MessageBox.Show(ex.Message);

throw;

}

## Dialog windows

MessageBox istifadə edilən zaman geri qayıdan dəyər tipi “DialogResult”olur. DialogResult tipində dialog pəncərəsi zamanı olan düymələrdən asılı olaraq müxtəlif əməliyatlar icra edilə bilər.

DialogResult dialog = MessageBox.Show

("ekranda eks olunan mesaj", "BASLIQ",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question);

if (dialog == DialogResult.Yes)

{

label1.Text = "YES";

}

else if (dialog == DialogResult.No)

{

label1.Text = "NO";

}

### SAVE File Dialog

SaveFileDialog save = new SaveFileDialog();

save.Title = "Save windows";

save.Filter = "txt files|\*.txt |all files|\*.\*|excel files|\*.xlsx";

save.FileName = "project\_files";

save.OverwritePrompt = true;

save.CreatePrompt = true;

if (save.ShowDialog() == DialogResult.OK)

{

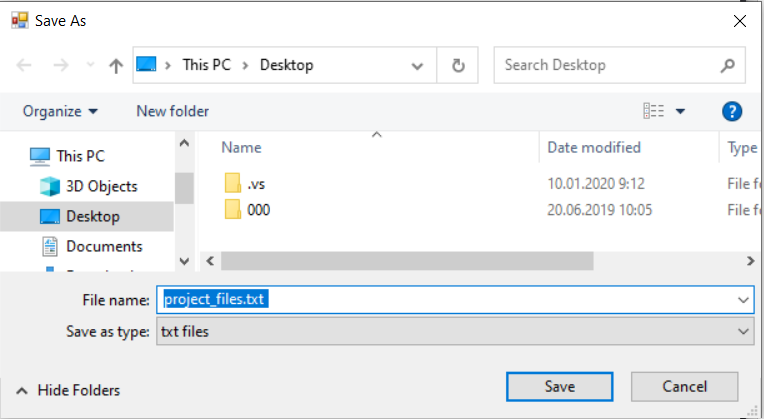
StreamWriter qeydiyyat = new StreamWriter(save.FileName);

qeydiyyat.WriteLine(richTextBox1.Text);

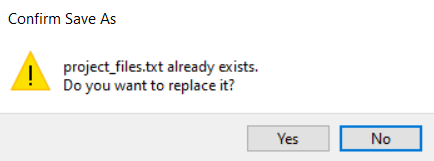
qeydiyyat.Close();

}

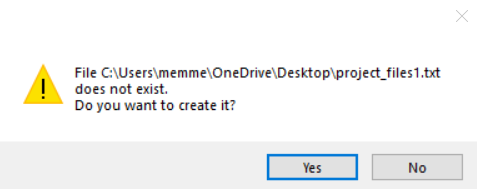
* Save.Filter: File yaddaşa verilən zaman hansı tipdə qeyd edilməsini təyin edir.
* Save.FileName: File-ın yaddaşa yazılan zaman default adını təyin edir.



* Save.OverWritePrompt: File yaradılan zaman eyni adlı File olarsa, eyni adın olması barədə xəbərdarlıq edir.



* Save.CreatePrompt: File yarad;lan zaman default verilmiş addan fərqli ad ilə yaddaşa verilən zaman xəbərdarlıq mesajı ekrana çıxarır. *(yaradılmış file daxilindən məlumat çəkən sistem mövcud olduğu halda əgər fərqli adla qeyd edilərsə, sistemin işləməsində müəyyən xətalar baş verə bilər)*



### OPEN File Dialog

OpenFileDialog open = new OpenFileDialog();

open.Title = "Open windows";

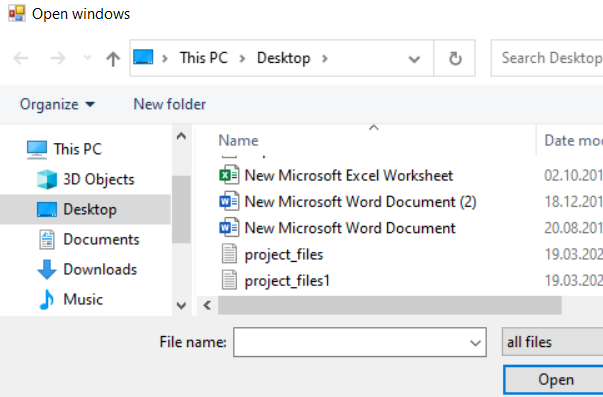
open.Filter = "txt files|\*.txt |all files|\*.\*";

if (open.ShowDialog() == DialogResult.OK)

{

StreamReader oxuma = new StreamReader(open.FileName);

richTextBox1.Text = oxuma.ReadToEnd();

 }

* Open.Title: Açılan pəncərəyə başlıq vermək üçün istifadə edilir.

# DƏRS 13. (DateTime Class, StreamWriter and StreamReader, File Class Methods)

## DateTime Class və DateTimePicker

DateTime class-nnın Now Methodu vasitəsi ilə cari tarix və saatı *(Short, Long)* gəlirmək mümkündür.

private void button1\_Click(object sender, EventArgs e)

{

textBox1.Text = DateTime.Now.ToShortDateString();

textBox2.Text = DateTime.Now.ToShortTimeString();

textBox3.Text = DateTime.Now.ToLongDateString();

textBox4.Text = DateTime.Now.ToLongTimeString();

textBox5.Text = DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss");

//dateTimePicker

label6.Text = dateTimePicker1.Text;

label8.Text = dateTimePicker1.Value.ToString("dd.MM.yyyy hh:mm:ss");

//Timer

timer1.Interval = 1000;

timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

label7.Text = DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss");

}

dd – gün (1-31)

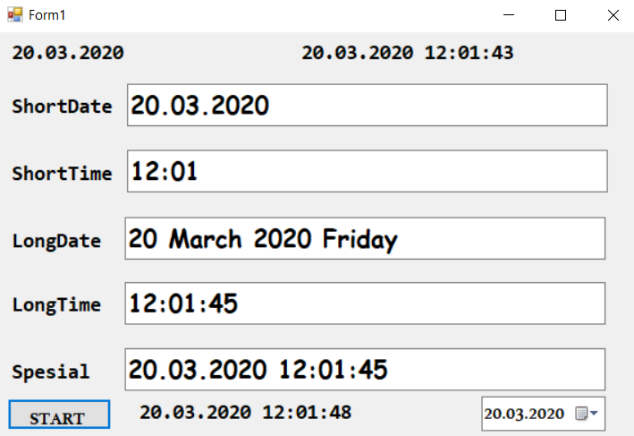
MM – ay (1-12)

yyyy – il (1-2020)

hh – saat (00-24)

mm – dəqiqə (0-60)

ss – saniyyə (0-60)



* Qeyd: DateTimePicker istifadə edilən zaman dəyəri ekrana çıxarmaq üçün TEXT və VALUE Methodundan istifadə edilir. TEXT Methodundan istifadə zamanı sadecə tarix *(ay/gün/il)* VALUE Methodu istifadə edilən zaman default olaraq *(ay/ gün/ il saat/dəqiqə/san)* formatında gəlir*. (koda müdaxilə edərək istəlinən formatda gəlməni təmin etmək olar)*
* UNUTMA: Proyektlər daxilində yerləşdirilmiş DateTimePicker tool-u baza məlumat ötürməkdə istifadə edilərsə, onun ayarlar hissəsində görünüşü SHORT tipinə keçirmək lazımdır. Əks halda Bazada DATE tipi verilmiş sütuna məlumat mübadiləsi problemi yaranacaqdır.

## StreamWriter

(File daxilinə məlumat əlavə edir)

* Qeyd: StreamWriter vasitəsilə fayl daxilinə məlumat qeyd edən zaman öncəki məlumatı silərək yeni məlumatı yada saxlayır.

string filename = textBox1.Text;

string text = richTextBox1.Text;

...

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

StreamWriter SW = new StreamWriter(path); // geriye deyer qaytarmayan Method

SW.WriteLine(text);

SW.Close();

MessageBox.Show("UGURLU");

Əgər yazılmış məlumatı daxil edən zaman hər dəfə eyni path-ünvanda yerləşən file-lar əvəzinə digər file-a məlumat daxil etmək tələb olunarsa SaveFileDialog class-dan istifadə etmək olar.

...

SaveFileDialog save = new SaveFileDialog();

save.Title = "BASLIQ";

save.Filter = "ment fayl|\*.txt";

if (save.ShowDialog() == DialogResult.OK)

{

StreamWriter SW = new StreamWriter(save.FileName);

SW.WriteLine(text);

SW.Close();

MessageBox.Show("UGURLU");

}

## StreamReader

(File daxilindəki məlumatı çekməkir)

string filename = textBox1.Text;

string text = "";

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

StreamReader SR = new StreamReader(path);

text = SR.ReadToEnd();

richTextBox1.Text = text;

Əgər içəriyinə baxılmalı olan file-ın adı unudulmuş olarsa bu zaman SaveOpenDialog class-nın Methodlarından istifadə etmək olar.

...

OpenFileDialog od = new OpenFileDialog();

od.Filter = "ment fayl|\*.txt";

if (od.ShowDialog() == DialogResult.OK)

{

StreamReader SR = new StreamReader(od.FileName);

text = SR.ReadToEnd();

richTextBox1.Text = text;

}

## FileClassMethods

### WriteAllText

StreamWriter class-da olduğu kimi WriteAllText Methodu da köhnə Methodu silərək əvəzində yeni məlumatı qeyd edir.

string filename = textBox1.Text;

string text = richTextBox1.Text;

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

File.WriteAllText(path, text);

MessageBox.Show("Write");

### AppendAllText

AppendAllText Methodu WriteAllText Methodundan fərqli olaraq yazılan yeni məlumatı köhnə məlumatı pozmadan onun sonuna əlavə edir.

string filename = textBox1.Text;

string text = richTextBox1.Text;

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

File.AppendAllText(path, text + "\r\n");

MessageBox.Show("Append");

### ReadAllText

StreamReader class-da olduğu kimi ReadAllText Methodu da məlumatların çekilməsi üçün istifadə edilir.

string filename = textBox1.Text;

string text = "";

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

text = File.ReadAllText(path);

richTextBox1.Text = text;

### Exists

Qeyd edilən path-a uyğun olaraq həmin adda file olub-olmamasını yoxlayır.

string filename = textBox1.Text;

string path = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

bool netice = File.Exists(path);

MessageBox.Show(netice.ToString());

### Move

1. Mövcud file-ı EYNİ adla fərqli qovluqda qeyd edə bilər. *(bu zaman SOURCE olan file silinir)*

string filename = textBox1.Text;

string path1 = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

string path2 = @"C:\Users\memme\source\log\" + filename + ".txt";

File.Move(path1, path2);

MessageBox.Show("Move");

2. Mövcud file-ı YENİ adla cari qovluqda qeyd edə bilər. *(bu zaman SOURCE olan file silinir)*

string filename = textBox1.Text;

string newfilename = textBox2.Text;

string path1 = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

string path2 = @"C:\Users\memme\source\log\ders13\" + newfilename + ".txt";

File.Move(path1, path2);

MessageBox.Show("Move");

3. Mövcud file-ı YENİ adla fərqli qovluqda qeyd edə bilər. *(bu zaman SOURCE olan file silinir)*

string filename = textBox1.Text;

string newfilename = textBox2.Text;

string path1 = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

string path2 = @"C:\Users\memme\source\log\" + newfilename + ".txt";

File.Move(path1, path2);

MessageBox.Show("Move");

### Replace

Mövcud iki file arasında SOURCE seçilən file-ın *(path1)* daxilindəki məlumatı ikinci file *(path2)* adı altında yaddaşa verir və ikinci file-ı təyin edilmiş yeni ad *(path3)* altında yadda saxlayır.

string filename = textBox1.Text;

string otherfile = textBox2.Text;

string path1 = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

string path2 = @"C:\Users\memme\source\log\ders13\" + otherfile + ".txt";

string path3 = @"C:\Users\memme\source\log\ders13\"+ otherfile +

"BackUpFile.txt";

File.Replace(path1, path2, path3);

MessageBox.Show("Replace");

# DƏRS 14. (Directory Class)

## CreateDirectory

(Qovluq yaradır)

string folderName = textBox1.Text;

Directory.CreateDirectory(@"C:\Users\memme\source\Directory\" +folderName);

MessageBox.Show("Test");

## Delete

(Qovluğu silir)

string folderName = textBox1.Text;

Directory.Delete(@"C:\Users\memme\source\Directory\" + folderName);

MessageBox.Show("Test");

## Exists

(Qovluğun varlığını yoxlayır)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

bool varliq = Directory.Exists(path);

if (varliq) { textBox2.Text = "Var";}

else { textBox2.Text = "Yoxdur";}

## GetCreationTime

(Qovluğun yaranma tarixini göstərir)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

DateTime tarix = Directory.GetCreationTime(path);

MessageBox.Show(tarix.ToString());

// ToString (dd-MM-yyyy) yazilaraq xwsusi formada vere biler

## GetCurrentDirectory

(Cari proqramın hansı ünvanda yerləşdiyini göstərir)

string path = Directory.GetCurrentDirectory();

MessageBox.Show(path);

## GetFiles

(Qovluq daxilindəki file-ların adlarını göstərir)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

string[] qovluqlar = Directory.GetFiles(path);

listBox1.Items.AddRange(qovluqlar);

## GetLastAccessTime

(Qovluq daxilində ən son nə zaman hər hansı bir iş görüldüyünü göstərir. *Qovluqa daxil olub-çıxmaq aid deyil.*)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

DateTime tarix = Directory.GetLastAccessTime(path);

MessageBox.Show(tarix.ToString());

## Move

1. Mövcud qovluğu EYNİ adla fərqli üst qovluqda qeyd edə bilər.

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

string path2 = @"C:\Users\memme\source\" + folderName1;

Directory.Move(path, path2);

MessageBox.Show("Test");

2. Mövcud qovluğu YENİ adla cari qovluqda qeyd edə bilər.

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

string folderName2 = textBox2.Text;

string path2 = @"C:\Users\memme\source\Directory\" + folderName2;

Directory.Move(path, path2);

MessageBox.Show("Test");

3. Mövcud qovluğu YENİ adla fərqli qovluqda qeyd edə bilər.

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

string folderName2 = textBox2.Text;

string path2 = @"C:\Users\memme\source\" + folderName2;

Directory.Move(path, path2);

MessageBox.Show("Test");

* Qeyd: Move əməliyatı zamanı SOURCE olan əməliyatdan asılı olaraq əvəzlənir

## GetLogicalDrives

(Komputer daxilindəki məntiqi disləri gətirir)

string[] mentiqidiskler = Directory.GetLogicalDrives();

listBox1.Items.AddRange(mentiqidiskler);

## GetParent

(Adı qeyd edilmiş qovluğun yerləşdiyi ünvanı göstərir)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

DirectoryInfo melumat= Directory.GetParent(path);

MessageBox.Show(melumat.ToString());

## GetDirectories

(Adı qeyd edilmiş qovluğun daxilindəki qovluqların adını göstərir)

string folderName = textBox1.Text;

string path = @"C:\Users\memme\source\Directory\" + folderName;

string[] qovluqlar = Directory.GetDirectories(path);

listBox1.Items.AddRange(qovluqlar);

# DƏRS 14. (Logs – [AppendAllText, StreamWriter, Nlog] )

Loglar icra olunan prosesləri qeyd etmək üçün istifadə edilir. Logların olması baş vermiş hadisələrin senari ardıcıllığına baxaraq təyin edilməsini və ya server üzərində olan proyektdə baş vermiş xətaların təyini və yaxud nəzarət prosesində faydalı hesab edilir.

#### Qeyd: Log-lar üzrə file yaradan zaman gələn məlumatların həçmindən asılı olaraq və daha tez məlumat eldə etmək üçün günlük və ya həftəlik file formatında yaratmaq daha məqbul hesab edilir. *(File adlandıran zaman Tarixdən istifadə etmək olar*

## AppendAllText

string fileName = DateTime.Now.ToString("dd--MM--yyyy") + " AppendAllTextMethod.txt";

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, "\n" + DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " proses START\r\n");

try

{

string userName = textBox1.Text;

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " <" + userName + "> userName SET\r\n");

int password = Convert.ToInt16(textBox2.Text);

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " <" + password + "> password SET\r\n");

if (userName == "admin" && password == 12345)

{

MessageBox.Show("Success Result");

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " <" + userName + "> <" + password + "> Login SUCCESS\r\n");

}

else

{

MessageBox.Show("Failed ");

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " <" + userName + "> <" + password + "> Login FAILED \r\n");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

File.AppendAllText(@"C: \Users\memme\source\log\ders15\" + fileName, DateTime.Now.ToShortDateString() + " " +

DateTime.Now.ToLongTimeString() + " <" + ex.Message + "> ERROR ocured\r\n");

}

## Nlog

Nlog tool paketi ilk öncədən visual studio yuklenen zaman əlavə edilə və ya sonradan proyektə uyğun olaraq aşağıdakı addımlarla yüklənə bilər.

SOLUTİON => Manage NuGet Packages for Solution => Browse => Nlog and Nlog.Config (install)

Nlog tool paketi yükləndikdən sonra SOLUTİON pəncərəsinə əlavə olunmuş nlog.config hissəninə daxil olaraq ilk öncə TARGETS sonra isə RULES hissələrində proyektə uyğun ayarlamalar aparılmalıdır.

<targets>

<target xsi:type="File" name="f" fileName="${basedir}/logs/${date:format=dd.MM.yy} Nlog.log" layout="${uppercase:${level}} ${message}" />

</targets>

<rules>

<logger name="\*" minlevel="Debug" writeTo="f" />

</rules>

#### AÇIQLAMA: type - hissəsindən log yazılacaq yeri *(file, db, mail və s.)* seçilir.

#### AÇIQLAMA: fileName – log qovluğunun yerləşməşdiyi ünvamı və adını təyin edir*. ($-işarəsi References-dəki Nlog kitabxanasındakı əmrləri işlətmək üçün istifadə edilir. {basedir} proyektin qovluğu daxilindəki logs qovluqu açaraq {date:format=dd.MM.yy} Nlog adını verəcəyini bildirir.)*

#### Qeyd: Nlog daha təkmil hesab edilir və böyük proyektlərdə istifadə edilir.

Logger log = LogManager.GetLogger("\*"); // Logger log = LogManager.GetCurrentClassLogger();

log.Info(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " " + "Login START");

try

{

string userName = textBox1.Text;

log.Debug(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " <" + userName + "> username SET");

int password = Convert.ToInt16(textBox2.Text);

log.Debug(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " <" + password + "> password SET");

if (userName == "admin" && password == 12345)

{

MessageBox.Show("Success Result");

log.Debug(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " <" + userName + "> and <" + password + "> Login SUCCESS\n");

}

else

{

MessageBox.Show("Failed ");

log.Debug(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " <" + userName + "> and <" + password + "> Login FAILED\n");

}

}

catch (Exception ex )

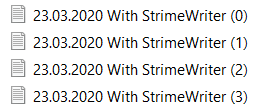
{

MessageBox.Show(ex.Message);

log.Debug(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " <" + ex.Message + "> ERROR ocured\n");

}

## StreamWriter



#### Qeyd: StreamWriter class-dan istifadə edərək log-larsın hazırlanması mümkündür. Lakin bu halda log məlumatlarının hər təkrar icrada fərqli file daxilində yazılmasını nəzərə almaq lazımdır.

int counter = 0;

private void button2\_Click(object sender, EventArgs e)

{

string fileName = DateTime.Now.ToString("dd.MM.yyyy") + " With StrimeWriter (" + counter++ + ").txt";

StreamWriter write = new StreamWriter(@"C: \Users\memme\source\log\ders15\" + fileName);

write.WriteLine(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " START");

int v1 = 0;

int v2 = 0;

int r = 0;

try

{

v1 = Convert.ToInt32(textBox1.Text);

v2 = Convert.ToInt32(textBox2.Text);

r = v1 \* v2;

label1.Text = r.ToString();

}

catch (Exception ex)

{

write.WriteLine(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + ex.Message + " SEHVI BAS VERDI");

write.Close();

}

write.WriteLine(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " birinci deyer " + v1);

write.WriteLine(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " ikinci deyer " + v2);

write.WriteLine(DateTime.Now.ToString("dd.MM.yyyy hh:mm:ss:") + " netice " + r);

write.Close();

}

# DƏRS 16. (Methods )

Methodlar proyeklərin daxilində görülən işləri bir hissədə yazaraq istifadə etmək üçün yaradılır. Methodların Class-lar daxilində yaradılır və yaradılması üç əsas məsələni həll etmək üçün nəzərdə tutulur:

1. Performansın artırılmasi. *(Misal olaraq qeyd etmək olar ki, cari form class-ı daxilində performans baxımından böyük iş görən bir necə hissə olarsa, və birinin işləməsi digərinin işləməsinə hər hansı bir təsirili olmadığına baxmayaraq eyni yerdə yer almasına görə performansın aşa düşməsinə səbəb olur. Methodlar halında yazılması zamanı isə konkret olaraq hansı kodun icrası tələb olunarsa onu çağıraraq, yalnız onun icrası təmin edilir.)*
2. Təkrar kod yazmasın qarşısını almaq. *(Misal olaraq hər hansı proyekt daxilində bir arifmetik operator 100 və daha çox yerdə təkrar edilir. Zaman içində bu əməliyyatın başqa bir arifmetik operatorla əvəz etmək üçün hər yerdə bir-bir dəyişmək lazım gəlir. Amma həmin operator Method daxilində yazılmış olarsa, sadecə Methodda dəyişiklik etməklə məsələni həll etmək olur.)*
3. Xətaların idarə edilməsi. *(Bir əvvəlki nümunəyə uyğun olaraq, əgər təkrar yazılan kod hər hansı bir işləmə xətası verərsə, onu təkrar təkrar dəyişmək əvəzinə son tələbi almaq üçün Method üzərində xətaları idarə etmək daha effektivdir.)*

#### Qeyd: Methodlar iki yerə bölünür: Geriyə Dəyər Qaytaran, Geriyə Dəyər Qaytarmayan

#### Qeyd: Geriyə Dəyər Qaytaran və Qaytarmayan Methodlar hər biri öz-özünə iki yerə bölünür: Parametrli, Parametrsiz (boş)

#### Qeyd: Methodlar button daxilində yox qlobal sahədə yaradılmalıdır. *(Yaradılan Form Class-dır)*

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//Methods

// 1. geriye deyer qaytrarmayan bos

void yaz()

{

MessageBox.Show("Test");

}

// 2. geriye deyer qaytramayan parametrli

void yazdir(string msj)

{

MessageBox.Show(msj);

}

// 3. geriye deyer qaytaran bos

int topla()

{

int cavab = 12 + 45;

return cavab;

}

// 4. geriye deyer qaytaran parametrli

int vurma(int deyer1, int deyer2)

{

int netice = deyer1 \* deyer2;

return netice;

}

private void button1\_Click(object sender, EventArgs e)

{

yaz();

}

private void button2\_Click(object sender, EventArgs e)

{

string metn = textBox1.Text;

yazdir(metn);

}

private void button3\_Click(object sender, EventArgs e)

{

int netice = topla();

label1.Text = netice.ToString();

}

private void button4\_Click(object sender, EventArgs e)

{

int d1 = Convert.ToInt32(textBox1.Text);

int d2 = Convert.ToInt32(textBox2.Text);

int netice = vurma(d1,d2);

label1.Text = netice.ToString();

}

}

#### Qeyd: Methodun tipi ilə return edilən dəyərin tipi eyni olmalıdır.

#### Qeyd: Giriş kəmiyyətinə malik metodlarda Default dəyər vermək mümkündür. Default dəyər həmin sahəyə bir giriş parametri daxil edilmədiyi zaman avtomatik olaraq dəyərə mənimsədilər.

void yazdir(string msj = ("Hello World")

{

MessageBox.Show(msj);

}

#### UNUTMA: Default dəyər verilən zaman son parametrdən başlayaq verilməlidir. *(öndəki parametrə default olaraq dəyər verib digərinə verməmək olmaz, amma əksi mümkündür)*

### TAPŞIRIQ

1. TextBoxdan girilmiş iki dəyər arasında cüt ədədlərin Sayı və Cəmi, Tək ədədlərin Sayı və Cəmini hesablayan Methodlar.

2. 1 – 20 arasındakı ədədlərin Faktorialını və Kvadratlarını hesablayan Methodlar - Listbox-a yazılsın.

3. Verilmiş deyerin Cüt olub olmamasını yoxlasın. *Təkminləşdirmək olar.*

## ref - Keyword- ü

Bu açar söz vasitəsi ilə aşağıdakı nümunədə göstərildiyi kimi kənardan verilən dəyər metod daxilinə referans dəyər kimi verildiyi üçün metod daxilində dəyişik baş verən zaman o dəyərdə dəyişir. Digər dəyər də isə bu hadisə baş verməz.

namespace Methods

{

class Program

{

static void Main(string[] args)

{

int dey1 = 20;

int dey2 = 30;

var result = Topla(ref dey1,dey2);

Console.WriteLine(result);

Console.WriteLine(dey1);

Console.ReadLine();

}

static int Topla(ref int d1, int d2)

{

d1 = 100;

return d1 + d2;

}

}

}

Bu formatda yazılmağı başqa sözlə ifadə etsək RAM daxilində həm metod daxili “d1,d2” həmdə kənar “dey1, dey2” yeyerləri yaradılması əvəzinə “d2,dey1,dey2” olmaqla üç ədəd dəyər yaradılır*. (Dəyər tipinin referans tip kimi istifadə edilməsini təmin edir.)*

## out - Keyword- ü

Məntiq olaraq ref açar sözü ilə eyni məntiqə malikdir. Aradakı fərq ondan ibarətdir ki istifadə edildikdə paramaterə dəyər verməsəkdə kodlarda “error” verməyəcəkdir. Amma əsasə məsələ Out olaraq göndərilən dəyər mütləq formada metod daxilində ən azı bir dəfə set edilməlidir.

static void Main(string[] args)

{

int dey1;

int dey2 = 30;

var result = Topla(out dey1,dey2);

Console.WriteLine(result);

Console.WriteLine(dey1);

Console.ReadLine();

}

static int Topla(out int d1, int d2)

{

d1 = 100;

return d1 + d2;

}

# DƏRS 17. (Method’s Overload and Class )

## Class

Class-ların istifadə məqsədi performansın artırılmasını təmin etməkdir. Nəzərə almaq lazımdır ki, böyük çaplı proyekt yaradan zaman bütün kodları FORM daxilində yazmaq proyektin işləməməsinə belə gətirib çıxara bilər. Bu halların qarşısını almaq üçün mütləq formada siniflər və onların daxilində PROPERTİES və METHODS-lardan istifadə etmək lazımdır.

#### Tərif: Class özəlliklər(PROPERTİES) və funksiyalardan(METHODS) təşkil olunmuş obyektdir.

class user

{

string name;

string surname;

string birth\_place;

int birth\_year;

int YasiQaytar(int year)

{

int result = DateTime.Now.Year - birth\_year;

return result;

}

}

Class daxilində yaradılmış olan method və propertiesləri digər hissələrdə görmək üçün ilk növbətə həmin class-ın obyekti yaradılır. Obyekt yaradıldıqdan sonra əgər yənə properties və methodlar görülməzsə, bu zaman “access modifiers”-ları PUBLİC etmək tələb olunur. *(Devoloperin seçimindən aılı olaraq nələrin access mofifiers-nin dəyişməsi qərarına gələ bilər.)*

class user

{

public string name;

public string surname;

public string birth\_place;

public int birth\_year;

public int YasiQaytar(int year)

{

int result = DateTime.Now.Year - birth\_year;

return result;

}

}

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// set proses

user us = new user();

us.name = textBox1.Text;

us.surname = textBox2.Text;

us.birth\_place = textBox3.Text;

us.birth\_year = Convert.ToInt16(textBox4.Text);

int yas = us.YasiQaytar(us.birth\_year);

// get proses

label1.Text = us.name;

label2.Text = us.surname;

label3.Text = us.birth\_place;

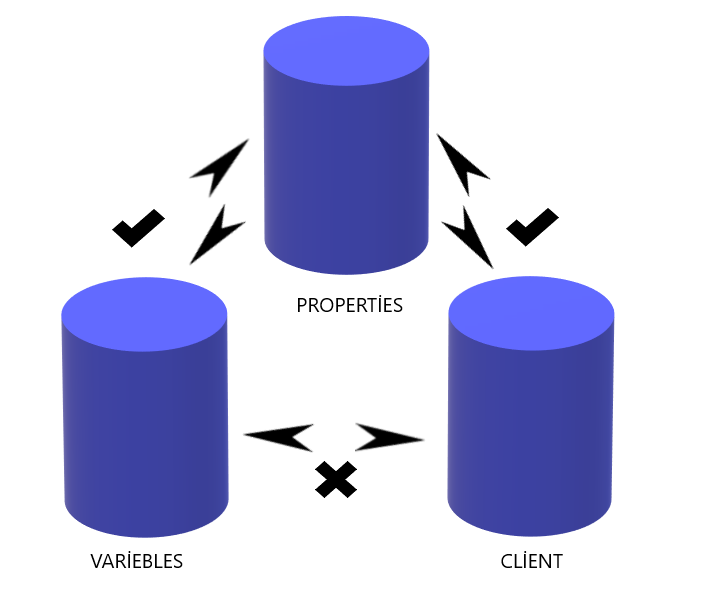
label4.Text = yas.ToString();

}

}

#### Qeyd: Kitabxanaların içində class-lar, class-ların içində isə method-lar *(bir və daha çox)* olur. K => C=> M

Yuxarıda qeyd edikmiş kod ardıcıllığının doğru şəkildə işləməsinə baxmayaraq OOP prinsiplərinə əsasən yazılış xətasına yol verilmişdir. Belə ki, kənardan gələn dəyərin set olunma prosesi zamanı bir başa VARIABLES-a (FİELD Name) mənimsətmək olmaz. Dəyər ilk növbətə Properties-lərə, Propertieslərdən isə Variables-lara set olunmalıdır.



#### Qeyd: Field-dən properties yaratmaq üçün klavyedən ctrl+R+E konbinasiyası və ya prop+TAB+TAB konbinasiyasını icra etmək lazımdır.

Field-lərə uyğun properties-lər yaranan zaman field-lərin hər birinin access modifiers-i private olaraq dəyişir və client-la olan birbaşa əlaqəsi kəsilir.

class user

{

#region Variables (fields)

private string name;

private string surname;

private string birth\_place;

private int birth\_year;

#endregion

#region Properties

public string Name { get => name; set => name = value; }

public string Surname { get => surname; set => surname = value; }

public string Birth\_place { get => birth\_place; set => birth\_place = value; }

public int Birth\_year { get => birth\_year; set => birth\_year = value; }

#endregion

#region Methods

public int YasiQaytar(int year)

{

int result = DateTime.Now.Year - Birth\_year;

return result;

}

#endregion

}

## Overload

Overload metodların əlavə yüklənməsi deməkdir. Overload əməliyyatı həm tipə görə, həmdə parametrlərə görə dəyişiklik etmək olur. Metodların overloadını yaratmaq üçün eyni adlı metodlar yaradılaraq onların tip və ya parametr dəyişikliyini etmək tələb olunr.

public int vurma(int d1, int d2)

{

int netice = d1 \* d2;

return netice;

}

public int vurma(int d1, int d2, int d3)

{

int netice = d1 \* d2 \* d3;

return netice;

}

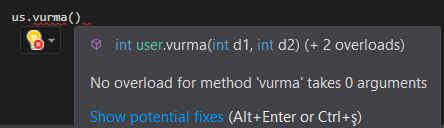
public int vurma(int d1, int d2, int d3, int d4)

{

int netice = d1 \* d2 \* d3 \* d4;

return netice;

}



## params - Keyword- ü

Metodların overload edilməsi zamanı əgər tələb bir neçə yox daha çox olarsa, və hər birinə uyğun bir metod əlavə etmək yanlış olar. Bu səbəbdən qeyd edilmiş açar sözdən istifadə edilir.

static void Main(string[] args)

{

Console.WriteLine(Cemle(1));

Console.WriteLine(Cemle(1,2));

Console.WriteLine(Cemle(1,2,3));

Console.WriteLine(Cemle(1,2,3,4));

Console.WriteLine(Cemle(1,2,3,4,5));

Console.ReadLine();

}

static int Cemle(params int[] number1)

{

return number1.Sum();

}

//əvvəldə adi parametr, sonradan param istifadəsi.

static int Cemle(int number1, params int[] numbers)

{

return numbers.Sum();

}

//aşağıdakı hall mümkün deyil

static int Cemle(int number1, params int[] numbers, int number1)

{

return numbers.Sum();

}

# DƏRS 18. (OOP – Encapsilation, inheritence )

OOP- obyekt yönlü proqramlaşdırma mövcud proqramların real həyata uyğunlaşdırılmasını təmin edir. OOP-nin müxtəlif nəzəriyyələri var və bu nəzəriyyələr əsasında uyğunlaşdırma prosesi həyata keçirilir.

## Encapsilation

Real həyata uyğunlaşdırma dedikdə kənar təsirləri nəzərə alaraq yekun formada işləməni təmin edə bilmək üçün Encapsilation-dan istifadə edilir. Başqa sözlə desək kənar təsirlərdən izoliyasiya olunur. *(Misal olaraq daxil edilmiş məlumatların bazana böyük hərflərlə olmasını təmin etmək; Maaş, yaş və s. Kimi məlumatların mənfi yazılmasının qarşısını almaq və s. )*

//OOP- Encapsilation

public string Name { get => name; set => name = value.ToUpper(); }

// böyük hərflər

public string Surname { get => surname; set => surname = value.ToUpper(); }

// böyük hərflər

public string Birth\_place { get => birth\_place; set => birth\_place = value; }

public int Birth\_year { get => birth\_year; set => birth\_year =Math.Abs(value); }

// Müsbət ədəd

## İnheritance

Digər class-dan miras almaq. Bir class başqa bir classdan həm properties-ni, həm də methods-ları miras ala bilər. Miras verən BASE class, miras alan SAB class adlanır.

Bütün classlar daxilində təkrarlanan məlumatları hər class-da ayrı ayrı yazmaq əvəzinə digər bir class-da yazaraq həmin classdan digərlərinin miras almasını təmin etmək olar.

Base Class

class Car

{

public string Marka { get; set; }

public string model { get; set; }

public string reng { get; set; }

public double motor { get; set; }

public double AtGucu(double motorhecmi)

{

double netice = motorhecmi \* 100;

return netice;

}

}

Sab Classes

class BMW : Car

{

public string mpackage { get; set; }

}

class Audi : Car

{

public string hpackage { get; set; }

}

#### Qeyd: BMV və AUDİ class-ları Car class-dan miras alıb. (BMW : Car, Audi : Car)

Məlumatların görülməsi üçün ilk növbədə miras verilmə prosesi icra olunur. Sonrakı mərhələdə Miras almış class-ın obyekti yaradılır və Base class-ın məlumatlarını görmə icazəsi yaranır.

private void button1\_Click(object sender, EventArgs e)

{

BMW bmw = new BMW();

bmw.Marka = textBox1.Text;

bmw.model = textBox2.Text;

bmw.reng = textBox3.Text;

bmw.motor =Convert.ToDouble(textBox4.Text);

bmw.mpackage = "M5";

double atgucu = bmw.AtGucu(bmw.motor);

label1.Text = bmw.Marka;

label2.Text = bmw.model;

label3.Text = bmw.reng;

label4.Text = bmw.mpackage;

label5.Text = bmw.motor.ToString();

label6.Text = atgucu.ToString();

}

private void button2\_Click(object sender, EventArgs e)

{

Audi audi = new Audi();

audi.Marka = textBox1.Text;

audi.model = textBox2.Text;

audi.reng = textBox3.Text;

audi.motor = Convert.ToDouble(textBox4.Text);

audi.hpackage = "P5";

double atgucu = audi.AtGucu(audi.motor);

label1.Text = audi.Marka;

label2.Text = audi.model;

label3.Text = audi.reng;

label4.Text = audi.hpackage;

label5.Text = audi.motor.ToString();

label6.Text = atgucu.ToString();

}

# DƏRS 19. (OOP – Polymorphyzm, Constructor metodu, static-nonstatic class və metodlar, Access Modifires )

## Override

Polymorphyzm – metodların əzilməsi və ya çoxbiçimlik funksiyasının verilməsidir.

class Product

{

public decimal price { get; set; }

public virtual decimal GetEDV(decimal qiymet)

{

decimal netice = qiymet \* 0.18M;

return netice;

}

}

Bir metodun əzilməsi üçün ilkin olaraq həmin metodun qarşısına mütləq formada VIRTUAL açar sözü yazılmalıdır, sonrakı mərhələdə (haradakı metodun ezilmesi- OVERRIDE baş verəcək) əzilən metodu çağıran class-ların Base class-dan miras alması təmin edilir. Bundan sonra hemin metodun virtual açar sözü override ilə əvəzlənir və metodun son halı yazılır.

class NoteBook:Product

{

public override decimal GetEDV(decimal qiymet)

{

decimal netice = qiymet \* 0.10M;

return netice;

}

}

class MobilePhone:Product

{

public override decimal GetEDV(decimal qiymet)

{

decimal netice = qiymet \* 0.15M;

return netice;

}

}

Son mərhələdə form daxilində OVERRIDE olunmuş metodlar olan class-ların obyektləri yaradılır və metodlar istifadə edilir.

private void button1\_Click(object sender, EventArgs e)

{

MobilePhone mp = new MobilePhone();

mp.price = Convert.ToDecimal(textBox1.Text);

decimal sonQiymet = mp.GetEDV(mp.price);

label1.Text = sonQiymet.ToString();

}

private void button2\_Click(object sender, EventArgs e)

{

NoteBook nb = new NoteBook();

nb.price = Convert.ToDecimal(textBox1.Text);

decimal sonQiymet = nb.GetEDV(nb.price);

label2.Text = sonQiymet.ToString();

}

## Constructor metodu.

Constructor metodu yaradılan hər class-ı yaradan kimi avtomatik olaraq yaradılır və class daxilində ən birinci işləyən metoddur. Constructor metodu yaradılan zaman mütləq formada class ilə eyni adda yaradılmalıdır. Constructor metodunun aşağıdakı qaydalaırı mövcuddur:

* Constructor metodu geriyə dəyər qaytarmır.
* Geriyə dəyər qaytarmayan metod olsada VOID açar sözü istifadə edilmir.
* Yazıldığı yerdən asılı olmayaraq ən birinci işləyən metoddur.
* Constructor metodunun adı class-ın adı ilə eyni olmalıdır.

class Constructor

{

public Constructor()

{

System.Windows.Forms.MessageBox.Show("constructor metod");

}

}

#### Qeyd: Constructor metod digər metodlar kimi çağırılmır. Class-ın obyekti yaradılan zaman avtomatik olaraq işə düşür.

class Constructor

{

public Constructor(string metn)

{

System.Windows.Forms.MessageBox.Show(metn);

}

public void msj()

{

System.Windows.Forms.MessageBox.Show("void metod");

}

}

private void button1\_Click(object sender, EventArgs e)

{

string mtn = textBox1.Text;

Constructor cs = new Constructor(mtn);

cs.msj();

}

#### Qeyd: Constructor metoduna parametr göndərilən vaxt həmin parametr class-ın obyekti yaradılanda obyekt daxilinə göndərilir.

#### UNUTMA1: Constructor metodutda OVERLOAD MÜMKÜNDÜR.

#### UNUTMA2: Metodlar daxildə işlədilirsə static edilə bilər. Kənara gedən bütün metodlar isə bir mənalı şəkildə static olmamalıdır. *(Səbəb – Static metodkar RAM-da bir dəfəlik yer ayırır və belə olan təqdirdə kənara verilmiş metod hər dəfə static yer ayırmalıdır. Əgər yer ayıra bilməzsə sistemin işləməsində xəta verəcəkdir)*

## Static və NonStatic class yaxud metod anlayışı.

Metodlar və class-lar yaradılan zaman default olaraq NonStatic formada olur. Static metodlar dəyişməyən, əsasən sabit işləri görən metodlara aid edilir. Static class-lara isə misal olaraq, base class-dan məlumat və metod alaraq işləyən bir class-ı static etmək olar. Static yaradılan class-ın və ya metodun ÜSTÜNLÜYÜ class-ın obyektini yaratmadan ona bir başa müraciət etmənin mümkün olmasıdır.

class StaticAndNonStatic

{

public static int topla (int deyer1, int deyer2)

{

int result = deyer1 + deyer2;

return result;

}

public int vurma(int deyer1, int deyer2)

{

int result = deyer1 \* deyer2;

return result;

}

}

#### Qeyd: Class daxilində static və NonStatic metodlar varsa və onları çağırmaq tələb olunursa, o zaman static metod bir başa class-ın adını yazaraq çağrılır. NonStatic metodun çağrılması üçün isə class-ın obyekti yaradılmalıdır.

private void button2\_Click(object sender, EventArgs e)

{

StaticAndNonStatic.topla(12, 15);

StaticAndNonStatic stn = new StaticAndNonStatic();

stn.vurma(12, 13);

}

#### Qeyd: Class static olarsa daxilindəki metodlar hər biri static olmalıdır. *(Səbəb – obyektin yaradıla bilməməsidir)*

public static class StaticAndNonStatic

{

public static int topla (int deyer1, int deyer2)

{

int result = deyer1 + deyer2;

return result;

}

}

private void button2\_Click(object sender, EventArgs e)

{

~~StaticAndNonStatic stn = new StaticAndNonStatic();~~

}

## Access Modifiers

#### PUBLİC: Solution daxilində yerləşən bütün proyektlər daxilində properties, method və class-ları görmək üçün istifadə edilən açar söz.

#### PRİVATE: Yanlız cari class daxilində məlumatları görmək üçün istifadə edilən açar söz. *(Default olaraq hər bir element private olur)*

#### İNTERNAL: Cari proyekt daxilində properties, method və class-ları görmək üçün istifadə edilən açar söz.

#### PROTECTED: Cari proyekt daxilində yalnız cari və ondan miras alan class daxilində properties, method-ları görmək üçün istifadə edilən açar söz.

#### PROTECTED İNTERNAL: Həm cari proyekt, həmdə digər proyekt daxilində miras verilmiş class daxilində properties, method-ları görmək üçün istifadə edilən açar söz.

#### UNUTMA: Solution daxilində yerləşən proyektlər arasında məlumat mübadiləsini yerinə yetirmək üçün proyektin Referances hissəsindən digər proyekti əlavə etmək lazımdır.

# DƏRS (Interface)

Interface-lərin məqsədi bir interface yaradaraq digər siniflərin ondan miras almasını təmin etməkdir. Proqramlama sahəsində interface-lər çox vacib yer tutur. İstifadə olunmasına gəldik də isə bunu yanlız zərurət halında deyil, hər zaman istifadə etmək lazımdır.

İnterface-lərin yaradılması zamanı açar söz olaraq “interface” sözündən istifadə edilir və adlandırma prosesində başlanğıc hərfi “I” olaraq təyin edilmişdir.

Nəzərə almaq lazımdır ki, interface-lər ondan miras alan class və ya class-lar olmadıqda heç bir məna kəsb etməyən obyetklərdir. Başqa sözlə desək onların təklikdə heç bir funksiyası yoxdur.

interface IPerson

{

int Id { get; set; }

int FirstName { get; set; }

int LastName { get; set; }

}

class Customer:IPerson

{

public int Id { get; set; }

public int FirstName { get; set; }

public int LastName { get; set; }

public string Address { get; set; }

}

class Student:IPerson

{

public int Id { get; set; }

public int FirstName { get; set; }

public int LastName { get; set; }

public string Department { get; set; }

}

Interface daxilində təyin edilmiş metodlar və ya property -lər mütləq formada ondan misar alan classların daxilində public formatda təyin edilməlidirlər.

namespace Interfaces

{

class Program

{

static void Main(string[] args)

{

PersonManager manager = new PersonManager();

Customer customer = new Customer

{

Id = 1,

FirstName = "Orxan",

LastName = "Memmedeli",

Address = "Baki"

};

manager.Add(customer);

Student student = new Student

{

Id = 2,

FirstName = "Kamil",

LastName = "Memmedelizade",

Department = "Muellim"

};

manager.Add(student);

Console.ReadLine();

}

interface IPerson

{

int Id { get; set; }

string FirstName { get; set; }

string LastName { get; set; }

}

class Customer : IPerson

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

}

class Student : IPerson

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Department { get; set; }

}

class PersonManager

{

public void Add(IPerson person)

{

Console.WriteLine(person.FirstName);

}

}

}

}

Bu nümunədə PersonManager class-ı daxilində metod yaradılır. Lakin normal halda metoda bu parametrləri çap etmək üçün Customer və ya Student class-larının obyekti yaradılmalı idi. Lakin bu o qədər də əlverişli olmadığından bir başa interface-nin obyekti yaradılaraq hər iki sinifə məlumat göndərmək lazım olur.

class PersonManager

{

public void Add(Customer customer // Student student)

{

Console.WriteLine(customer // student.FirstName);

}

}

#### Qeyd: Sonrakı dövrdə hər hansı bir yeni class yaradılan zaman onunda “Add” metodundan istifadə etməsi mümkün olur.

. . .

class Worker : IPerson

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Education { get; set; }

}

static void Main(string[] args)

{

. . .

Worker worker = new Worker

{

Id = 3,

FirstName = "Namiq",

LastName = "Imanzade",

Education = "AZTU"

};

manager.Add(worker);

. . .

}

#### Qeyd: Interface-lərin obyektlərinin yaradılması mümkün deyil. (~~new InterfaceName~~) Lakin bunun əvəzinə : Iperson person = new Worker/Student/Customer nümunədəki kimi ondan miras alan siniflərin obyektləri yaradıla bilər.

## Nümunə

Misal olaraq bizdən baza üzərində “Add”, “Update” və “Delete” əmrlərini icra edən metodların yazılması tələb edilir. Lakin biz bunu “SQL” ilə yazdıqda əgər qarşı tərəf “Oracle” üzərində baza işlədərsə bu metodan istifadə edə bilməz. Bu səbəbdən də İnterface istifadə edərək hər iki server-ə uyğun kodlar yazılır və program hissədə göndərilən məlumata əsasən icra olunur.

İlk olaraq proyektə inrface əlavə **ICustomerDAL** edilir.

namespace Interfaces

{

interface **ICustomerDAL** //Data Access Layer

{

void Add();

void Update();

void Delete();

}

}

Sonra uyğun olaraq SQL və Oracle kodlarını özündə saxlayan class-lar yaradılır.

namespace Interfaces

{

class SQLServerCustomerDAL:**ICustomerDAL**

{

public void Add()

{

Console.WriteLine("SQL added");

}

public void Delete()

{

Console.WriteLine("SQL updated");

}

public void Update()

{

Console.WriteLine("SQL deleted");

}

}

}

namespace Interfaces

{

class OracleCustomerDAL : **ICustomerDAL**

{

public void Add()

{

Console.WriteLine("Oracle added");

}

public void Delete()

{

Console.WriteLine("Oracle updated");

}

public void Update()

{

Console.WriteLine("Oracle deleted");

}

}

}

Sonra metodların istifadə ediləcəyi class yaradılaraq daxilində metodan istifadə edilir.

namespace Interfaces

{

class CustomerManager

{

public void AddCustomer(**ICustomerDAL** customerDAL)

{

customerDAL.Add();

}

}

}

Son mərləhədə tətbiq hissəsində classın obyekti yaradılaraq daxilində təyin edilmiş metoddan istifadə edilir.

static void Main(string[] args)

{

CustomerManager customerManager = new CustomerManager();

customerManager.AddCustomer(new SQLServerCustomerDAL());

customerManager.AddCustomer(new OracleCustomerDAL());

Console.ReadLine();

}

Metodların interface ilə istifadə olunması təkrar istifadə hallarında və ya sonradan dəyişməsi tələb edilən sahələrdə kütləvi dəyişikliklər yerinə daha qısa zamanda məsələləri həll etmək üçün faydalıdır.

#### Qeyd: Eyni zamanda hər iki serverə məlumat eyni zamanda məlumat yükləmək üçün array-lardan istifadə etmək olar.

static void Main(string[] args)

{

**ICustomerDAL**[] customerDALs = new **ICustomerDAL**[2]

{

new SQLServerCustomerDAL(),

new OracleCustomerDAL(),

};

foreach (var customerDAL in customerDALs)

{

customerDAL.Add();

}

Console.ReadLine();

}

#### Qeyd: Əgər yeni bir server-in sistemə daxil edilməsi tələbi yaranarsa, bu zaman kodlarda heç bir deyəşiklik etmədən həmən servərə uyğun class yaradılacaq. İnterface əsasında misal alacaq. Nəticədə proqram daxilində digər servələrin hüquqlarına malik olacaqdır.

## SOLID, Interface Segragation (Inteyfeyslərin doğru planlanması)

Nümunə olaraq bir şirkət daxilində işçilərin həyata keçirdiyi prosesləri nəzərə alan bir interfeys yaratdıla bilər. Bu zaman yaradılan interfeys implament etdiyini siniflərin hər birində uyğun olaraq implament edilmədiri.

Şirkət daxilində 3 növ işçinin (Manager, Worker, Robot) olduğunu nəzərə alaq və işçilərin ümumi cəhətləri olar 3 amili (Work, Eat, GelSalary) interfays daxilində qeyd edək və uyğun classların hər biri buradan implament edilsin. Lakin bu zaman Robot tipli işçilərdə iki amil olmadığından class daxilində sıxıntı yaranacaqdır. (*Bəzi hallarda onlar implament edilir və içləri boş buraxılır. Bu isə estetik açıdan doğru hesab edilmir.*)

Bu problemi aradan qaldırmaq üçün hər prosesə uyğun interfeys yaradaraq onları ardıcıl formada class-lara aşağıdakı kimi əlavə etmək lazımdır.

class Program

{

static void Main(string[] args)

{

IWork[] workers = new IWork[3]

{

new Manager(),

new Worker(),

new Robot()

};

foreach (var item in workers)

{

item.Work(); // eyni zamanda butun isciler isleyir

}

IEat[] eats = new IEat[2]

{

new Manager(),

new Worker()

}; // Robot sinifinin obyektini elave etmek olmur

foreach (var item in eats)

{

item.Eat();

}

}

}

interface IWork

{

void Work();

}

interface IEat

{

void Eat();

}

interface ISalary

{

void GetSalary();

}

class Manager : IWork, IEat, ISalary

{

public void Eat()

{

throw new NotImplementedException();

}

public void GetSalary()

{

throw new NotImplementedException();

}

public void Work()

{

throw new NotImplementedException();

}

}

class Worker : IWork, IEat, ISalary

{

public void Eat()

{

throw new NotImplementedException();

}

public void GetSalary()

{

throw new NotImplementedException();

}

public void Work()

{

throw new NotImplementedException();

}

}

class Robot : IWork

{

public void Work()

{

throw new NotImplementedException();

}

}

}

**T-SQL**

**ORXAN MƏMMƏDƏLİ**

Kurs və əlavə məlumatlar

# SQL TİPLƏRİ

1. Bigint (8 Bayt)
2. Binary(50) (2-Lik Kod)
3. Bit (True; Falce [1;0])
4. Char(10) (Simvollar Toplusu, Sabit Uzunluqlu)
5. Varchar(10) (Dinamik Uzunluq)
6. Date (3 Bayt [Yyyy-MM-Dd])
7. Datetime (8 Bayt [Yyyy-MM-Dd-Hh-Mm-Ss])-1753-2999
8. Datetimeoffset(7) (Ölkəyə Görə Dəyişən Saat)
9. Datetime2(7) (Saniyəni 7 Simvolluq Yazdırır. [Yarışlar Üçün Istifadə Oluna Bilər.])
10. Decimal(18,0) (+-1038, Onluq Ədəd)
11. Float (4 Bayt, Onluq Ədəd)
12. Geography (Lokasiya, GPS)
13. Geometry (En, Uzunlu, Hündürlük)
14. Hierarchyid (Iyerarxiq)
15. Image

#### Qeyd: İmage Tipi Verilən Zaman Bazada Yüklənmələər Baş Verə Bilər. Bu Səbəbdən Də Şəkil Əvəzinə Onun Path-I Vermək Daha Məqbul Hesab Edilir.

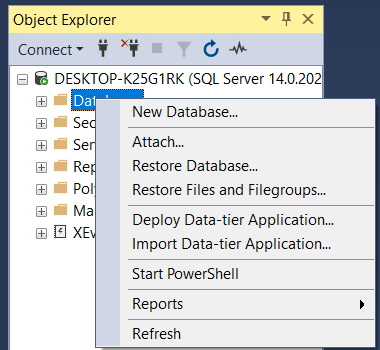
1. Int (2 Mln)
2. Money (8 Bayt - Valyuta)
3. Nchar(10) (2 Qat Uzunluqlu, Müxtəlif Dilləri Dəsdəkləyən Sabit Uzunluqlu Tip)
4. Nvarchar(10) (2 Qat Uzunluqlu, M. D. D. Dəyişən Uzunluqlu Tip)
5. Numeric(18,0) (Tam Ədəd, Uzulunluq-18, Sıfırların Sayı-0)
6. Real (8 Bayt – Kəsli Say)
7. Smallint (+-32000)
8. Sql\_Variant (Müxtəlif Tiplər)
9. Time (Saat)
10. Timestamp (Müəyyən Zaman Aralığı Üçün. – 09:00-15:00 Arası Yoxlanış)
11. Tinyint (0-225)
12. Umqueidentifier (Random 6 Rəqəmli Ədəd.)
13. XML

# DƏRS 20. (SQL – ilkin anlayşlar [SQLServer ilə])

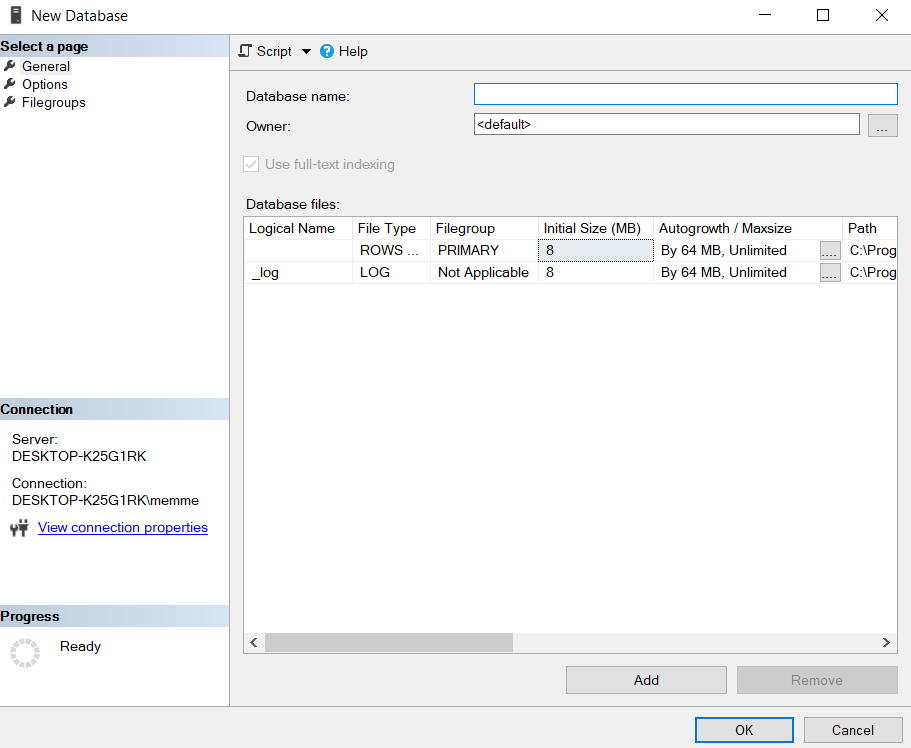
#### UNUTMA: SQL Server-i aktivləşdirən zaman bütün əmrləri icra edə bilmək üçün RUNasADMİNSTRATİON formatında aktivləşdirmək lazımdır.

SQL Server-ə daxil olan zaman açılan pəncərədə computerin local serverinin adı qeyd edilir. OK düyməsini kliklədikdən sonra SQL Server Management Studio (SSMS) açılır. Developerlər üçün burada əsas hissə Databases hissəsidir.

İlkin olaraq bu hissənin üzərinə sağ düyməni klik edərək yeni database əlavə etmək olar.

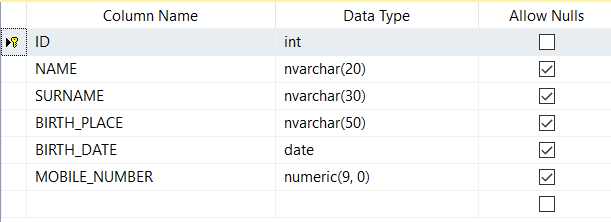


Növbəti mərhələdə açılar pəncərədən Databese üçün seçilir. Adı qeyd edən zaman boşluqların verilməməsi lazımdır. Bunun üçün “\_” və ya hər sözün başlanğıcını böyük hərflərlə yazmaq olar.



Yaradılmış DATABASE daxilində TABLE və digər komponentlər mövcud qovluqlar şəklində ayrı-ayrılıqda saxlanılır. Növbəti mərhələdə TABLE daxil etmək üçün TABLES qovluğunun üzərinə sağ düyməni klik edərək NEW, sonra isə TABLE sözünə klik edərək yeni cədvəl əlavə etmək üçün pəncərə açılır .

#### UNUTMA: Bazaya cədvəl daxil etmək üçün 3 əsas məlumatı *(sütunun adı, tipi, cədvəldə məlumatın boş olub-olmaması)* bilmək lazımdır.



#### UNUTMA: Cədvəlin yaradılması zamanı ən vacib sütun ID sütunudur. ID-nin olması hər sətrin unikal olmasına gətirib çıxarır. Bütün məlumatlar eyni olsa belə ID unikal olur.

Cədvələ ID daxil edən zaman IDENTİTY SPECİFİCATİON hissəsini a ktivləşdirərək avto ID-ni təmin etmək olur. Burada SEED – başlanğıc dəyər və INCREMENT – addımı vermək kifayət edir və eyni zamanda ID sütünunu PRIMERY KEY - əsas açar etmək lazımdır *(digər cədvəllərlə əlaqə yaratmaq üçün)*.

Sütün başlıqlarə və tipləri daxil edindikdən sonra cədvəl yaddaşa verilir. Əlavə edilmiş cədvəl üzərində dəyişikliklər etmək, sütün əlavə etmək, sütun silmək, tipi dəyişmək, allow nulls-u dəyişmək və s..

# DƏRS 21. (DDL, DML)

**DML**

*(DATA MANIPULATION LANGUAGE)*

SELECT

UPDATE

INSERT

DELETE

**DDL**

*(DATA DEFINITION LANGUAGE)*

CREATE (table, proc, trigger)

ALTER (add, drop, alter)

DROP

TRUNCATE

**DCL**

*(DATA CONTROL LANGUAGE)*

GRANT

DENY

REVOKE

**TCL**

*(TRANZACTION CONTROL LANGUAGE)*

#### UNUTMA: SQL kodlarında böyük-kiçik hərf anlayışı yoxdur.

#### Qeyd: Hansı Database ilə işləyən zaman script – code yazmaq üçün yeni query yaratmaq lazımdır. C#- dan fərqli olaraq code icra edildikdən sonra saxlanılması vacib deyil.

## DDL

* SQL vasitesi ilə hər hansı bir şey yaratmaq üçün CRETE,
* Nəyisə dəyişmək üçün ALTER,
* Cədvəlin içini təmizləmək üçün TRUNCATE,
* Cədvəli silmək üçün DROP əmrlərindən istifadə edilir.

create table CUSTOMER(

ID int identity(1,100) primary key not null, 1den baslayaraq-100 addim, esas acar, bos olmayan

CUS\_NAME nvarchar(20),

CUS\_SURNAME nvarchar(30),

BIRTH\_DATE DATE)

* Code seçilir və EXECUTE edilir. Nəticədə CUSTOMER adında cədvəl yaranır.

alter table CUSTOMER add BIRTH\_PLACE nvarchar(50)

* Customer cedvelinə birth\_place sütunu əlavə edir.

alter table CUSTOMER drop column BIRTH\_PLACE

* Customer cedvelində birth\_place sütununu silir.

alter table CUSTOMER alter column CUS\_SURNAME nchar(25)

* Customer cedvelinə cus\_surname sütununun tipini dəyişir.

truncate table customer

* Customer cedvelində içliyini silir. *(Delete əmrindən fərqli olaraq ID sıfırlanır.)*

drop table CUSTOMER

* Customer cedvelini silir.

## DML

* Code vasitəsi ilə məlumat çəkmək üçün SELECT,
* Yeni sətrə məlumat əlavə etmək üçün INSERT,
* Mövcud sətrdə və ya toplu məlumat yeniləmək UPDATE,
* Mövcud sətrdə və ya toplu məlumat silmək üçün UPDATE əmrlərindən istifadə edilir.

insert into PERSONAL

values ('NATIQ','MEDETOV','MASACUSSES','1856-12-02',NULL)

* Personal cedvelinə yeni sətr əlavə edir.

INSERT INTO PERSONAL (NAME,SURNAME,MOBILE\_NUMBER)

VALUES ('AGANATIQ','QAFAROV',NULL)

* Personal cedvelinə seçilmiş xanalara uyğun yeni sətr əlavə edir.

UPDATE PERSONAL SET BIRTH\_PLACE = 'BINƏ',MOBILE\_NUMBER = 502252142

WHERE ID = 11020

* Personal cedvelinə ID-si 11020 olan sətrdə iki xananın dəyərini dəyişir.

DELETE PERSONAL WHERE ID = 11020

* Personal cedvelinə ID-si 11020 olan sətri silir.

## EYNI ANDA TOPLU INSERT

#### Qeyd: Eyni vaxtda insert əmri ilə iki və daha çox sətr əlavə etmək.

INSERT KITABLAR VALUES ('VERITABANI YONETIM SISTEMI 2', 'TURGUT OZSN', 33.50, '2002-10-23')

('VERI YONETIMI', 'ASIL MUTLU', 25.20, '2008-06-23') . . .

## SELECT ILE INSERT

INSERT DBO.KITABLAR\_RENG(KITAB\_AD) SELECT KL.KITAB\_AD FROM DBO.KITABLAR KL

# DƏRS 22. (SELECT, WHERE, LIKE, BETWEEN, IN, AS, COUNT GROUP BY, ORDER BY, DISTINCT, JOINS)

#### Qeyd: Cədvəlin sütun adlarını və içəriyini dəqiq bilmədiyiniz halda cədvəli adlanırmaqla sütun adlarına əlyetənliyi təmin edə bilərik.

SELECT KL.KITAB\_AD, KL.KITAB\_YAZAR FROM DBO.KITABLAR KL

## WHERE

(Şərt ilə məlumat çağırışını təmin edir)

SELECT \* FROM DBO.KITABLAR KL WHERE KL.KITAB\_SEHIFE=200

## IN

(Eyni zamanda bir neçə dəyərə bərabər olanları çağırmağı təmin edir)

SELECT \* FROM DBO.KITABLAR KL WHERE KL.KITAB\_SEHIFE IN (150,200,300,220)

## LIKE

(Söz daxilində, başlanğıcında, sonunda və s. yerlərdə simvol və ya simvollar toplusunun varlığını axtarmaq üçün istifadə edilir)

SELECT KL.KITAB\_AD, KL.KITAB\_YAZAR FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE 'A%'

SELECT KL.KITAB\_AD, KL.KITAB\_YAZAR FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '%A'

SELECT KL.KITAB\_AD, KL.KITAB\_YAZAR FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '%A%'

SELECT KL.KITAB\_AD, KL.KITAB\_YAZAR FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '\_A%'

## ORDER BY

(Sıralamaq imkanı yaradır)

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL ORDER BY KL.KITAB\_QIYMET ASC

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL ORDER BY KL.KITAB\_QIYMET DESC

SELECT \* FROM DBO.KITABLAR KL ORDER BY KL.KITAB\_QIYMET desc, KL.KITAB\_SEHIFE asc

-- əvvəlcə qiymət görə sıralayır, sonra sıralanmışı səhifəyə əsasən sıralayır.

## AND, OR, BETWEEN, >, <, < >

(Şərt daxilində məlumat çağırışını təmin edir.)

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET >20 AND KITAB\_QIYMET <29

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET >20 OR KITAB\_QIYMET <29

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET <>20 AND KITAB\_QIYMET <29

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET >=20 AND KITAB\_QIYMET <=29

SELECT KL.KITAB\_QIYMET FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET BETWEEN 25.20 AND 29

## AS

(Sütunu adlandırmaq üçün istifadə edilir)

SELECT KL.KITAB\_AD AS ADI, KL.KITAB\_YAZAR AS YAZARI FROM DBO.KITABLAR KL

WHERE KL.KITAB\_AD LIKE 'A%'

## COUNT

(Sayını tapmaq üçün istifadə edilir)

SELECT COUNT(\*) AS SAY FROM DBO.KITABLAR KL

SELECT COUNT(\*) AS SAY FROM DBO.KITABLAR KL WHERE KITAB\_QIYMET >20 AND KITAB\_QIYMET <29

## DISTINCT

(Dəyərlərin təkrarlanmalarını nəzərə almadan fərqli dəyərləri gətirir.)

SELECT DISTINCT KL.KITAB\_CAPEVI FROM DBO.KITABLAR KL

## GROUP BY

(Qruplaşdıraraq məlumat çağırmanı təmin edir)

SELECT KL.KITAB\_CAPEVI, COUNT(\*) AS SAYI FROM DBO.KITABLAR KL GROUP BY KL.KITAB\_CAPEVI

ORDER BY SAYI DESC

## ƏLAVLƏR ( '[^A]%', '[A-D]%', '[ADV]%' )

SELECT KL.KITAB\_AD FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '[^A]%'

-- A hərfi ilə başlamayan

SELECT KL.KITAB\_AD FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '[A-D]%'

-- A-dan başlayaraq D-hərfinə kimi olan bütün hərflərlə başlayanlar

SELECT KL.KITAB\_AD FROM DBO.KITABLAR KL WHERE KL.KITAB\_AD LIKE '[ADV]%'

-- A, D, V -hərfləri ilə başlayan

## SQL JOINS

### INNER JOIN

**Yalnız ortaq olan hissələri gətirir.**

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET,KR.KITAB\_RENG

FROM DBO.KITABLAR KL INNER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD=KR.KITAB\_AD

### LEFT JOIN

**Left Join açar sözündən əvvəl yazılan cədvəlin hamısı və ondan sonra yazılan cədvəl ilə ortaq hissəsi.**

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL LEFT OUTER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL LEFT JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

### RIGHT JOIN

**Right Join açar sözündən sonra yazılan cədvəlin hamısı və ondan əvvəl yazılan cədvəl ilə ortaq hissəsi.**

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL RIGHT OUTER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL RIGHT JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

### FULL JOIN

**Birinci cədvəl, ortaq hissələr və ikincidə olub birincidə olmayan məlumatlar**

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL FULL OUTER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL FULL JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

## ƏLAVLƏR

### INNER JOIN EKVİVALENTİ

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET,KR.KITAB\_RENG

FROM DBO.KITABLAR KL, DBO.KITABLAR\_RENG KR

WHERE KL.KITAB\_AD=KR.KITAB\_AD

SELECT SUM(KL.KITAB\_QIYMET) AS 'KITAB CEM', KR.KITAB\_RENG

FROM DBO.KITABLAR KL INNER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

GROUP BY KR.KITAB\_RENG

# DƏRS 23. (PROCEDURE, VİEWS)

## Procedure

#### Qeyd: Prosedur yaradan zaman prosedurun adının əvvəlinə “SP” – stored procedure yazmaq məsləhət bilinir.

#### UNUTMA: Prosedur yazmaq kod yazmaqdan daha etibarlı-təhlükəsiz və sürətlidir.

1. İNSERT PROCEDURE

CREATE PROC SP\_İNSERT\_PERSONAL

@NAME NVARCHAR(20),

@SURNAME NVARCHAR(30),

@BIRTH\_PLACE NVARCHAR(50),

@BIRTH\_DATE DATE,

@MOBILE\_NUMBER NUMERIC(9,0)

AS

BEGIN

INSERT DBO.PERSONAL VALUES (@NAME, @SURNAME, @BIRTH\_PLACE, @BIRTH\_DATE, @MOBILE\_NUMBER)

END

EXEC SP\_İNSERT\_PERSONAL 'ORXAN','MEMMEDELI','MASSALLI','1996-12-21',777556632

2. SELECT PROCEDURE

CREATE PROC SP\_SELECT\_PERSONAL

@ID INT

AS

BEGIN

SELECT \* FROM DBO.PERSONAL WHERE ID = @ID

END

EXEC SP\_SELECT\_PERSONAL 1010

3. UPDATE PROCEDURE

CREATE PROC SP\_UPDATE\_PERSONAL

@ID INT,

@NAME NVARCHAR(20),

@SURNAME NVARCHAR(30),

@BIRTH\_PLACE NVARCHAR(50),

@BIRTH\_DATE DATE,

@MOBILE\_NUMBER NUMERIC(9,0)

AS

BEGIN

UPDATE DBO.PERSONAL

SET NAME = @NAME, SURNAME = @SURNAME, BIRTH\_PLACE = @BIRTH\_PLACE,

BIRTH\_DATE = @BIRTH\_DATE, MOBILE\_NUMBER = @MOBILE\_NUMBER WHERE ID = @ID

END

EXEC SP\_UPDATE\_PERSONAL 11030,'TEYMUR','ALIXANL','MASALLI','2002-02-22',705568632

4. DELETE PROCEDURE

CREATE PROC SP\_DELETE\_PERSONALROW

@ID INT

AS

BEGIN

DELETE DBO.PERSONAL WHERE ID = @ID

END

EXEC SP\_DELETE\_PERSONALROW 1010

ALTER PROC SP\_İNSERT\_PERSONAL

@NAME NVARCHAR(20),

@SURNAME NVARCHAR(30),

@BIRTH\_PLACE NVARCHAR(50),

@BIRTH\_DATE DATE,

@MOBILE NUMERIC(7,0) -- AD VE TIP DEYISIKLIYI

AS

BEGIN

INSERT DBO.PERSONAL VALUES (@NAME, @SURNAME, @BIRTH\_PLACE, @BIRTH\_DATE, @MOBILE)

END

CREATE PROC SP\_INST\_PERSONAL

@NAME NVARCHAR(20),

@SURNAME NVARCHAR(30)

AS

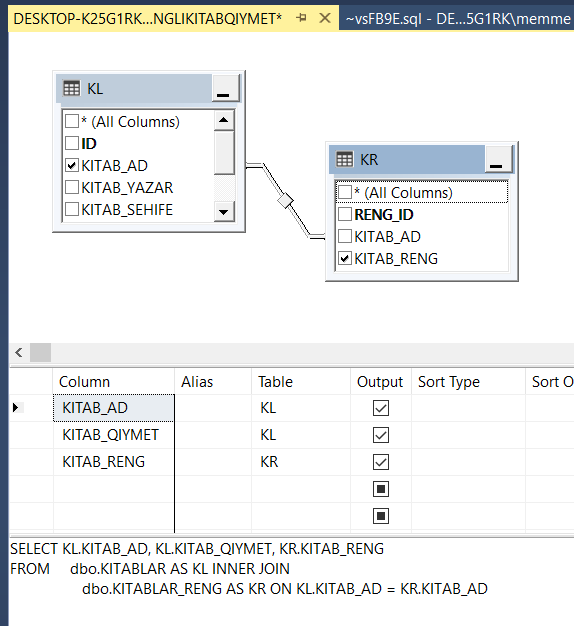
BEGIN

INSERT DBO.PERSONAL(NAME,SURNAME) VALUES (@NAME, @SURNAME)

END

EXEC SP\_INST\_PERSONAL 'OR','KHAN'

## Views

Select ilə çəkilən sorğuların bir table kimi birləşdirilməsinə VIEWS deyilir. *(Nümunə olaraq select ilə bir neçə cədvəldən bəzi məlumatlar çekərək onları virual teble şəklində bir ad altında saxlamaq olar)*

* Ən başlıca üstünlüyü yaddaşda yer tutmamasıdır.
* Digər cədvəllərlə Sinxron-əlaqəlidir. Bu səbəbdən də digərlərində olan dəyişikliklər bir başa bu virtual cədvəldə də əks olunur.
* Kodların daha güvənli formada istifadəsi üçün nəzərdə tutula bilər.
* Uzun və mürəkkəb select sorğusunu bir dəfə əlavə edərək və sonradan sadecə View-nun adını yazaraq çağırmaq olur.

CREATE VIEW V\_RENGLIKITABQIYMET

AS

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL INNER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

SELECT \* FROM DBO.V\_RENGLIKITABQIYMET

### VIEW - ENCRYPTION

(View daxilində yazılmış kodun sonradan görülməməsi üçün istifadə edilir.)

CREATE VIEW V\_RENGLIKITABQIYMET

WITH ENCRYPTION

AS

SELECT KL.KITAB\_AD, KL.KITAB\_QIYMET, KR.KITAB\_RENG

FROM DBO.KITABLAR KL INNER JOIN DBO.KITABLAR\_RENG KR

ON KL.KITAB\_AD = KR.KITAB\_AD

SELECT \* FROM DBO.V\_RENGLIKITABQIYMET

# DƏRS 24-25. (TRIGGER)

Triggerlər DML (insert, delete, update, ~~select~~) və DDl(create, drop, alter) əməliyyatlarına uyğun bir əməliyyatın icrası prosesindən ƏVVƏL və yaxud SONRA hər hansı əməliyatı icra etmək üçün yazılır. Triggerlər iki cür olur:

* FOR - əvvəl
* AFTER – sonra

#### Qeyd: Triggerlər birbaşa olaraq cədvələ tətbiq olunur. *(Cədvəl üçün yazılır)*

#### Qeyd: Triggerlər cədvəl daxilində yerləşən Trigger qovluğunun içərisində yerləşir.

* 1. İlk olaraqtTriggerlər yaradılan zaman create açar sözündən istifadə edilir və Triggerin adı, sonra isə ON – keyword-ü vasitəsi ilə cədvəlin adı yazılır (Nəyin üzərinə yazılacaqsa).
  2. Növbəti addımda triggerin əməliyyatdan əvvəl və ya sonra olunması və tətbiq olunacağı əməliyyat yazılır.
  3. Növbəti addımda AS bağlayıcı isə əməliyyat triggerə bağlanır.
  4. Sonuncu addımda Begin-End daxilində triggerin icra edecəyi əməliyyat yazılır. *(Begin-End yazılmayada bilər. Lakin yazılması daha məqbul hesab edilir.)*

### İNSERT – İCRAÇI USER NAME

CREATE TRIGGER DBO.TRG\_INSERT\_USER ON DBO.PERSONAL

AFTER INSERT – insertdən sonra

AS

BEGIN

DECLARE @ID INT -- deyisen eleva edir (yeni elave olunan setrin id-ni yaxalamaq ucun)

SELECT @ID = ID FROM INSERTED -- virtual cedveldir/ personalin ekvivalentidir

UPDATE DBO.PERSONAL SET INSERT\_USER = 'RAUL' WHERE ID = @ID

END

INSERT DBO.PERSONAL VALUES('ORXAN','MEMMEDELI','OGUZ','1996-11-23',4565685,123,'')

YAXUD INSERT DBO.PERSONAL(NAME, SURNAME) VALUES('ORXAN','MEMMEDELI')

#### DECLARE: Anlıq məlumatı saxlamaq üçün anlıq dəyişən əlavə etmək üçün istifadə olunur.

#### İNSERTED: İnsert, Delete və Update əmrlərinin icrası zamanı onlardan əvvəl virtual cədvəllər yaranır. Yaranan cədvəllər uyğun olaraq İNSERTED, DELETED adlanır. Yaranan hər virtual cədvəl RAM -da saxlanılanaraq mövcud cədvəlin ekvivalenti formatında yaranın və əməliyyatın icrasından öncə məlumatlar ora yazılır və heç bir problem olmazsa əməliyyatın icrası sonrası məlumatlar əsas cədvələ keçir, eyni zamanda virtual cədvəllər silinir. Yeni əlavə olunan sətrin ID-ni yaxalamaq üçün virtual cədvələ yazılan ID DECLARE deyişəninə mənimsədilir və ona uyğun trigger əməliyyatı yerinə yetirilir.

#### UPDATED: Belə bir cədvəl yoxdur. Çünki prosesinə əvvəlcə DELETE sonra INSERT yerinə yetirilir. Nəticədə uyğun olaraq əvvəlcə DELETED, sonra INSERTED cədvəli yaranır. Update prosesində məlumatın orjinalıdı DELETED cədvəlindən, yenisini isə INSERTED cədvəlindən əldə etmək olar.

### DELETE – SİLMƏK QADAĞASI

CREATE TRIGGER DBO.BOTIFICATION ON DBO.PERSONAL

FOR DELETE -- Silməzdən əvvəl

AS

BEGIN

PRINT ('SILMEK QADAGANDIR') -- MessageBox

ROLLBACK TRANSACTION -- emri geri qaytarir

END

DELETE FROM DBO.PERSONAL WHERE ID=1000

#### PRINT: C# -da olduğu kimi SQL daxilində mesaj əks etdimək üçün istifadə edilən açar söz

#### ROLLBACK TRANSACTION: Yazılmış trigger silmək əmrinin yerinə yetirilməsinin qarşısını almaq üçün yazıldığından burada Delete əmrindən imtina etmək və onu geri qaytaramaq üçün istifade edilən açar söz. (*Ona kimi görülən prosesi geriyə alar)*

### Satış Edən Zaman Bir Cədvələ Daxil Edilən Məlumat Əsasən Bazada Dəyişiklik.

Dükanda yerləşən məhsullar PRODUCTS cədvəli daxilində saxlanılır. Satış edilən zaman kassadan gələn məlumatlar isə SALES cədvəlinə qeyd edilir. Məqsəd Sales cədvəlinə hansı məhsuldan nə qədər satıldığını nəzərə alaraq ümumi bazada məhsulun sayından çıxılmasını təmin etməkdir. Bu proses həm məhsul ID-nə, həm də məhsul NAME-ə əsasən yerinə yetirilən iki trigger hazırlanmışdır.

CREATE TRIGGER T\_SATIS ON DBO.SALES

AFTER INSERT

AS

BEGIN

DECLARE @PROD\_ID INT

DECLARE @SALES\_COUNT INT

SELECT @PROD\_ID = PR\_ID, @SALES\_COUNT = SALES\_COUNT FROM inserted

UPDATE DBO.PRODUCTS SET PR\_STOCK = PR\_STOCK - @SALES\_COUNT

WHERE ID = @PROD\_ID

END

INSERT INTO DBO.SALES VALUES (18, 101)

CREATE TRIGGER T\_ADA\_GORE\_SATIS ON DBO.SALES

AFTER INSERT

AS

BEGIN

DECLARE @PROD\_NAME NVARCHAR(50)

DECLARE @SALES\_COUNT INT

SELECT @PROD\_NAME = PR\_NAME, @SALES\_COUNT = SALES\_COUNT FROM inserted

UPDATE DBO.PRODUCTS SET PR\_STOCK = PR\_STOCK- @SALES\_COUNT

WHERE PR\_NAME = @PROD\_NAME

END

INSERT INTO DBO.SALES(SALES\_COUNT, PR\_NAME) VALUES (18, 'ALMA')

## Əlavələr

### Instead of

Trigger yaradılan zaman For və After açar sözlərindən əlavə olaraq INSTEAD OF açar sözündən də istifadə edilə bilər. Sorğuların əngəllənməsi üçün istifadə olunur. Nəticə olaraq Begin-End bloku daxilində yazılmış kod işləyir lakin təyin edilmiş əmr icra olunmur.

CREATE TRIGGER T\_SILMEKOLMAZ ON DBO.KITABLAR

INSTEAD OF DELETE

AS

BEGIN

SELECT ('SILMEK OLMAZ')

END

DELETE FROM DBO.KITABLAR WHERE ID = 1390

### Multiple actions Trigger

Trigger yaradılan zaman eyni anda birdən çox əmrdən istifadə etmək mümkündür

CREATE TRIGGER T\_MULTİTR ON DBO.KITABLAR

AFTER DELETE, UPDATE

AS

BEGIN

PRINT ('MULTİ TRİGGER')

END

UPDATE DBO.KITABLAR SET KITAB\_QIYMET=16.56 WHERE ID = 1390

### Şərt funksiyalı Trigger

Trigger yaradılan zaman şərt bloku ilə tənzimələr həyata keçirmək olar.

CREATE TRIGGER T\_MULTİTR ON DBO.KITABLAR

AFTER DELETE, UPDATE

AS

BEGIN

DECLARE @AD NVARCHAR(50)

SELECT @AD = KITAB\_YAZAR FROM deleted

IF @AD = 'Anar'

BEGIN

PRINT ('ANARIN KITABINI DEYISMEK OLMAZ')

ROLLBACK

END

PRINT ('MULTİ TRİGGER')

END

UPDATE DBO.KITABLAR SET KITAB\_QIYMET=16.56 WHERE ID = 1390

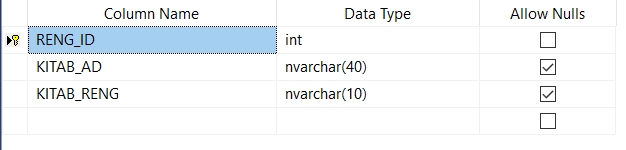
### Enabled-Disabled Trigger

Trigger yaradılan sonra aktiv və passiv edilməsi.

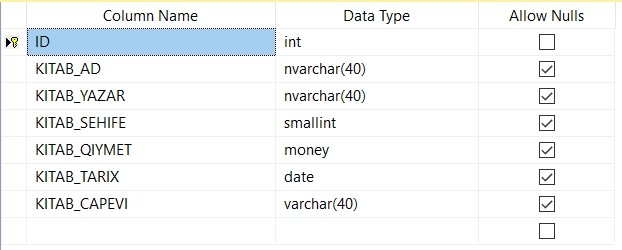
DISABLE TRIGGER T\_SILMEKOLMAZ ON DBO.KITABLAR

ENABLE TRIGGER T\_SILMEKOLMAZ ON DBO.KITABLAR

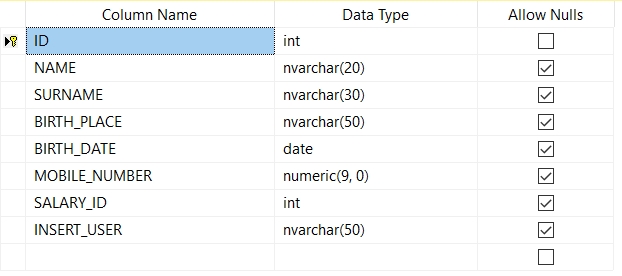
KITABLAR\_RENG



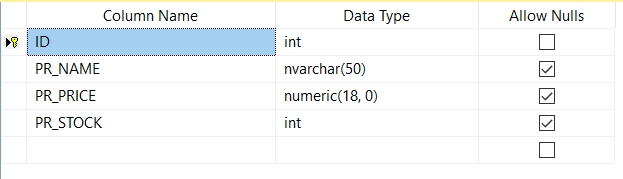
KITABLAR



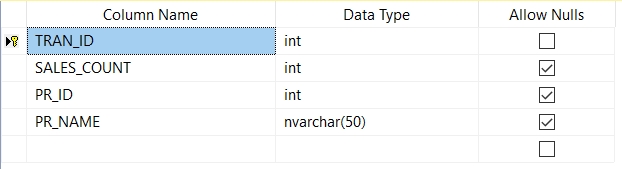
PERSONAL



PRODUCT



SALES

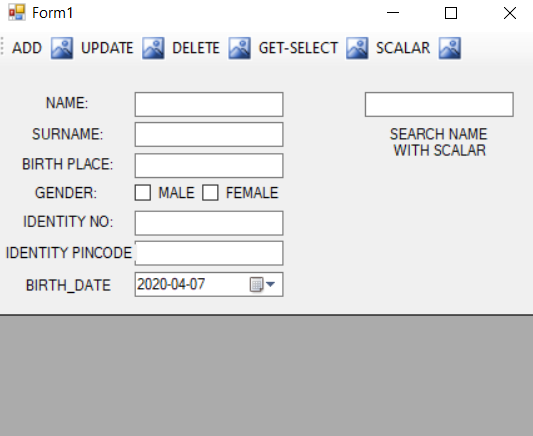


Kurs və əlavə məlumatlar

**ORXAN MƏMMƏDƏLİ**

**ADO.NET**

# DƏRS 26. (ADO.NET) – DataGridView-CellClick { INSERT, UPDATE, DELETE, SELECT, SCALAR }



## ADO.Net – INSERT (Connected)

ADO.Net codlarında istifadə edərək baza ilə əlaqə yaradılır və məlumaqlar SQLSERVER daxilində yaradılmış bazaya daxil edilir. Bunun üçün aşağıdakı mərhələ ardıcıllıqla yerinə yetirilməlidir.

1. İlkin olaraq Connected metodundan istifadə etdiyim üçün hər dəfə baza ilə əlaqə yaratmaq üçün “connection” kodunu yazmamaq üçün bir class yaradılır və məlumat buraya yazılır. DataAccessLayerClass yaradılır.

class DataAccessLayerClass

{

public static string GetConnect()

{

string connection = "Data Source=DESKTOP-K25G1RK;Initial Catalog =

Gr25DB;Integrated Security=True ";

return connection;

}

}

#### Qeyd: Classın yaradılması serverdə hər hansı dəyişiklik edilən zaman bir başa classda dəyişmək üstünlüyü yaradır. Əks halda yorucu və uzun proses ola bilər

#### DİQQƏT!!! Əgər “connection string”-i alan zaman error verərsə, Visual Studio daxilində yerləşən “Data Sourse => add new data source” ardıcıllığını edərək doğru string məlumatı əldə etmək olar.

1. Sonraki mərhələdə form daxilində kodların yazılması prosesi icra olunur.
   1. İlk olaraq SqlConnection-nın obyekti yaradılır və obyekt daxilində DALC class-nın icində yaradılmış GetConnect metodu çağrılır.
   2. Sistemin daxil edilmiş kodları işlətmədən öncə connection-nın işləyib-işləməməsini öyrənmək məqsədi ilə con.Open(); istifadə edilir.
   3. Əgər heç bir problem olmasa SqlCommand yazılır.
   4. Command yazan zaman dəyərlərin bir başa mənimsədilməsi peşəkar üsul hesab edilmədiyindən aşağıdakı kod kambinasiyası daxil edilir.
   5. Növbəti addımda yenidən connection-nın açıq olub-olmaması yoxlanılır. Əgər hər hası bir səbəbdən əlaqə kəsilibsə, yenidən bərpa edilir.
   6. Sonra SqlCommand əmri “excute” edilir.
   7. Ən əsası sistemin işləyib-işləməməsindən asılı olmayaraq connection mütləq CLOSE edilir.

SqlConnection con = new SqlConnection();

con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("INSERT INTO DBO.CUSTOMER VALUES(@CUS\_NAME, @SURNAME,

@BIRTH\_PLACE, @GENDER, @IDENTITY\_NO, @IDENTITY\_PINCODE, @BIRTH\_DATE)", con);

cmd.Parameters.AddWithValue("@CUS\_NAME", textBox1.Text);

cmd.Parameters.AddWithValue("@SURNAME", textBox2.Text);

cmd.Parameters.AddWithValue("@BIRTH\_PLACE", textBox3.Text);

if (checkBox1.Checked == true)

{

cmd.Parameters.AddWithValue("@GENDER", 'K');

}

else if (checkBox2.Checked == true)

{

cmd.Parameters.AddWithValue("@GENDER", 'Q');

}

else

{

cmd.Parameters.AddWithValue("@GENDER", null);

}

cmd.Parameters.AddWithValue("@IDENTITY\_NO", textBox4.Text);

cmd.Parameters.AddWithValue("@IDENTITY\_PINCODE", textBox5.Text);

cmd.Parameters.AddWithValue("@BIRTH\_DATE", dateTimePicker1.Text);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("İNSERTED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

Finally

{

con.Close();

}

### ADO.Net – INSERT (Daha düzgün forması)

İnsert prosesini yuxarıdakı formada icra emək mümkündür. Lakin doğru forması user-in girdiyi məlumatların bir başa bazaya getməsi doğru bir hall deyil. Bu səbəbdən OOP qaydalarına uyğun olaraq SQL SERVER daxilindəki cədvələ uyğun “Customer” class-ı yaradılır və lazım olan ENCAPSİLATİON tətbiq edərək user səhvlərindən qaçınmaq olar.

class Customer

{

private int id;

private string cus\_name;

private string surname;

private string birth\_place;

private char gender;

private int identity\_no;

private int identity\_pincode;

private string birth\_date;

public int Id { get => id; set => id = value; }

public string Cus\_name { get => cus\_name; set => cus\_name = value.ToUpper(); }

public string Surname { get => surname; set => surname = value.ToUpper(); }

public string Birth\_place { get => birth\_place; set => birth\_place = value.ToUpper(); }

public char Gender { get => gender; set => gender = value; }

public int Identity\_no { get => identity\_no; set => identity\_no = value; }

public int Identity\_pincode { get => identity\_pincode; set => identity\_pincode = value; }

public string Birth\_date { get => birth\_date; set => birth\_date = value; }

}

Sonra uyğun olaraq Form daxilində Class-ın obyekti yaradılır və dəyərlər class-ın properties-lərinə mənimsədilir. Və SqlCommand əmrinin parametrlərinin əlavəsi zamanı form atributları əvəzinə class daxilindən mənimsədilmə həyata keçirilir.

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

Customer cs = new Customer();

cs.Cus\_name = textBox1.Text;

cs.Surname = textBox2.Text;

cs.Birth\_place = textBox3.Text;

cs.Identity\_no = Convert.ToInt32(textBox4.Text);

cs.Identity\_pincode = Convert.ToInt32(textBox5.Text);

cs.Birth\_date = dateTimePicker1.Text;

if (checkBox1.Checked == true)

{

cs.Gender = 'K';

}

else if (checkBox2.Checked == true)

{

cs.Gender = 'Q';

}

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("INSERT INTO DBO.CUSTOMER VALUES(@CUS\_NAME, @SURNAME,

@BIRTH\_PLACE, @GENDER, @IDENTITY\_NO, @IDENTITY\_PINCODE, @BIRTH\_DATE)", con);

cmd.Parameters.AddWithValue("@CUS\_NAME", cs.Cus\_name);

cmd.Parameters.AddWithValue("@SURNAME", cs.Surname);

cmd.Parameters.AddWithValue("@BIRTH\_PLACE", cs.Birth\_place);

if (checkBox1.Checked == true)

{

cmd.Parameters.AddWithValue("@GENDER", cs.Gender);

}

else if (checkBox2.Checked == true)

{

cmd.Parameters.AddWithValue("@GENDER", cs.Gender);

}

else

{

cmd.Parameters.AddWithValue("@GENDER", null);

}

cmd.Parameters.AddWithValue("@IDENTITY\_NO", cs.Identity\_no);

cmd.Parameters.AddWithValue("@IDENTITY\_PINCODE", cs.Identity\_pincode);

cmd.Parameters.AddWithValue("@BIRTH\_DATE", cs.Birth\_date);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("İNSERTED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

Finally

{

con.Close();

}

#### DİQQƏT!!! EXECUTE prosesində üç növ command-dan istifadə edilir.

1. ExecuteNonQuery() – {İnsert, update, delete} geriyə dəyər qaytarmır.
2. ExecuteReader() – {Select} geriyə dəyər qaytarır – Məlumatı oxuyur
3. ExecuteScalar() – {Select} geriyə dəyər qaytarır – Varlığını yoxluyur

## ADO.Net – SELECT (Connected)

Select sorğusunun icrası zamanı insertdə olduğu kimi:

**1.** Connection yaradılır, **2.** Əlaqə yoxlanılır, **3.** Command yazılır, **4.** Əlaqə yenidən yoxlanılır,

**5.** Əsas mərhələ yazılmış əmrin icrasına aid edilir. Burada məlumat bazadan application-na çəkildiyinə görə ExecuteReader əmri icra olunur. ExecuteReader əmri geriyə SqlDataReader tipində dəyər qaytarır. Lakin geriyə qayıdan bu dəyəri bi başa DataGridView-ya göndərmək olmur. Bu səbəbdən də sqldəki cədvəlin virtualını yaratmaq üçün DataTable-dan istifadə olunur.

**6.** Həmin DataTable-ın içinə SqlDataReader tipindəki dəyər mənimsədilir.

**7.** Sonra isə DataTable-ın içinə mənimsədilmiş dəyər DataGridView-ya göndərilir.

**8.** Son olaraq connection bağlanılır.

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("SELECT \* FROM DBO.CUSTOMER", con);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

SqlDataReader sdr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(sdr);

dataGridView1.DataSource = dt;

con.Close();

## DataGridView – CellClick

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

DataGridViewRow row = dataGridView1.CurrentRow;

textBox1.Tag = row.Cells["ID"].Value.ToString();

textBox1.Text = row.Cells["CUS\_NAME"].Value.ToString();

textBox2.Text = row.Cells["SURNAME"].Value.ToString();

textBox3.Text = row.Cells["BIRTH\_PLACE"].Value.ToString();

textBox4.Text = row.Cells["IDENTITY\_NO"].Value.ToString();

textBox5.Text = row.Cells["IDENTITY\_PINCODE"].Value.ToString();

dateTimePicker1.Text = row.Cells["BIRTH\_DATE"].Value.ToString();

if (Convert.ToChar(row.Cells["GENDER"].Value) == 'K')

{

checkBox1.Checked = true;

}

else if (Convert.ToChar(row.Cells["GENDER"].Value) == 'Q')

{

checkBox2.Checked = true;

}

}

## ADO.Net – UPDATE (Connected)

ExecuteScalar geriyə obyek tipində bir dəyər qaytarır. ExecuteScalar ilə çəkilən sorğunun ən birini məlumat nə olarsa onu özünə mənimsədir *(ID və s.)* və ya aşağıdakı nümunəyə uyğun olaraq sayın gətirilməsi kimidə yazmaq olar.

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

Customer cs = new Customer();

cs.Id =Convert.ToInt32(textBox1.Tag);

cs.Cus\_name = textBox1.Text;

cs.Surname = textBox2.Text;

cs.Birth\_place = textBox3.Text;

cs.Identity\_no = Convert.ToInt32(textBox4.Text);

cs.Identity\_pincode = Convert.ToInt32(textBox5.Text);

cs.Birth\_date = dateTimePicker1.Text;

if (checkBox1.Checked == true)

{

cs.Gender = 'K';

}

else if (checkBox2.Checked == true)

{

cs.Gender = 'Q';

}

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("UPDATE DBO.CUSTOMER SET CUS\_NAME=@N, SURNAME=@S,

BIRTH\_PLACE=@B\_P, GENDER=@G, IDENTITY\_NO=@I\_N, IDENTITY\_PINCODE=@I\_P,

BIRTH\_DATE=@B\_D WHERE ID=@ID", con);

cmd.Parameters.AddWithValue("@ID", cs.Id);

cmd.Parameters.AddWithValue("@N", cs.Cus\_name);

cmd.Parameters.AddWithValue("@S", cs.Surname);

cmd.Parameters.AddWithValue("@B\_P", cs.Birth\_place);

if (checkBox1.Checked == true)

{

cmd.Parameters.AddWithValue("@G", cs.Gender);

}

else if (checkBox2.Checked == true)

{

cmd.Parameters.AddWithValue("@G", cs.Gender);

}

else

{

cmd.Parameters.AddWithValue("@G", null);

}

cmd.Parameters.AddWithValue("@I\_N", cs.Identity\_no);

cmd.Parameters.AddWithValue("@I\_P", cs.Identity\_pincode);

cmd.Parameters.AddWithValue("@B\_D", cs.Birth\_date);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("UPDATED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

## ADO.Net – DELETE (Connected)

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

Customer cs = new Customer();

cs.Id = Convert.ToInt32(textBox1.Tag);

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("DELETE FROM DBO.CUSTOMER WHERE ID=@ID", con);

cmd.Parameters.AddWithValue("@ID", cs.Id);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("DELETED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

## ADO.Net – SCALAR (Connected)

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("SELECT COUNT(\*) FROM CUSTOMER WHERE CUS\_NAME = @N", con);

cmd.Parameters.AddWithValue("@N", textBox6.Text);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

string say = cmd.ExecuteScalar().ToString();

MessageBox.Show(say);

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

# DƏRS 27. (ADO.NET DISCONNECTED-ADAPTER)

Disconnected üsulunda digərində fərqli olaraq SqlCommand əvəzinə SqlDataAdapter istifadə olunur. Disconnecded üsulu etibarlı hesab olunmur və istifadə halları yox səviyyəsinə bərəbərdir. Ən çox istifadə edilə SELECT sorğusudur. *(daha sürətli ilşləməsinə görə).* Bu üsulla Azərbaycan əlifbasına uyğun məlumatlar bazaya sual işarəsi kimi daxil edilməs müşahidə edildi.

## ADO.Net – INSERT (DisConnected)

İlk olaraq Connection əmrini Qlobal sahəyə yazaraq hər dəfə yazmaqdan qurtulmuş oluruq. Digər üsuldan fərqli olaraq SqlDataAdapter yaradılır. Sonra məlumatlar SqlDataAdapter-dan DataTable virtual cədvələ yazılır. Bütün hallarda məlumat ilk olaraq adapter-ə, oradan isə virtual cədvələ yüklənir.

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

private void toolStripButton1\_Click(object sender, EventArgs e)

{

Customer cs = new Customer();

cs.Id = Convert.ToInt32(textBox1.Tag);

cs.Cus\_name = textBox1.Text;

cs.Surname = textBox2.Text;

cs.Birth\_place = textBox3.Text;

cs.Identity\_no = Convert.ToInt32(textBox4.Text);

cs.Identity\_pincode = Convert.ToInt32(textBox5.Text);

cs.Birth\_date = dateTimePicker1.Text;

if (checkBox1.Checked == true)

{

cs.Gender = 'K';

}

else if (checkBox2.Checked == true)

{

cs.Gender = 'Q';

}

SqlDataAdapter adapter = new SqlDataAdapter("INSERT INTO DBO.CUSTOMER VALUES('" +

cs.Cus\_name + "','" + cs.Surname + "','" + cs.Birth\_place + "','" + cs.Gender +

"','" + cs.Identity\_no + "','" + cs.Identity\_pincode + "','" + cs.Birth\_date +

"')", con);

DataTable dt = new DataTable();

adapter.Fill(dt);

MessageBox.Show("INSERTED");

}

## ADO.Net – SELECT-GET (DisConnected)

private void toolStripButton4\_Click(object sender, EventArgs e)

{

SqlDataAdapter adpt = new SqlDataAdapter("SELECT \* FROM CUSTOMER", con);

DataTable dt = new DataTable();

adpt.Fill(dt);

dataGridView1.DataSource = dt;

}

## ADO.Net – UPDATE (DisConnected)

Customer cs = new Customer();

cs.Id = Convert.ToInt32(textBox1.Tag);

cs.Cus\_name = textBox1.Text;

cs.Surname = textBox2.Text;

cs.Birth\_place = textBox3.Text;

cs.Identity\_no = Convert.ToInt32(textBox4.Text);

cs.Identity\_pincode = Convert.ToInt32(textBox5.Text);

cs.Birth\_date = dateTimePicker1.Text;

if (checkBox1.Checked == true)

{

cs.Gender = 'K';

}

else if (checkBox2.Checked == true)

{

cs.Gender = 'Q';

}

SqlDataAdapter adapter = new SqlDataAdapter("UPDATE DBO.CUSTOMER SET CUS\_NAME =

'" + cs.Cus\_name + "', SURNAME = '" + cs.Surname + "', BIRTH\_PLACE = '" +

cs.Birth\_place + "', GENDER = '" + cs.Gender + "', IDENTITY\_NO = " +

cs.Identity\_no + ", IDENTITY\_PINCODE = " + cs.Identity\_pincode + ", BIRTH\_DATE

= '" + cs.Birth\_date + "' WHERE ID = " + cs.Id + "", con);

DataTable dt = new DataTable();

adapter.Fill(dt);

MessageBox.Show("UPDATED");

## ADO.Net – DELETE (DisConnected)

Customer cs = new Customer();

cs.Id = Convert.ToInt32(textBox1.Tag);

SqlDataAdapter adpt = new SqlDataAdapter("DELETE FROM CUSTOMER WHERE ID = '" +

cs.Id + "'", con);

DataTable dt = new DataTable();

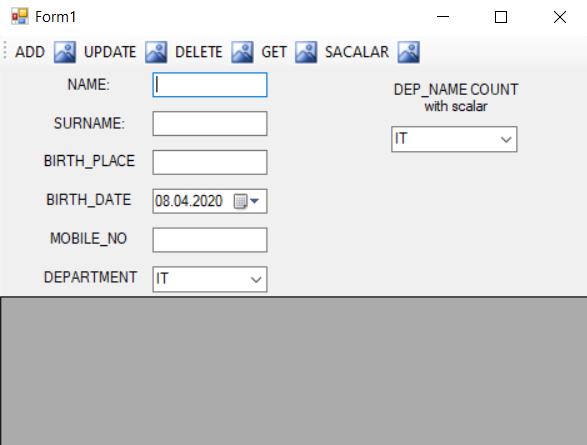
adpt.Fill(dt);

dataGridView1.DataSource = dt;

MessageBox.Show("DELETED");

# İki cədvəldə join daxil edərək məlumat mübadiləsi ilə INSERT, SELECT, DELETE, UPDATE və SCALARA

Connected üsulundan istifadə edərək, iki cədvəlin birləşmiş məlumatların üzərində əmrlərin icrasını yerinə yetirmək. DEPARTMENT və PERSONAL cədvəlləri arasında əlaqə yaradılır. Yaradılan əlaqə ondan ibarətdir ki, məlumatlar PERSONAL cədvəlinə daxil edilən zaman hər bir şəxsin DEPARTAMENTİ əvəzinə onun ID kodu getsin. Məqsəd kənar yerdər PERSONAL cədvəli çağrılan zaman yanlız ID əsaslı məlumat görməsini təmin etməkdir.



Application-nun görünüşü digər proyektlərə bənzər formada yaradılır.

Digər proyektlərdən fərqli olaraq burada iki DEPARTAMENT cədvəli daxilindəki Departament adlarını görmək üçün həmin cədvəldən bu məlumatları ComboBox-un sorsuna daxil etmək lazımdır. Bunun üçün ilk növbətə DataTable tipində GetDepartament() metodu və ona uyğun form-un LOAD hissəsinə müvafiq kodlar daxil edilir.

public DataTable GetDepartament()

{

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM DEPARTAMENT", con);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

SqlDataReader dr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(dr);

con.Close();

return dt;

}

private void Form1\_Load(object sender, EventArgs e)

{

// məlumatların daxil edilməsi üçün yaradılan comboBox1

DataTable dir = GetDepartament();

comboBox1.DataSource = dir;

comboBox1.DisplayMember = "DEP\_NAME";

comboBox1.ValueMember = "ID";

// SCALAR əmrinin icra edilməsi üçün yaradılan comboBox2

DataTable dir2 = GetDepartament();

comboBox2.DataSource = dir2;

comboBox2.DisplayMember = "DEP\_NAME";

comboBox2.ValueMember = "ID";

}

Metod daxilində SELECT sorğusu ilə DEPARTAMENT cədvəlindən məlumat çəkilir. Çəkilən məlumat virtual DataTable cədvəlinə mənimsədilir. Mənimsədilmiş məlumatlar comboBox.DataSource hissəsinə əlavə edilir. Bundan sonra ekradan görülən məlumata (DisplayMember) "DEP\_NAME" sütunu, bazaya daxil ediləcək məlumat hissəsinə (ValueMember) isə "ID" sütunu seçilir.

#### Qeyd: Form daxilində iki dəfə metodun cağrılma səbəbi ComboBox-ların bir-birindən əlaqəsiz işləməsini təmin etməkdir.

## ADO.Net – INSERT (JOİN)

con.Open();

SqlCommand cmd = new SqlCommand("INSERT PERSONAL2 VALUES

(@AD,@SOYAD,@D\_Y,@D\_T,@M\_N,@DEP)", con);

cmd.Parameters.AddWithValue("@AD", textBox1.Text);

cmd.Parameters.AddWithValue("@SOYAD", textBox2.Text);

cmd.Parameters.AddWithValue("@D\_Y", textBox3.Text);

cmd.Parameters.AddWithValue("@D\_T", dateTimePicker1.Value.ToString

("yyyy-MM-dd")); // FERQLI YANASMA – custom etmədən kod ilə dəyişmə

cmd.Parameters.AddWithValue("@M\_N", textBox4.Text);

cmd.Parameters.AddWithValue("@DEP", comboBox1.SelectedValue);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("INSERED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

## ADO.Net – SELECT (JOİN)

con.Open();

SqlCommand cmd = new SqlCommand("SELECT P2.ID, P2.NAME,P2.SURNAME,

P2.BIRTH\_PLACE, P2.BIRTH\_DATE, P2.MOBILE\_NUMBER, DP.DEP\_NAME FROM DBO.PERSONAL2

P2 INNER JOIN DEPARTAMENT DP ON P2.DEP\_ID = DP.ID", con);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

SqlDataReader dr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(dr);

dataGridView1.DataSource = dt;

con.Close();

## ADO.Net – UPDATE (JOİN)

con.Open();

SqlCommand cmd = new SqlCommand("UPDATE PERSONAL2 set NAME = @AD, SURNAME =

@SOYAD, BIRTH\_PLACE = @D\_Y, BIRTH\_DATE = @D\_T, MOBILE\_NUMBER = @M\_N, DEP\_ID =

@DEP WHERE ID = @ID", con);

cmd.Parameters.AddWithValue("@ID", textBox1.Tag);

cmd.Parameters.AddWithValue("@AD", textBox1.Text);

cmd.Parameters.AddWithValue("@SOYAD", textBox2.Text);

cmd.Parameters.AddWithValue("@D\_Y", textBox3.Text);

cmd.Parameters.AddWithValue("@D\_T", dateTimePicker1.Value.ToString

("yyyy-MM-dd")); // FERQLI YANASMA

cmd.Parameters.AddWithValue("@M\_N", textBox4.Text);

cmd.Parameters.AddWithValue("@DEP", comboBox1.SelectedValue);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("UPDATE");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

## ADO.Net – DELETE (JOİN)

con.Open();

SqlCommand cmd = new SqlCommand("DELETE FROM PERSONAL2 WHERE ID = @ID", con);

cmd.Parameters.AddWithValue("@ID", textBox1.Tag);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("DELETED");

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

finally

{

con.Close();

}

## ADO.Net – SCALAR (JOİN)

con.Open();

SqlCommand cmd = new SqlCommand("SELECT count(\*) FROM PERSONAL2 WHERE DEP\_ID =

@ID", con);

cmd.Parameters.AddWithValue("@ID", comboBox2.SelectedValue);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

string say = cmd.ExecuteScalar().ToString();

MessageBox.Show(say);

con.Close();

## DataGridView-CellClick

DataGridViewRow row = dataGridView1.CurrentRow;

textBox1.Tag = row.Cells["ID"].Value.ToString();

textBox1.Text = row.Cells["NAME"].Value.ToString();

textBox2.Text = row.Cells["SURNAME"].Value.ToString();

textBox3.Text = row.Cells["BIRTH\_PLACE"].Value.ToString();

dateTimePicker1.Text = row.Cells["BIRTH\_DATE"].Value.ToString();

textBox4.Text = row.Cells["MOBILE\_NUMBER"].Value.ToString();

comboBox1.Text = row.Cells["DEP\_NAME"].Value.ToString();

# DƏRS 28. (ADO.NET -yekun) Buttons Command

## Axdarılan dəyərə uyğun olaraq çəkilmiş məlumatları textbox-lara yazmaq.

(Üsula uyğun olaraq iki variant verilmişdir.)

private void button1\_Click(object sender, EventArgs e)

{

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("SELECT \* FROM DBO.CUSTOMER WHERE IDENTITY\_PINCODE = '" +

textBox5.Text + "'", con);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

// DAHA OPTIMAL HESAB OLUNUR

textBox1.Text = dr["CUS\_NAME"].ToString();

textBox2.Text = dr["SURNAME"].ToString();

textBox3.Text = dr["BIRTH\_PLACE"].ToString();

textBox4.Text = dr["IDENTITY\_NO"].ToString();

dateTimePicker1.Text = dr["BIRTH\_DATE"].ToString();

if (dr["GENDER"].ToString() == "K")

{

checkBox1.Checked = true;

checkBox2.Checked = false;

}

else if (dr["GENDER"].ToString() == "Q")

{

checkBox2.Checked = true;

checkBox1.Checked = false;

}

//textBox1.Text = dr.GetString(1);

//textBox2.Text = dr.GetString(2);

//textBox3.Text = dr.GetString(3);

//textBox4.Text = dr.GetInt32(5).ToString();

//dateTimePicker1.Text = dr.GetDateTime(7).ToString();

//if (dr.GetString(4) == "K")

//{

// checkBox1.Checked = true;

// checkBox2.Checked = false;

//}

//else if (dr.GetString(4) == "Q")

//{

// checkBox2.Checked = true;

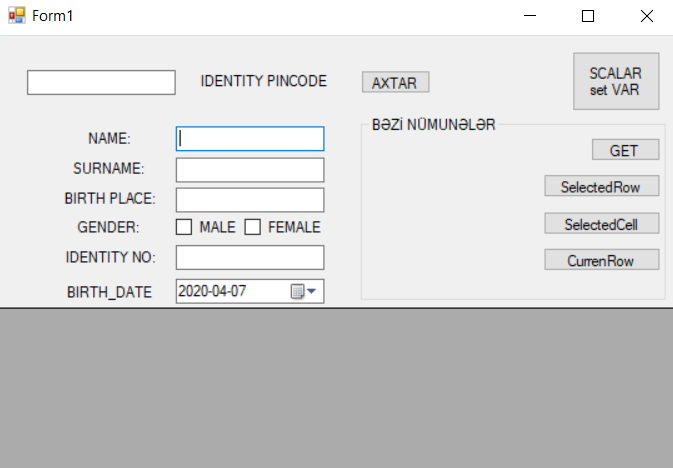
// checkBox1.Checked = false;

//}

}

con.Close();

}



## Selected Row

private void button3\_Click(object sender, EventArgs e)

{

// secilmis butov setrin id nomresini getirir

DataGridViewRow SelectedRov = dataGridView1.SelectedRows[0];

int id = (int)SelectedRov.Cells["ID"].Value;

MessageBox.Show(id.ToString());

}

## Selected Cell

private void button4\_Click(object sender, EventArgs e)

{

DataGridViewCell SelectCell = dataGridView1.SelectedCells[0];

string deyer = SelectCell.Value.ToString();

MessageBox.Show(deyer);

// index = 0

// secilmis xana ve ya secilmis xanalarin en sonucusunun deyerini ekrana cixarir

}

## Current Row

private void button5\_Click(object sender, EventArgs e)

{

// secilmis xanaya uygun olan setrin id nomresini ve diger komponentleri getirir

DataGridViewRow CurrentRov = dataGridView1.CurrentRow;

int id = (int)CurrentRov.Cells["ID"].Value;

string soyad = CurrentRov.Cells["surname"].Value.ToString();

MessageBox.Show(id.ToString());

MessageBox.Show(soyad);

}

## ExecuteScalar deyerinin var tipine menimsetmek

private void button6\_Click(object sender, EventArgs e)

{

SqlConnection con = new SqlConnection(DataAccessLayerClass.GetConnect());

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("SELECT Count (\*) FROM DBO.CUSTOMER WHERE ID = 100" , con);

var result = cmd.ExecuteScalar();

// ExecuteScalar Obyekt tipinde deyer qaytardigina gore onu VAR tipine menimsede bilerik

MessageBox.Show(result.ToString());

}

Kurs və əlavə məlumatlar

**ORXAN MƏMMƏDƏLİ**

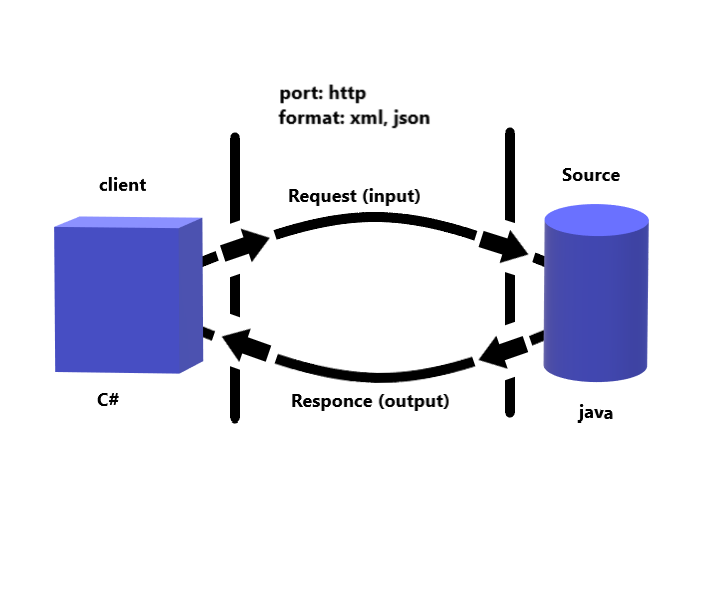
**WEB  
SERVICES**

# DƏRS 29. (Web Service – anlayışı, ASMX)

Web Service(WS)-lər localda işləyən client proqramlara servis verməklə bəzi metodların servis üzərindən icrasını həyata keçirməyə imkan verən texnologiyadır. Lakin WS-lərə Web aplication demək olmaz. Çünki WS-lər localda işləyən proqramların sadecə bəzi məlumatlarını veb üzərindən göndərərək idarə edilməsini təmin edir. *(Daha ciddi və təhlükəsizliyi təmin edilməsi hissələrində WS istifadə olunur.)* Bir başa Web layihələr isə bütün məlumatların veb üzərində saxlanılması təmin edilir.

Misal olaraq, Milliön ödəmə portalı yazılmış bir client proqramdır. Müxtəlif yerlərdən servislər alaraq ödəmələrin reallaşdırılmasını təmin edir. Misal olaraq mobil operatorlar. Amma nəzərə almaq lazımdır ki, bu məlumatlar veb üzərində açıq şəkildə olsa idi hər kəs rahatlıqla məlumat göndərək balans artıra bilərdi. Bu səbəbdən də məlumatlar WS-lər asasında alınması və göndərilməsi təmin edilir.

**NƏTİCƏ:** Local applicationlar arası məlumat mübadiləsinin icrası üçün WS-lərdən istifadə olunur.



WS-lər iki hissədən CLİENT *(servisi istifadə edən)* və SOURCE *(servisi təmin edən)* hissələrdən ibarətdir. Məlumatların ötürülməsi şəbəkə üzərindən *(port: http)* ötürülür. Lakin lokal məlumatların ötürülməsi zamanı yüksək təhlükəsizlik tədbirləri və şifrləmə metodlarından istifadə edilməlidir ki, məlumatlar kiber cinayətkarlar tərəfindən ələ keçirilməsin.

Əlavə olaraq nəzərə almaq lazımdır ki, Service və Client müxtəlif proqramlama dilləri ilə yazıla bilər. Bu proyektlər arasında əlaqə qurmaq üçün eyni dil olmasını təmin etmək absurt məsələyə çevrilir. Bu səbəbdən dil promlemi format dilləri vasitəsi ilə həll olunur *(XML)*. Yəni məlumat şəbəkəyə çıxan zaman XML formatında göndərilir.

Məlumat mübadilər iki prosesdə icra edilir. Birinci Client tərəf Source tərəfə REQUEST – yəni İNPUT dəyər göndərilir. Göndərilmiş dəyər Serverdə yoxlanılır və ona uyğun olaraq təsdiq və ya imtina tipli RESPONDE – yəni OUTPUT məlumat göndərilir.

Məlumat qəbulu zamanı məlumatlarən təsdiqlənməsi müxtəlif metodlar ilə həyata keçirilir. Nümunə olaraq mobil operatorlara ödəniş edilən zaman ilkin olaraq nömrə göndərilir və göndərilən nömrə yoxlanılması ilkin metod tərəfindən icra edilir. Əgər məlumat doğrudursa ödənişin daxil edilməsi tələb olunur və ona uyğun əməliyyat icre edən metot son olaraq sorğuya görə nəticəni geri qaytarır.

#### Qeyd: WS-lərdə ilk olaraq SOURCE tərəf yazılmalıdır. Soruce tərəf yazıldıqdan sonra dərhal Clinet yazılmaq doğru deyil, ilk olaraq Source tərəf müxtəlif vasitələr ilə yoxlanılmalıdır *(Google Chrome-WSDL, Soap UI və s.)*. Əgər Source tərəfin məlumatları problemsiz olaraq işləyirsə, ondan sonra client hissəsi qurulmalıdır.

## ASMX – toplama, vurma

WS proyektinin əlavə edilməsi üçün ilkin olaraq EMPTY-MVC-WEB APPLICATION yaradılır. Yaradılan web application-nın içinə NEW ITEM hissəsindən ASMX-WS əlavə edilir və daxilində metodlar yazılır.

public class WebService\_Operations : System.Web.Services.WebService

{

[WebMethod]

public string HelloWorld()

{

return "Hello World";

}

[WebMethod]

public int Topla(int deyer1, int deyer2)

{

int netice = deyer1 + deyer2;

return netice;

}

[WebMethod]

public int Vur(int deyer1, int deyer2)

{

int netice = deyer2 \* deyer1;

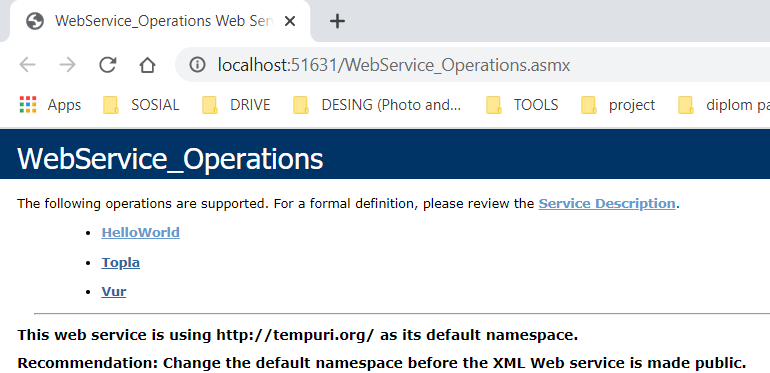
return netice;

}

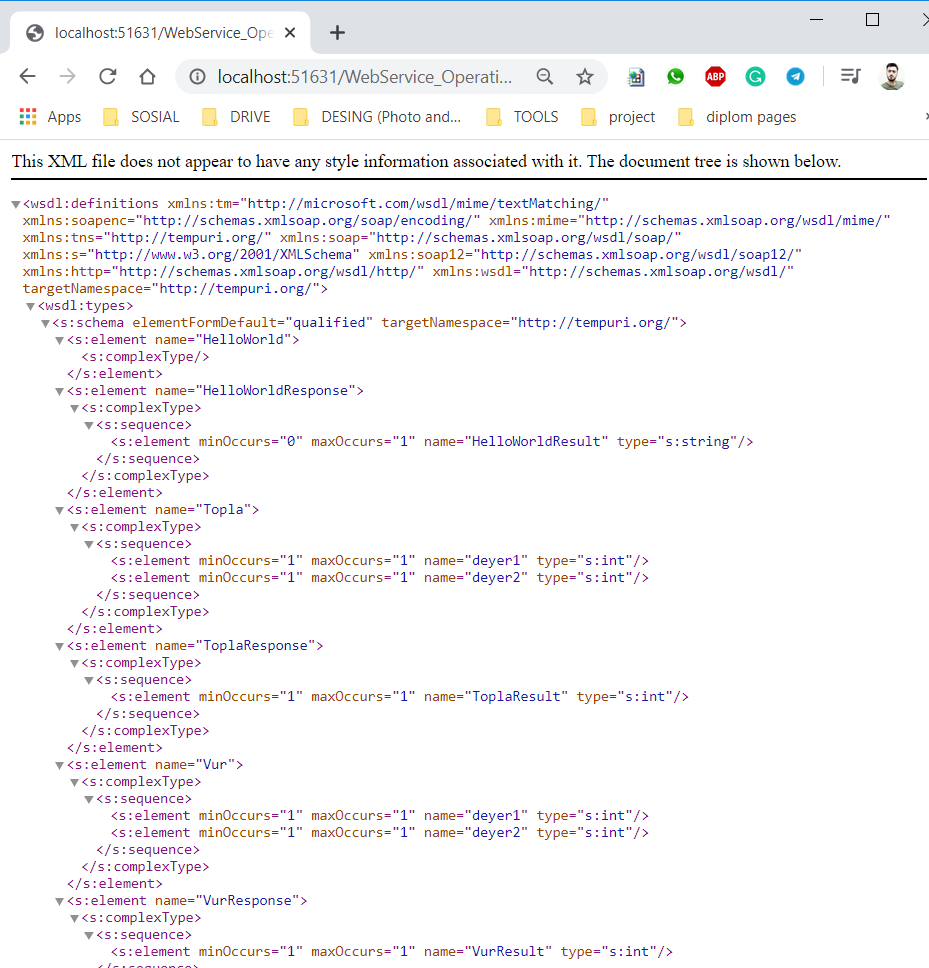
}

#### Qeyd: Yaradılan metodun client tərəfindən görülməsini istəyiriksə, metodun əvvəlinə [WebMethod] ifadəsi yazılmalıdır.

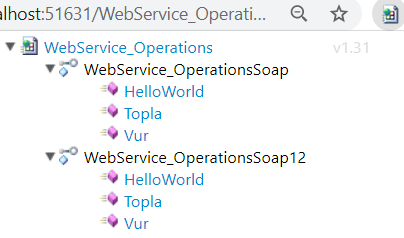
Proyekti işə saldığımız zaman browser-in ekranında metodları yoxlamaq üçün müvafiq sadə açılır. Açılmış bu sahədən metodları yoxlamaq mümkündür.



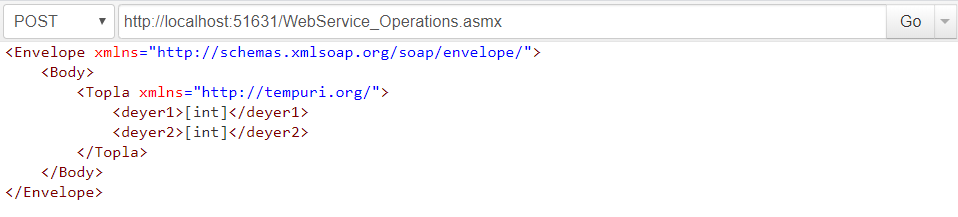
Lakin burada yoxlanılma local olduğundan problem verməyə bilər. Bu səbəbdən də onu şəbəkə üzərindən yoxlamaq üçün lazım olan tətbiqlərdən istifadə edilir. *(Yazılmış metodlar Google Chrome WSDL tətbiqi ilə yoxlanılır.)* Yoxlama prosesini reallaşdırmaq üçün səhifədə qeyd edilən Services Description hissəsindən klik edərək WSDL formatını əldə etmək lazımdır.



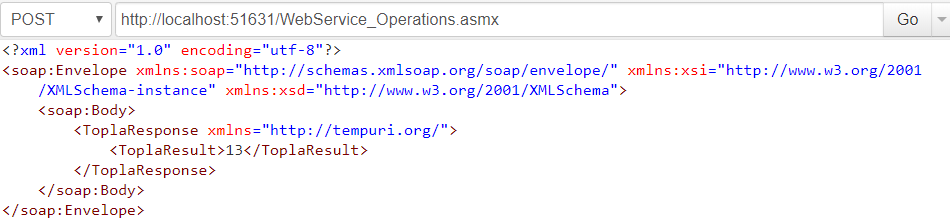
Əldə olunmuş WSDL ilə metodların yoxlanılması təmin olunur. (<http://localhost:51631/WebService_Operations.asmx?WSDL>)

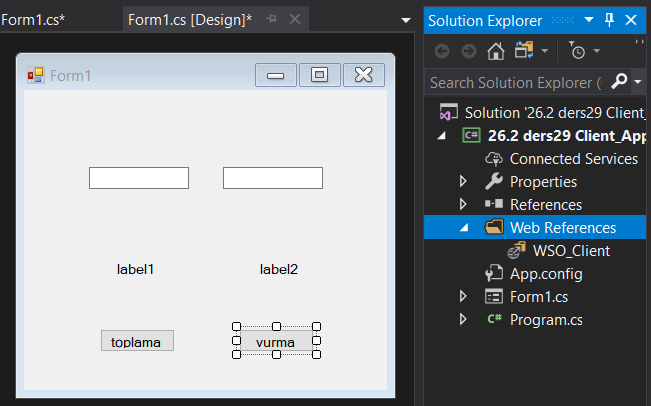


REQUEST



RESPONCE





Hər iki metodun işlədiyinə əmin olduqdan sonra ona uyğun client yaradılır. Client daxilində servisdən istifadə etmək üçün REFERANCE hissəsinə AddWebReferance hisəsindən referans əlavə edirik. Bundan sonra obyekti yaradılır və daxilindəki metodlardan istifadə edilir.

private void button1\_Click(object sender, EventArgs e)

{

int a = Convert.ToInt32(textBox1.Text);

int b = Convert.ToInt32(textBox2.Text);

WSO\_Client.WebService\_Operations op = new WSO\_Client.WebService\_Operations();

int n = op.Topla(a, b);

label1.Text = n.ToString();

}

private void button2\_Click(object sender, EventArgs e)

{

int a = Convert.ToInt32(textBox1.Text);

int b = Convert.ToInt32(textBox2.Text);

WSO\_Client.WebService\_Operations op = new WSO\_Client.WebService\_Operations();

int n = op.Vur(a, b);

label2.Text = n.ToString();

}

#### Qeyd: Client application-nın işləyə bilməsi üçün, servis tərəfinin işlək vəziyyətdə olması lazımdır.

#### Qeyd: Client hissəsinə servis verildikdən sonra client tərəfi yeni bir metod istədiyi alda source tərəfində yeni metod yazılır və client yeni metodu istifadə edə bilmək üçün əlavə edilmiş referance hissəsini UPDATE etmək lazımdır.

# DƏRS 30. (WS-ASMX) ADO.Net – insert, select, update, delete {Prosedur ilə}

## Prosedurlar

### SELECT

CREATE PROC SP\_CUSTOMER\_GET

AS

BEGIN

SELECT \* FROM DBO.CUSTOMER

END

CREATE PROC SP\_CUSTOMER\_SPESIALGET

@ID INT

AS

BEGIN

SELECT CR.CUS\_NAME,CR.SURNAME FROM DBO.CUSTOMER CR WHERE CR.ID=@ID

END

### INSERT

CREATE PROC SP\_CUSTOMER\_INSERT

@N NVARCHAR(20),

@S NVARCHAR(30),

@B\_P NVARCHAR(50),

@G CHAR(1),

@I\_N INT,

@I\_P INT,

@B\_D DATE

AS

BEGIN

INSERT INTO DBO.CUSTOMER VALUES (@N, @S, @B\_P, @G, @I\_N, @I\_P,@B\_D)

END

### UPDATE

CREATE PROC SP\_CUSTOMER\_UPDATE

@ID INT,

@N NVARCHAR(20),

@S NVARCHAR(30),

@B\_P NVARCHAR(50),

@G CHAR(1),

@I\_N INT,

@I\_P INT,

@B\_D DATE

AS

BEGIN

UPDATE DBO.CUSTOMER SET CUS\_NAME = @N, SURNAME = @S, BIRTH\_PLACE = @B\_P, GENDER = @G, IDENTITY\_NO = @I\_N, IDENTITY\_PINCODE = @I\_P, BIRTH\_DATE = @B\_D WHERE ID = @ID

END

### DELETE

CREATE PROC SP\_CUSTOMER\_DELETE

@ID INT

AS

BEGIN

DELETE FROM DBO.CUSTOMER WHERE ID = @ID

END

## Web Services hisəsi

SQL SERVER hissəsində prosedurlar yaradıldıqdan sonra, WS hazırlanması üçün EMPTY web proyekt yaradılır. WS-də baza ilə connection yaratmaq üçün DALC class-ı yaratmaq əvəzinə Web.Config içərisində connectionStrings yaradılır. *(Bu serverdə hər hansı bir dəyişiklik olan zaman məlumatın rahat şəkildə yenilənməsini təmin edir.)*

<connectionStrings>

<add name="constring" connectionString="Data Source=DESKTOP-K25G1RK;Initial Catalog=Gr25DB;Integrated Security=True"/>

</connectionStrings>

Yaradılmış connectionStrings WS daxilində metodlarda istifadə edilə bilməsi üçün local metod yaradılır. Lakin üzərinə WebMethod açar sözü yazılmır. Sonra isə metodlar ardıcıllıqla yazılır.

public string GetConnectionString()

{

string cnstrng = ConfigurationManager.ConnectionStrings["constring"].ConnectionString;

return cnstrng;

}

[WebMethod]

public string SelectCustomerSpesial(int id)

{

string result = "";

SqlConnection con = new SqlConnection(GetConnectionString());

SqlCommand cmd = new SqlCommand("SP\_CUSTOMER\_SPESIALGET", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter ID = new SqlParameter("@ID", SqlDbType.Int);

ID.Direction = ParameterDirection.Input;

ID.Size = 32;

ID.Value = id;

cmd.Parameters.Add(ID);

con.Open();

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

result = dr.GetString(0) + " " + dr.GetString(1);

}

con.Close();

return result;

}

#### Qeyd: Prosedurdan istifadə edilən zaman müqlət formada bu kodun prosedur kodu olduğunu proqrama bildirmək lazımdır. (*mavi ilə işarənmiş sətr*)

#### Qeyd: Prosedurdan istifadə edilən zaman əgər prosedura giriş dəyər vermək tələb olunursa sarı rəng ilə işarələnmiş hissədəki kimi ilk növbədə tələb olunan dəyər və tipinin obyekti yaradılır.

Sonra dəyərin giriş dəyəri olduğu təyin edilir. Növbəti mərhələdə uzunluğu, sonra isə dəyəri metodun giriş dəyəri ilə mənimsədilir. Və son olaraq yaradılmış obyekt parametr olaraq SQLCommand-a əlavə edilir. Commad daxilinə mənimsədilmə prosesi bitdikdən sonra əmrin icra edilməsi həyata keçirilir və SqlDataReader tipində geriyə dönən dəyər dövr daxilinə salınır. Və “dr” dəyərinə məlumat gələnə kimi dövr edir.

[WebMethod]

public string InsertCustomer(string name, string surname, string birthplace, char

gender, int identityNo, int identityPincode, DateTime birthdate)

{

string result = "";

SqlConnection con = new SqlConnection(GetConnectionString());

SqlCommand cmd = new SqlCommand("SP\_CUSTOMER\_INSERT", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter NAME = new SqlParameter("@N", SqlDbType.NVarChar);

NAME.Direction = ParameterDirection.Input;

NAME.Size = 20;

NAME.Value = name;

cmd.Parameters.Add(NAME);

SqlParameter SURNAME = new SqlParameter("@S", SqlDbType.NVarChar);

SURNAME.Direction = ParameterDirection.Input;

SURNAME.Size = 30;

SURNAME.Value = surname;

cmd.Parameters.Add(SURNAME);

SqlParameter BIRTHPLACE = new SqlParameter("@B\_P", SqlDbType.NVarChar);

BIRTHPLACE.Direction = ParameterDirection.Input;

BIRTHPLACE.Size = 50;

BIRTHPLACE.Value = birthplace;

cmd.Parameters.Add(BIRTHPLACE);

SqlParameter GENDER = new SqlParameter("@G", SqlDbType.Char);

GENDER.Direction = ParameterDirection.Input;

GENDER.Size = 1;

GENDER.Value = gender;

cmd.Parameters.Add(GENDER);

SqlParameter IDEN\_N = new SqlParameter("@I\_N", SqlDbType.Int);

IDEN\_N.Direction = ParameterDirection.Input;

IDEN\_N.Size = 32;

IDEN\_N.Value = identityNo;

cmd.Parameters.Add(IDEN\_N);

SqlParameter IDEN\_P = new SqlParameter("@I\_P", SqlDbType.Int);

IDEN\_P.Direction = ParameterDirection.Input;

IDEN\_P.Size = 32;

IDEN\_P.Value = identityPincode;

cmd.Parameters.Add(IDEN\_P);

SqlParameter BIRTHDATE = new SqlParameter("@B\_D", SqlDbType.Date);

BIRTHDATE.Direction = ParameterDirection.Input;

BIRTHDATE.Size = 50;

BIRTHDATE.Value = birthdate;

cmd.Parameters.Add(BIRTHDATE);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

return ex.Message;

}

finally

{

con.Close();

}

result = "iNSERT";

return result;

}

[WebMethod]

public string UpdateCustomer(int id, string name, string surname, string birthplace,

char gender, int identityNo, int identityPincode, DateTime birthdate)

{

string result = "";

SqlConnection con = new SqlConnection(GetConnectionString());

SqlCommand cmd = new SqlCommand("SP\_CUSTOMER\_UPDATE", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter ID = new SqlParameter("ID", SqlDbType.Int);

ID.Direction = ParameterDirection.Input;

ID.Size = 32;

ID.Value = id;

cmd.Parameters.Add(ID);

SqlParameter NAME = new SqlParameter("@N", SqlDbType.NVarChar);

NAME.Direction = ParameterDirection.Input;

NAME.Size = 20;

NAME.Value = name;

cmd.Parameters.Add(NAME);

SqlParameter SURNAME = new SqlParameter("@S", SqlDbType.NVarChar);

SURNAME.Direction = ParameterDirection.Input;

SURNAME.Size = 30;

SURNAME.Value = surname;

cmd.Parameters.Add(SURNAME);

SqlParameter BIRTHPLACE = new SqlParameter("@B\_P", SqlDbType.NVarChar);

BIRTHPLACE.Direction = ParameterDirection.Input;

BIRTHPLACE.Size = 50;

BIRTHPLACE.Value = birthplace;

cmd.Parameters.Add(BIRTHPLACE);

SqlParameter GENDER = new SqlParameter("@G", SqlDbType.Char);

GENDER.Direction = ParameterDirection.Input;

GENDER.Size = 1;

GENDER.Value = gender;

cmd.Parameters.Add(GENDER);

SqlParameter IDEN\_N = new SqlParameter("@I\_N", SqlDbType.Int);

IDEN\_N.Direction = ParameterDirection.Input;

IDEN\_N.Size = 32;

IDEN\_N.Value = identityNo;

cmd.Parameters.Add(IDEN\_N);

SqlParameter IDEN\_P = new SqlParameter("@I\_P", SqlDbType.Int);

IDEN\_P.Direction = ParameterDirection.Input;

IDEN\_P.Size = 32;

IDEN\_P.Value = identityPincode;

cmd.Parameters.Add(IDEN\_P);

SqlParameter BIRTHDATE = new SqlParameter("@B\_D", SqlDbType.Date);

BIRTHDATE.Direction = ParameterDirection.Input;

BIRTHDATE.Size = 50;

BIRTHDATE.Value = birthdate;

cmd.Parameters.Add(BIRTHDATE);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

return ex.Message;

}

finally

{

con.Close();

}

result = "UPDATE";

return result;

}

[WebMethod]

public string DeleteCustomer(int id)

{

string result = "";

SqlConnection con = new SqlConnection(GetConnectionString());

SqlCommand cmd = new SqlCommand("SP\_CUSTOMER\_DELETE", con);

cmd.CommandType = CommandType.StoredProcedure;

SqlParameter ID = new SqlParameter("@ID", SqlDbType.Int);

ID.Direction = ParameterDirection.Input;

ID.Size = 32;

ID.Value = id;

cmd.Parameters.Add(ID);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

result = "DELETE";

return result;

}

## Client hissəsi

private void toolStripButton4\_Click(object sender, EventArgs e)

{

char cins = ' ';

string ad = textBox1.Text;

string soyad = textBox2.Text;

string dogumyeri = textBox3.Text;

int idnomresi = Convert.ToInt32(textBox4.Text);

int idpincodu = Convert.ToInt32(textBox5.Text);

DateTime dogumtarixi = dateTimePicker1.Value;

if (checkBox1.Checked == true)

{

cins = 'K';

checkBox2.Checked = false;

}

else if (checkBox2.Checked == true)

{

cins = 'Q';

checkBox1.Checked = false;

}

ADO\_opertion.WS\_AdoCommand vs = new ADO\_opertion.WS\_AdoCommand();

string message = vs.InsertCustomer(ad, soyad, dogumyeri, cins, idnomresi,

idpincodu, dogumtarixi);

MessageBox.Show(message);

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

int id = Convert.ToInt32(textBox6.Text);

ADO\_opertion.WS\_AdoCommand vs = new ADO\_opertion.WS\_AdoCommand();

string message = vs.SelectCustomerSpesial(id);

MessageBox.Show(message);

}

# DƏRS 31. (WS-WCF) Cbar.az - XML

## Mərkəzi bankın gündəlik valyuta məzənnəsinin çekilməsi

XmlDocument – mərkəzi bankın xml URL daxilindəki məlumatları yükləmək üçün istifadə olunur. Yaradılmış XmlDocument içinə mərkəzi bankın vermiş olduğu xml URL-nin WSDL formasını daxil etdikdən sonra XmlNodeList yaratmaq üçün mövcud olan listləmək metodundan istifadə edilir. (Valute tagName-nə əsasən). Bundan sora giriş kəmiyyəti kimi tələb olunan valyutanın qısa adının XmlNode-dan birinin 0-cı atribuna bərabərliyini yoxlayır. Əgər bu hall bərabər olarsa, root daxilindəki {result} uyğun XmlNode ilə əvəzlənilir və yaradılmış yeni XmlDocument içinə mənimsədilərək çıxış kəmiyyəti kimi return edilir.



public class WS\_GetCurr : System.Web.Services.WebService

{

[WebMethod]

public XmlDocument GetCurrency(string curname)

{

string result = @"<root>{result}</root>";

XmlDocument xmldoc = new XmlDocument();

string date = DateTime.Now.ToString("dd.MM.yyyy");

xmldoc.Load("<https://www.cbar.az/currencies/>" + date + ".xml?WSDL");

XmlNodeList xmllist = xmldoc.GetElementsByTagName("Valute");

foreach (XmlNode item in xmllist)

{

if (item.Attributes[0].Value == curname)

{

result = result.Replace("{result}", item.InnerXml);

}

}

XmlDocument myxml = new XmlDocument();

myxml.LoadXml(result);

return myxml;

}

## WCF – interface anlayışı

Interface-in əsas məqsədi class-ın dublikatını yaradaraq performansın artırılmasını təmin edir. İnterface olmadan da proyektlər yaratmaq mümkündür. Lakin performans nəzərə alınmalı olan hissələrdə İnterface-lərdən istifadə etmək lazım gəlir. Proyekt daxilində Class və İnterface istafə edilməsinin fərqi ondan ibarətdir ki, sadecə class olan zaman həm işin görülməsi həmdə məlumat mübadiləsi class ilə həyata keçirilir, lakin interface olan zaman class yənə öz işini görür amma məlumat mübadiləsi interface vasitəsi ilə icra edilir. Başqa sözlə deyək interface class-ın təmsilçi kimidir. *(Sinif nümayəndəsi, ölkə səfrləri və s.)*

#### Qeyd: Miras almaqdan – (inharitance) fərqli olaraq bir class bir neçə class-dan interface ala bilər.

WCF-lər iki əsas hissədən (\*.svc və I\*.cs) ibarət olur. \*.scv uzantılı hissədə metodlar yazılır. Yazılmış metodların client tərəfə ötürülməsini təmin etmək üçün eyni ad ilə adlantırımış interface class-ı daxilində OperationContract açar sözündən istifadə edərək I\*.cs class-ı daxilində metodun tipi adı və giriş dəyərlər verilir.

public class Service1 : IService1

{

public string GetData(int value)

{

return string.Format("You entered: {0}", value);

}

public int topla(int d1, int d2)

{

int netice = d1 + d2;

return netice;

}

public string vurma(string d1, string d2)

{

int d1copy = Convert.ToInt32(d1);

int d2copy = Convert.ToInt32(d2);

int netice = d1copy \* d2copy;

return netice.ToString();

}

}

[ServiceContract]

public interface IService1

{

[OperationContract]

string GetData(int value);

[OperationContract]

int topla(int d1, int d2);

[OperationContract]

string vurma(string d1, string d2);

}

#### Qeyd: WCF-lər RUN edilən zaman I\*.cs class-ı olan hissəyə gəldindən sonra RUN edilə bilir.

#### Qeyd: WCF-lər daxilində metod yazarkən metod tipinin mümkün dərəcədə string tipində olması lazımdır. Əks halda Client tərəfdə yazılan kodlarda uzanmalar müşahidə edilir.

private void button1\_Click(object sender, EventArgs e)

{

int a = Convert.ToInt32(textBox1.Text);

int b = Convert.ToInt32(textBox2.Text);

int netice = 0;

bool status = true;

WS\_operation.Service1 wss = new WS\_operation.Service1();

wss.topla(a, true, b, true, out netice, out status);

label1.Text = netice.ToString();

}

private void button2\_Click(object sender, EventArgs e)

{

string a = textBox1.Text;

string b = textBox2.Text;

WS\_operation.Service1 wss = new WS\_operation.Service1();

string net = wss.vurma(a, b);

label1.Text = net;

}

#### Qeyd: int tipində olan metod client hissədə çağırılan zaman VOID metod kimi əks edildiyindən əlavə kodlar tələb olunur.

Kurs və əlavə məlumatlar

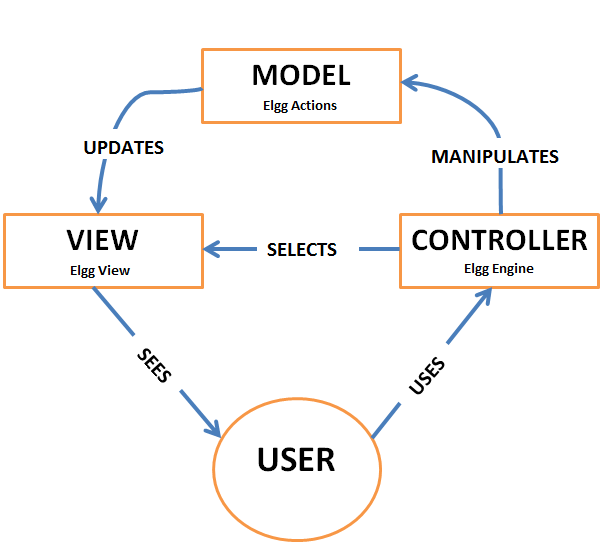
**ORXAN MƏMMƏDƏLİ**

**BACKEND**

**“MVC – ASP.NET”  
“ORM”-ENTITY FRIMEWORK**

# DƏRS 32. (MVC-Model View Controller)

* **View** – front hisssə *(istifadəçiyə görünən hissə)*
* **Model** – Baza ilə əlaqə hissəsi *(DB)*
* **Controller** – View və Model arasında əlaqəni təmin edən, proyektin əsas hissəsi *(sütunu)*



İstifadəçiyə məlumatlar View-lar tərəfindən əks olunur. İstifadəçi səhifədə hər hansı bir proses icra edən zaman Controller-lərə müraciət etmiş olur. Controllerə müraciət olunan yazman iki tip metodlardan istifadə edilir (bazaya müraciət edən və etməyən). Əgər bazaya müraciət etməyən bir əmr icra olunursa, nəticə bir başa view hissəsinə əks olunur. Əks halda controller modellərə müraciət edir. Modellər eyni zamanda baza ilə əlaqə yaradır və lazımı məlumatları controllerdəki metodun tələblərinə uyğun olaraq view əks etdirir.

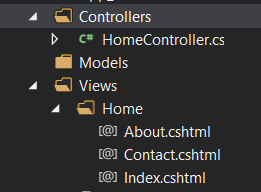
Tətbiqlərimizdə, hazırladığımız layihələrdə verilənlər bazasına qoşulmaq lazım olduqda, tez-tez ADO.Net adlanan metoddan istifadə etdik. Verilənlər bazasına birbaşa tətbiqetmədən qoşulduq və SQL sorğuları ilə verilənlər bazası əməliyyatları apardıq. ADO.net-in mürəkkəbliyini asanlaşdıra bilən və proqram texnologiyalarının, obyekt yönümlü proqramlaşdırma əsaslı dillərin inkişafı və geniş yayılması ilə oxunan kod xətlərini daha çox yaza bilən ORM (Object Relational Mapping) strukturları meydana gəldi.

**ORM nədir?**

Ado.net istifadə edən layihələrdə, verilənlər bazası ilə əlaqəli əməliyyatlarda, proqram təminatçısı birbaşa sql sorğularını yazmaq məcburiyyətindədir. ORM yaranması ilə yazılmış SQL kodları minimuma enir. Verilənlər bazamızdakı cədvəllər bir class olaraq təyin olunur və sütundakı hər bir sahəmiz bir property olaraq təyin olunur və verilənlər bazasındakı qeydlərə aid olduğu sinifdə bir obyekt olaraq daxil olub istifadə edə bilərik.

public class HomeController : Controller

{

 // GET: Home

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

</head>

<body>

<div>

<h1> MVC giris dersi </h1>

<h3> Khan </h3>

@{

int[] deyer = new int[100];

Random eded = new Random();

for (int i = 0; i < 100; i++)

{

deyer[i] = eded.Next(1, 10);

@deyer[i] <br/>

}

}

@Html.ActionLink("Haqqımızda","About","Home")

@Html.ActionLink("Əlaqə","Contact","Home")

</div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>About</title>

</head>

<body>

<div>

<h1>Haqqımızda</h1>

@Html.ActionLink("Ana Səhifə", "Index", "Home")

</div>

</body>

</html>

# DƏRS 33-34. (Master page – model class)

## Master page

Master page CONTROLLERLƏRDƏN və ya ACTİONRESAULTDAN bağımsız olan bir səhifədir. Əsasən səhifələrin HEADER və FOOTER hissələrini özüdə əks etdirir. Digər səhifələr isə onun daxilinə RENDERBODY hissəsində əks olunur. Başqa sözlə ifadə etsək master page anlayışı artıq kod yazmaqdan qaçınmaq üçün yaradılmışdır.

Master page yardılan zaman ilkin olaraq Layout pəncərəsi yaradılır. Yaradılan pəncərə daxilində html kod kombinasiyasını bildirəm HTML TAG-i, onun daxilində isə ayırmaq üçün istifadə edilmiş HEAD, BODY TAG-ları yer alır. Head hissesində standart olaraq başlıq, Body hissəsində isə səhifənin başlıq hissəsi olan HEARDER, Səhifənin son hissəsi olan FOOTER və digər səhifələrin görülməsini təmin edilən sahə olan @RenderBody()-dən ibarətdir. (*Digər səhifələr çağırılan zaman hər biri Layout daxilindəki kodlardan istifadə edir*).

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Deneme</title>

</head>

<body>

<header>

<div>

<b> Khan</b>

<div>

<div>

<h2> Bashliq </h2>

</div>

<div>

@Html.ActionLink("Ana Sehife", "Index", "Home");

@Html.ActionLink("Haqqinda", "About", "Home");

@Html.ActionLink("Elaqe", "Contact", "Home");

</div>

</div>

</div>

</header>

<body>

<div>

@RenderBody()

</div>

</body>

<footer>

<div>

<h3> Copy writer by Khna 2020</h3>

</div>

</footer>

</body>

</html>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<div style="width:400px; height:400px; background-color:burlywood; color:skyblue; font-size:20px; font-family:'Palatino Linotype'">

Bu ana sehifedir

</div>

@{

ViewBag.Title = "About";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>About</h2>

<div style="width:400px; height:400px; background-color:goldenrod; color:lightseagreen; font-size:20px; font-family:'Palatino Linotype'">

Bu haqqinda sehifedir

</div>

@{

ViewBag.Title = "Contact";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Contact</h2>

<div style="width:400px; height:400px; background-color:greenyellow; color:aquamarine; font-size:20px; font-family:'Palatino Linotype'">

Bu elaqe sehifedir

</div>

## Model Class

(Yaradılma ardıcıllığı: )

1. Proyekt daxilinə “Data” adında qovluq əlavə edilir.
2. Model qovluğu daxilində “Student” class-ı yaradılır.

public class Student

{

public int ID { get; set; }

public string Name { get; set; }

public string Surname { get; set; }

public string Identityno { get; set; }

public string FacultyName { get; set; }

public int EnteredYear { get; set; }

}

1. Data qovluğu daxilində “StudentDb” adında class yaradılır. *List formatında məlumat üçün*

public class StudentVb

{

public static List<Student> Students = new List<Student>

{

new Student

{

ID = 100,

Name = "Orxan",

Surname = "Məmmədəli",

Identityno = "1234asc",

FacultyName = "IT",

EnteredYear = 2006,

},

new Student

{

ID = 101,

Name = "Tahir",

Surname = "Zahirli",

Identityno = "4321axs",

FacultyName = "IT",

EnteredYear = 2009,

}

};

}

1. Controller qovluğu daxilinə “StudentController” əlavə edilir.
2. Yaradılmış controller daxilində aşağıdaki metodlar ardıcıllıqla yazılır.

public class StudentController : Controller

{

[HttpGet]

public ActionResult Listle()

{

return View(StudentVb.Students);

}

Listlə əməliyyatında View hissəsinə StudentDb.Students məlumatlarını göndəririk və rahatlıqla ekranda əks olunur.

[HttpGet]

public ActionResult New()

{

return View();

}

Yeni məlumat daxil etmə hissəsində get metodu daxilinə heç bir məlumat göndərilmir. Çünki həmin metod birbaşa olaraq view yaradılan zaman “create-tamplate” və “model-student” class-dan istifadə edildiyinə görə məlumatlara uyğun formada avtomatik yaradılır.

[HttpPost]

public ActionResult New(Student newStudent)

{

StudentVb.Students.Add(newStudent);

return RedirectToAction("Listle");

}

Post əməliyyatı zamanı göndərilən “Student” tipində məlumatlar Data hissəsindəki listə mənimsədilir və eyni zamanda RedirectToAction funksiyası ilə “listle” hissəsi əks olunur.

[HttpGet]

public ActionResult Edit(int id)

{

var student = StudentVb.Students.Where(o => o.ID == id).FirstOrDefault();

return View(student);

}

Məlumat seçilən zaman onun id-si tutulur və içəri göndərilir. Və Ona uyğun olaraq Listə mənimsədilmiş məlumatı ekrana gətirir. Geri dönən dəyərin tipləri müxtəlif olduğundan həmin məlumatlar bir başa var tipinə mənimsədilir.

[HttpPost]

public ActionResult Edit(Student student)

{

Student editstudent = StudentVb.Students.Where(o => o.ID ==

student.ID).FirstOrDefault();

editstudent.Name = student.Name;

editstudent.Surname = student.Surname;

editstudent.Identityno = student.Identityno;

editstudent.FacultyName = student.FacultyName;

editstudent.EnteredYear = student.EnteredYear;

return RedirectToAction("Listle");

}

Post edilən zaman “student” tipində gələn dəyərlərin ID-nə uyğun Köhnə məlumatların Id-ləri bərabər olduğuna görə köhnə məlumatın (Student editstudent) yeni məlumatla (Student student) əvəzlənməzi icra olunur. Və eyni zamanda RedirectToAction funksiyası ilə “listle” hissəsi əks olunur.

[HttpGet]

public ActionResult Delete(int id)

{

var student = StudentVb.Students.Where(o => o.ID == id).FirstOrDefault();

return View(student);

}

Məlumat seçilən zaman onun id-si tutulur və içəri göndərilir. Və Ona uyğun olaraq Listə mənimsədilmiş məlumatı ekrana gətirir.

[HttpPost]

public ActionResult Delete(Student student)

{

Student deletestudent = StudentVb.Students.Where(o => o.ID ==

student.ID).FirstOrDefault();

StudentVb.Students.Remove(deletestudent);

return RedirectToAction("Listle");

}

[HttpGet]

public ActionResult Details(int id)

{

var student = StudentVb.Students.Where(o => o.ID == id).FirstOrDefault();

return View(student);

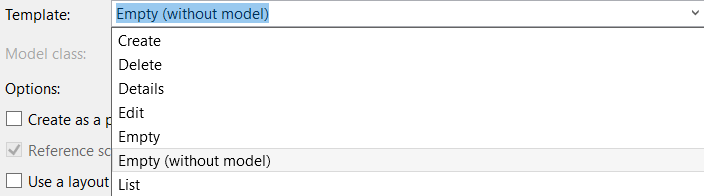
}

}

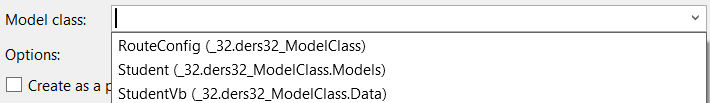
Details zamanı göndərilən ID – yə əsasən məlumatları return edir.

#### Qeyd: Edit, Delete və Details metodlarının “Get – ActionResualt”-ları daxilində yazılan kodlar eynidir. Lakin onları hamısının əvəzinə bir dəfə kod yazmaq doğru hall hesab edilmir.

1. Sonra “View” yaradılan zaman hər bir [HttpGet] qeyd edilən metodan istifadə edilir.
   1. View əlavə edilən zaman hazır formatın əldə edilməsi üçün “Tamplate” hissəsindən ada uyğun olaraq tamplate seçilir.



* 1. Tamplate seçiləndən sonra Model class bölməsindən Model qovluğu daxilində olan class seçilir. Və bu proses hər biri üçün icra edilir.



1. Nəticədə avtomatik olaraq View-lar yaradılır. Yaradılan View daxilində ilkin olaraq “Listle” daxilində @Html.ActionLink("New Student", "New") dəyişikliyi edilir.
2. Digər view-lar daxilində “BacktoList” hissələrində isə @Html.ActionLink("Back to List", "Listle") dəyişikliyi edilir. Və bütün errorlar aradan qalxır.

#### Qeyd: Bir Action içində başqa bir action-ı çağırmaq üçün “return RedirectToAction ("Listle")” funksiyası istifadə edilir. Bu kodun icrası zamanı ilkin olaraq ActionResult öz funksiyasını icra edir. Sonra isə Listlə ActionResultunu işlədir. Başqa sözlə desək iki əməliyyatı eyni zamanda icra edir.

#### Qeyd: Məlumatları yadda saxlamaq üçün model class Data hissəsindəki list əvəzinə data baza ilə əlaqədə olmalıdır.

# DƏRS 35. (Model class-kiçik nümunə)

Model qovluğu daxilinde “Emel” adında class yaradılır. Yaradılan class daxilində property və method daxil edilir.

public class Emel

{

public int deyer1 { get; set; }

public int deyer2 { get; set; }

public int netice { get; set; }

public int topla(int val1,int val2)

{

int result = val1 + val2;

return result;

}

}

#### Qeyd: Model daxilində yaradılan class içində metod yaratmaq doğru hall hesab edilmir. Metodlar Controller tərəfində yaradılmalıdır. *(Burada metod yaratmaqda məqsəd controller tərəfdə çağrılma prosesini göstərməkdir)*

Növbəti addımda controller yaradılır. Yaradılmış controller daxilində iki ədəd “Get və Post” olmaqda “ActionResult”-lar yaradılır. Post metodu daxilində məlumatları əldə etmək üçün “Emel” classının using-I çağrılır və clasın “emel” adında obyekti yaradılır. Yaradılmış obyekt ilə metod daxilində class-ın atributları və metodlarını görmək mümkün olur. Son olaraqda View daxilində yaradılmış obyekt return edilir.

public class HomeController : Controller

{

// GET: Home

[HttpGet]

public ActionResult Index()

{

return View();

}

[HttpPost]

public ActionResult Index(Emel emel)

{

emel.netice = emel.topla(emel.deyer1, emel.deyer2);

// emel.netice = emel.deyer1 + emel.deyer2;

return View(emel);

}

}

#### Qeyd: HttpGet ilk səhifəni qaytarır. HttpPost əməlyatın icrasını reallaşdırır.

#### Qeyd: Post metod daxilinə giriş məlumatları göndərən zaman onların hər bizinin yazılması doğru hall hesab edilmir. Belə olan halda göndərin məlumatlara uyğun Model qovluğunda class yaradılır *(Misal olaraq Emel class-ı)* və giriş kəmiyyəti olaraq həmin clas-ın obyekti yaradılır. Sonradan metod daxilində həmin obyekt ilə lazımi atributlar çağrılır.

Növbəti addımda View yaratmaq üçün “Tamplate => Empty, Model class => Emel” seçilərək view yaradılır. Yaradılmış view boş olduğuna görə onu class əmrlərini icra etmək üçün Html və C# *(@ - işarəsindən istifadə edilən kodlar)* kod kombinasıyasından istifadə edilir. İlk əvvəl “BeginForm” yaradır *(using (Html.BeginForm("Index", "Home", FormMethod.Post)))* və C# kodlarını istifadə etmək üçün konstruktor metod *(@Html.AntiForgeryToken())* çağrılır. Bundan sonra hər bir atribunt üçün Label *(@Html.LabelFor(model => model.deyer1))* və Textbox *(@Html.EditorFor(model => model.deyer1))* yaradılır. Yaradılmış “Textbox-editor for” kənardan gələn dəyərin nəyə mənimsədilməsini təyin etmək üçün, modelin obyekti ilə həmin atributa mənimsədilir *((model => model.deyer1))*. Son olaraq daxil edilmiş dəyərlərə uyğun olaraq “post” metodunun işləməsi üçün input parametr olan “Button – submit” yaradılır. Hər iki metod işlədikdən sonra əgər model boş dəyər göndərməzsə, nəticə ekrana əks etdirilir.

@model \_33.ders35\_ModelClass\_kicik\_Numune.Models.Emel

@{

Layout = null;

}

@using (Html.BeginForm("Index", "Home", FormMethod.Post))

{

@Html.AntiForgeryToken()

// HTML Kodlari hissesinde C# komponentlerinin hazirlanmasini temin edir.

<div class="editor-label">

@Html.LabelFor(model => model.deyer1)

</div>

<div class="editor-label">

@Html.EditorFor(model => model.deyer1)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.deyer2)

</div>

<div class="editor-label">

@Html.EditorFor(model => model.deyer2)

</div>

<div class="form-group">

<input type="submit" value="Topla" id="Submit" class="btn btn-default"/>

</div>

}

<div>

@if (Model != null)

{@Html.DisplayFor(model => model.netice)

</div>

#### Qeyd: Html səhifəsində modeldən istifadə etmək üçün onun obyekti yatadılmalıdır. @modelProyektAdi.ClassAdi

# DƏRS 36. (ORM – Object Relational Mapping) LinqToSql

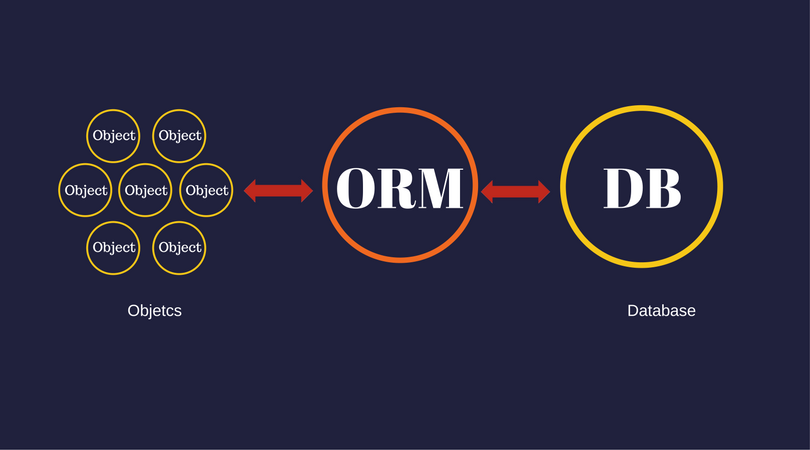
* ORM – obyektlərlə işdir. Baza ilə görülən işlərin yerinə yetirilməsini təmin edir.
* ORM ADO.Net vasitəsi ilə yazılan kodların peşəkar formasıdır. *(ADO.Net ilə çalışılsa ORM-dən serətli olan kodlar yazıla bilər)*
* ORM sadecə C sharp deyil eyni zamanda Java, PHP və s. dillərdə də mövcuddur.
* ORM-nin Csharpda işləməsini təmin edən framework EntityFramework-dür.
* ORM ilə iki üsulla DB-yə bağlanmaq mümkündür. *(1. Model First – CodeFirst, 2. DbFirst - LingToSQL)*

**ÜSTÜNLÜKLƏR**

* ORM bilən şəxsin SQL bilməsi o qədər də vacib hesab edilmir.
* ORM təkrar kod yazmanın qarşısını alır.

**ÇATIŞMAZLIQLARI**

* ORM o zaman istifadə edilməlidir ki, əlaqədə olduğu DB tez-tez dəyişilmir və ya daha peşakar bir sistem mövcuddur. Əgər Cədvəllər tez - tez dəyişərsə o zaman ORM hissəsindədə dəyişikliklər edilməli olur bu isə əlavə kodlamaya lazım olur.
* ADO.Net dən zəif işləyir. Çünki ADO.Net ilə yazılmış sistemə təkminləşdirmə üçün müdaxilə etmə imkanına malik olmasıdır.



## Linq To SQL

LinqToSql vasitəsi ilə əlaqə yaratmaq üçün ilkin olaraq DB tərəfdə cədvəl və ya DB hazırlanır. DB tərəfdı hazırlıq prosesi tam bitdikdən sonra, Proyekt daxilində “Add new item => Data => LinqToSql Classes” ardıcıllığı ilə LinqToSql Class-ı əlavə edilir. Sonra “Server Explore => Data Connections => Add connection” pəncərəsindən “Server name və Database Name” seçilirək əlaqə yaradılır. “Server Explore” pəncərəsindən yaradılmış connection-nun üzərinə klik edərək lazımı cədvəl seçilir və boş sahəyə buraxılır. Bu proseslərin icrasından sonra LinqToSql classı daxilində cədvəlin adı ilə adlanmış və əlavə olaraq “Context” classı yaradılır.

#### Qeyd: Cədvəlin əlavə edilməsi üçün mütləq formada “Primary Key”-ə malik olan sütün olmalıdır.

#### Qeyd: Cədvəlin uyğun olan class və ya class-lar obyekt class-larıdır və onlar bazadakı eyni adlı cədvəli xarakterize edir. Context class-ı isə DB ilə edilən əməliyyatları özündə saxlayan sinifdir.

private void Form1\_Load(object sender, EventArgs e)

{

DBEntityDataContext Dbctx = new DBEntityDataContext();

dataGridView1.DataSource = Dbctx.PRODUCTs;

}

Form yüklənən kimi məlumatların Bazadan çəkilərək əks olunması üçün “Context” clasının obyekti yaradılır və həmin obyet daxilindəki əks olunanaq cədvələ DataGridView -un Source-na mənimsədilir.

private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

DataGridViewRow rov = dataGridView1.CurrentRow;

textBox1.Tag = rov.Cells["ID"].Value;

textBox1.Text = rov.Cells["NAME"].Value.ToString();

textBox2.Text = rov.Cells["CATEGORY"].Value.ToString();

textBox3.Text = rov.Cells["MADED\_BY"].Value.ToString();

dateTimePicker1.Text = rov.Cells["PRODUCT\_DATE"].Value.ToString();

numericUpDown1.Value = (decimal)rov.Cells["PRICE"].Value;

}

private void button1\_Click(object sender, EventArgs e)

{

PRODUCT prd = new PRODUCT();

prd.NAME = textBox1.Text;

prd.CATEGORY = textBox2.Text;

prd.MADED\_BY = textBox3.Text;

prd.PRODUCT\_DATE = dateTimePicker1.Value;

prd.PRICE = numericUpDown1.Value;

DBEntityDataContext Dbins = new DBEntityDataContext();

Dbins.PRODUCTs.InsertOnSubmit(prd);

Dbins.SubmitChanges();

dataGridView1.DataSource = Dbins.PRODUCTs;

}

Məlumatların insert edilməsi zamanı ilkin olaraq giriş dəyərləri cədvələ uyğun class-ın obyetri yaradılır və girilən dəyərlər class-ın atributlarına mənimsədilir. Bundan sonra Context class-nın obyekti yaradılır və obyekt daxilində olan cari class-ın uzantısına “InsertOnSubmit” metodu ilə mənimsədilir. Metodun icra olunması üçün “SubmitChanges” – başqa şəkildə ifadə etsək təsdiq əmri yazılır.

private void button2\_Click(object sender, EventArgs e)

{

int id = Convert.ToInt32(dataGridView1.CurrentRow.Cells["ID"].Value);

DBEntityDataContext dbdelete = new DBEntityDataContext();

PRODUCT prod = dbdelete.PRODUCTs.SingleOrDefault(o => o.ID == id);

dbdelete.PRODUCTs.DeleteOnSubmit(prod);

dbdelete.SubmitChanges();

dataGridView1.DataSource = dbdelete.PRODUCTs;

}

Məlumatların delete edilməsi zamanı ilkin olaraq seçilmiş sətr üzrə məlumatın Id-si bir dəyərə mənimsədilir. Sonra cari cədvəli ifadə edən class-ın obyekti yaradılır və giriş ID bərabər ID si olan sətr obyektə mənimsədilir. Növbəti mərhəldə Context clasıının obyeti ilə “DeleteOnSubmit” metodu ilə həmin sətr silinir və əmrin icrasının təsdiqi üçün “SubmitChanges” metodundan istifadə edilir.

private void button3\_Click(object sender, EventArgs e)

{

DBEntityDataContext dbupdate = new DBEntityDataContext();

int id = Convert.ToInt32(textBox1.Tag);

PRODUCT prd = dbupdate.PRODUCTs.SingleOrDefault(o => o.ID == id);

prd.NAME = textBox1.Text;

prd.CATEGORY = textBox2.Text;

prd.MADED\_BY = textBox3.Text;

prd.PRODUCT\_DATE = Convert.ToDateTime(dateTimePicker1.Text);

prd.PRICE = numericUpDown1.Value;

dbupdate.SubmitChanges();

dataGridView1.DataSource = dbupdate.PRODUCTs;

}

Məlumatların update edilməsi zamanı bütün proses Delete prosesindeki kimi davam edir. Əsas fərq ondan ibarətdir ki, Seçilmiş sətrin ID nə uyğun köhnə məlumatlar ilə daxil edilmiş yeni məlumatlar arasında mənimsətmə prosesi icra edilir.

private void button4\_Click(object sender, EventArgs e)

{

int prodid = Convert.ToInt32(dataGridView1.CurrentRow.Cells["ID"].Value);

MessageBox.Show(prodid.ToString());

}

Sətrin ID -ni ekrana çıxarır.

# DƏRS 37. (ORM – Code First)

Code First ilə ilkin olaraq proqram daxilində cədvəli xarakterizə edən class və ona uyğun context classı yaradaraq DB və daxilindəki cədvəlin (cədvəllərin) yaradılmasını təmin etməkdir. Bu prosesi icra etmək üçün aşağıdakı ardıcıllığı yerinə yetirmək lazımdır.

* 1. Proyektə “Entity Framework” paketi yüklənilir.
  2. Web.Config faylı daxilində yaradılır.

<connectionStrings>

<add name="constring" connectionString="Data Source=DESKTOP-K25G1RK;Initial

Catalog=CodeFirstDb;Integrated Security=True;"

providerName="System.Data.SqlClient"/>

</connectionStrings>

* 1. Model class-ı daxilində cədvəli xarakterizə edən class və ona uyğun olaraq DbContext class-ı yaradılır.

public class User

{

public int ID { get; set; }

public string NAME { get; set; }

public string SURNAME { get; set; }

}

public class UserDbContext:DbContext

{

public UserDbContext()

: base("constring")

{

}

public DbSet<User> users { get; set; }

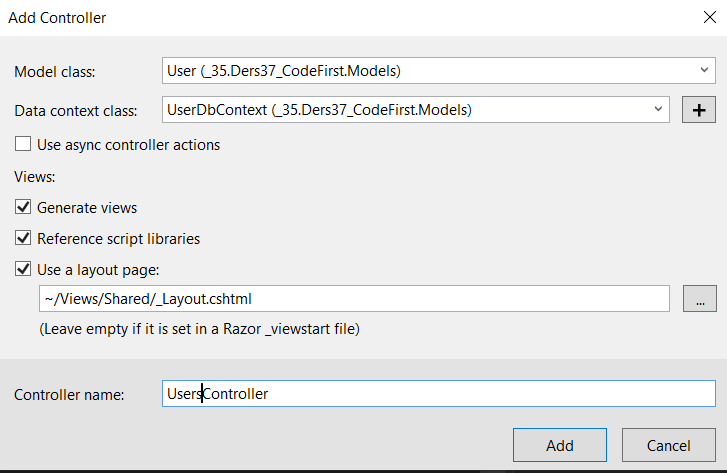
}

DbContext class-ı kimi yaradılmış class mütləq formada Entiti F. NameSpace-də olan “DbContext” class-dan miras almalıdır. Class miras aldıqdan sonra constructor metod yaradılır və metoda “connectionStrings”-in adı verilir. Sonra “DbContext” class-nın “DbSet” metodundan istifadə edilərək məlumatları yazılması təmin edilir.

* 1. Model class-ı və uyğun DbContext class yaradılandan sonra bu iki class əsasında Controller əlavə edilir. Bu zaman avtomatik olaraq Controller daxili metodlar və onlara uyğun View-lar avtomatik olaraq yaradılır.

“Controllers => Add => Controller => MVC 5 Controller with views, using Entity Framework”

* + 1. Model class: User
    2. Data Context class: UserDbContext



## CodeFirst Engin Demiroğ

# DƏRS 38. ()

# DƏRS 39. (Object to XML, ExcellExport)

Obyekt faylı XML-ə çevirmək üçün aşağıdakı kod kombinasiyasından istifadə edilir. İlkin olaraq “Main” və “Invoice” class-ları yaradılır. Yaradılmış “Main” class-ı daxilində “Invoice” class-nın atributları List formatında saxlanılır. Buna səbəb eyni adamın adına müxtəlif protokol nömrələrinə malik cərimələrin ola bilməsidir (*əlaqə növü birin - çoxa*). Class-lar yaradıqdıqdan sonra Form daxilində metod və button kodları yazılır.

public class Invoice

{

public string protocol { get; set; }

public int amount { get; set; }

public string currency { get; set; }

}

public class Main

{

public string name { get; set; }

public string surname { get; set; }

public string dlno { get; set; }

public List<Invoice> invoices = new List<Invoice>();

}

// DİQQƏT main class mütləq public olmalıdır. Main public olan zaman avtomatik olaraq invoice class-ı public olmalıdır.

#### Qeyd: Yaradılmış class-lar public olmazsa metod işləməyəcəkdir.

public static string GetXMLFromObject(object o)

{

StringWriter sw = new StringWriter();

XmlTextWriter tw = null;

try

{

XmlSerializer serializer = new XmlSerializer(o.GetType());

tw = new XmlTextWriter(sw);

serializer.Serialize(tw, o);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

sw.Close();

if (tw != null)

{

tw.Close();

}

}

return sw.ToString();

}

private void button1\_Click(object sender, EventArgs e)

{

Main mn = new Main();

mn.name = "Orxan";

mn.surname = "Memmedeli";

mn.dlno = "KH099123";

mn.invoices = new List<Invoice>()

{

new Invoice() {protocol = "MMX52623", amount = 50, currency = "AZN"},

new Invoice() {protocol = "MMX32600", amount = 30, currency = "AZN"},

new Invoice() {protocol = "MMX40065", amount = 20, currency = "AZN"}

};

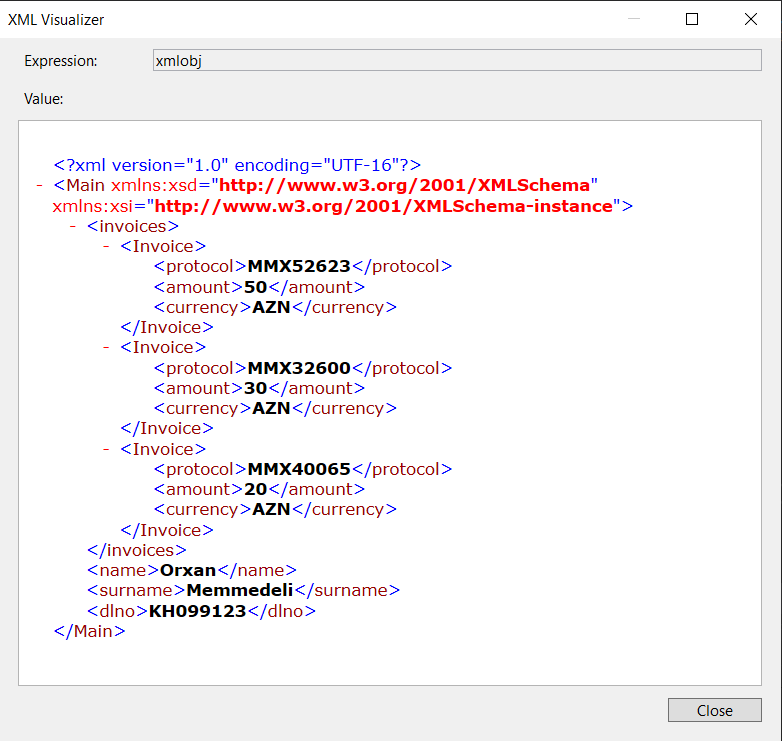
string xmlobj = GetXMLFromObject(mn);

File.AppendAllText(@"C:\Users\memme\source\log\ders39\log-XML.txt", xmlobj);

MessageBox.Show("Test");

}

XML formatında mətni yazalamaq üçün string məlumatı log üsulu ilə “txt” faylı daxilinə yaza və ya “Debug” prosesi zamanı mövcud üsullarla yaxalamaq olar.



Txt daxilində tək sətr kimi yazılmış XML kodlarını səqliqəli formada görmək üçün <https://codebeautify.org/xmlviewer> web səhifəsindən istifadə etmək olar.

## Excel Export

Class daxilidə Məlumatların saxlanılması çekilməsi üçün və export prosesinin içrası üçün iki ədəd metod yazılır.

public static class sinif

{

public static void ExportToExcel(this DataTable tbl, string excelFilePath = null)

{

try

{

if (tbl == null || tbl.Columns.Count == 0)

throw new Exception("ExportToExcel: Null or empty input table!\n");

// load excel, and create a new workbook

var excelApp = new Microsoft.Office.Interop.Excel.Application();

excelApp.Workbooks.Add();

// single worksheet

Microsoft.Office.Interop.Excel.\_Worksheet workSheet = excelApp.ActiveSheet;

// column headings

for (var i = 0; i < tbl.Columns.Count; i++)

{

workSheet.Cells[1, i + 1] = tbl.Columns[i].ColumnName;

}

// rows

for (var i = 0; i < tbl.Rows.Count; i++)

{

// to do: format datetime values before printing

for (var j = 0; j < tbl.Columns.Count; j++)

{

workSheet.Cells[i + 2, j + 1] = tbl.Rows[i][j];

}

}

// check file path

if (!string.IsNullOrEmpty(excelFilePath))

{

try

{

workSheet.SaveAs(excelFilePath);

excelApp.Quit();

MessageBox.Show("Excel file saved!");

}

catch (Exception ex)

{

throw new Exception("ExportToExcel: Excel file could not be saved! Check filepath.\n" + ex.Message);

}

}

else

{ // no file path is given

excelApp.Visible = true;

}

}

catch (Exception ex)

{

throw new Exception("ExportToExcel: \n" + ex.Message);

}

}

public static DataTable GetDataTable()

{

#region connected get

SqlConnection con = new SqlConnection();

con = new SqlConnection("Data Source=DESKTOP-K25G1RK;Initial Catalog=Gr25DB;Integrated Security=True");

con.Open();

SqlCommand cmd = new SqlCommand();

cmd = new SqlCommand("Select \* from Customer", con);

if (con.State == ConnectionState.Closed)

{

con.Open();

}

SqlDataReader dr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(dr);

con.Close();

return dt;

#endregion connected

}

}

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

DataTable dt = sinif.GetDataTable();

sinif.ExportToExcel(dt, null);

}

private void Form1\_Load(object sender, EventArgs e)

{

dataGridView1.DataSource = sinif.GetDataTable();

}

}

## Excel Export (DataGridView -> Excel)

DataGridView-ya çıxarılmış məlumatları excell faylına əlavə etmək üçün üçün ilk növbədə Entity Framework vasitəsi ilə baza ilə əlaqə yaradılır. Növbəti addımda “References => Add Reference => COM => Microsoft Excel 16.0 object Library” referansı əlavə edilir və aşağıdakı kod kombinasiyası yazılır.

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

HilalKursuEntities db = new HilalKursuEntities();

private void button1\_Click(object sender, EventArgs e)

{

Microsoft.Office.Interop.Excel.\_Application app = new

Microsoft.Office.Interop.Excel.Application();

\_Workbook wb = app.Workbooks.Add(Type.Missing);

\_Worksheet ws = null;

ws = wb.Sheets["Sheet1"];

ws = wb.ActiveSheet;

ws.Name = "Khan";

for (int i = 0; i < dataGridView1.Columns.Count; i++)

{

ws.Cells[1, i+1] = dataGridView1.Columns[i].HeaderText;

}

for (int i = 0; i < dataGridView1.Rows.Count; i++)

{

for (int j = 0; j < dataGridView1.Columns.Count; j++)

{

ws.Cells[i + 2, j + 1] = dataGridView1.Rows[i].Cells[j].Value;

}

}

var sfd = new SaveFileDialog();

sfd.FileName = "Output";

sfd.DefaultExt = ".xlsx";

if (sfd.ShowDialog() == DialogResult.OK)

{

wb.SaveAs(sfd.FileName, Type.Missing, Type.Missing, Type.Missing,

Type.Missing, Type.Missing, XlSaveAsAccessMode.xlExclusive, Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing);

}

app.Quit();

}

private void Form1\_Load(object sender, EventArgs e)

{

dataGridView1.DataSource = db.IV\_SINIF.ToList();

}

}

# Kurumsal Mimaride ASP.NET MVC Dersleri

Peşəkar formada MVC layihə hazırlayan zaman diqqət edilməsi gərəkən ən başlıca amil Layihənin bir birindən bağımsız amma bir birinə inteqrə edilmiş formada işləməsini təmin etməkdir. Bu səbəblə layihə yaradılan zaman ilkin olaraq layihədən asılı olmayaraq aşağıdakı proseslər icra edilməlidir. Adlandırma Standartlarına riayət edilir.

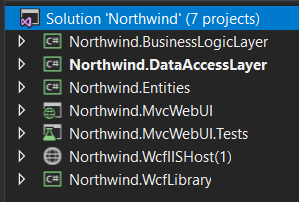
## Layihələndirmə

* Visual Studio hissəsində Boş “Solution” pəncərəsi əlavə edilir. (Name: Northwind)

1. “Data Access Layer-baza ilə əlaqə” üçün və eyni zamanda daxilində interface və Baza ilə işləmə kodlarını saxlayan “Class library” proyekti yüklənilir. *(VS-2017 [.Net Framework])* (Name: Northwind.DataAccessLayer)

#### Qeyd: Bu sahəsə proyektin çox vacib sahələrindən biridir. Misal olaraq Sistemin baza ilə əlaqəsi ilkin olaraq Entity Framework ilə yaradılır. Sonradan performansın artırılması məqsədi ilə ADO.Net kodlarından istifadə tələbi yarandıqdan sadecə buradan yeni kodları əlavə edərək sistemə heç bir müdaxilə edilməməni təmin edir. Əksi baş verərsə sistem yenidən layihələndirilərək bu tələblərə uyğunlaşdırılmalıdır.

1. “Business Logic Layer” üçün “Class library” proyekti yüklənilir. *(VS-2017 [.Net Framework])* (Name: Northwind.BusinessLogicLayer)
2. “WCF Service Library”-Service proyekti əlavə edilir. Servisin olması müxtəlif dillərdəki client-ların hər birinin rahatlıqla proqramla işləyə bilməsi üçündür. (Name: Northwind.WcfLibrary)
3. Proyekti ayaga qaldirmaq üçün “Web” proyekt daxilindəki “WCF Service” proyekti əlavə edilir. (Name: Northwind.WcfIISHost)
4. Baza hissəsindəki cədvəllərə uyğun olan classları özündə saxlamaq üçün yaradılan “Northwind.Entities” proyekti əlavə edilir. (Name: Northwind.Entities)
5. Son olaraq istifadəçi interfeysi əlavə etmək üçün ASP.Net MVC layihəsi əlavə edilir. İstifadə olunan layihədə ümumi məntiqi anlamaq üçün MVC proyektin hazır imkanlar tətbiq edən və ona uyğun olaraq test proyektin yaradılmasını təmin edirik. (Name: Northwind.MvcWebUI; Northwind.MvcWebUI.Tests)



**2**

**1**

**5**

**6**

**6**

**4**

**3**

## Textini tapşırığın hazırlanması

Layihələndirmə prosesi bitdikdən sonra Northwind.Entities proyekti daxilində Database-də yer alan cədvəllərə uyğun classlar yaradılır. Cədvəlin bütün sütunlarından istifadə edilməmişdir.

public class Category

{

public int CategoryID { get; set; }

public string CategoryName { get; set; }

}

public class Customer

{

public string CustomerID { get; set; }

public string CotactName { get; set; }

public string CompanyName { get; set; }

public string City { get; set; }

public string Country { get; set; }

}

public class Order

{

public int OrderID { get; set; }

public DateTime OrderDate { get; set; }

public string ShipCity { get; set; }

public string ShipCountry { get; set; }

}

public class Product

{

public int ProductID { get; set; }

public string ProductName { get; set; }

public int UnitsInstock { get; set; }

public decimal UnitsPrice { get; set; }

public int CategoryID { get; set; }

}

Sonrakı addım olaraq DAL proyekti daxilində iki ədəd qovluq (Abstract və Concrete) yaradılır. “Abstract” qovluğu daxilində “interface” “Concrete” qovluğu daxilində isə tətbiq ediləcək framework-ə uyğun qovluq yaradılaraq onun daxilində kodlar yazılır və İnterfacedən miras verilir. Nəzərə almaq lazımdır ki məhz hər hansı bağlanma üsulu dəyişdirilən zaman “Concrete” qovluğuna ona uyğun qovluq əlavə edilməli və yeni kodlar yazılaraq İnterfacedən miras verilməlidir. Belə olan təqdirdə sistemdə dəyişikliklər etmədən qısa zamanda inteqrə prosesi icra edilmiş olur.

namespace Northwind.DataAccessLayer.Abstract// Abstract qovluğu daxilində interface yaradılır

{

public interface IProductDal

{

List<Product> GetAll(); // hamisini getir

Product Get(int productId); // birini getir

void Add(Product product); // elave et

void Delete(int productId); // birini sil

void Update(Product product); // yenile

}

}

“Concrete” qovluğuna daxilində aşağıdaki iki class yaradılır. Cədvələ uyğun kodları əks etdirəcək class interfacedən miras alaraq ona uyğun yuxarıdakı 5 əmri icra etməlidir. Bu səbəbdən bizə Entity Framework kodlarından istifadə etmək üçün “Context” sinfi yaradılmalı və Entity Framework -ün “DbContext” sinifindən miras almalıdır.

namespace Northwind.DataAccessLayer.Concrete.Entity\_Framework

{

public class NorthwindContext:DbContext

{

public DbSet<Product> Products { get; set; }

}

}

namespace Northwind.DataAccessLayer.Conrete.Entity\_Framework

{

public class EfProductDal : IProductDal

{

NorthwindContext \_context = new NorthwindContext();

// E.F. kodlarının tətbiqi üçün obj yaradılır.

public void Add(Product product)

{

\_context.Products.Add(product);

\_context.SaveChanges();

}

public void Delete(int productId)

{

\_context.Products.Remove(\_context.Products.FirstOrDefault

(o => o.ProductID == productId));

\_context.SaveChanges();

}

public Product Get(int productId)

{

//return \_context.Products.FirstOrDefault(o => o.ProductID == productId);

// ekvivalentidir

return \_context.Products.Where(o => o.ProductID == productId).FirstOrDefault();

}

public List<Product> GetAll()

{

return \_context.Products.ToList();

}

public void Update(Product product)

{

Product update = \_context.Products.FirstOrDefault

(o => o.ProductID == product.ProductID);

update.ProductName = product.ProductName;

update.CategoryID = product.CategoryID;

update.UnitsInstock = product.UnitsInstock;

update.UnitsPrice = product.UnitsPrice;

\_context.SaveChanges();

}

}

}

Sadalanan proseslər icra olunduqdan sonra BLL hissəsinə keçid edilir. Burada da iki sinif (Abstract və Concrete) yaradılır. *(Abstract içində BASE sinifləri saxlayır. Bu siniflər digər siniflərə miras vermək üçün istifadə edilir. Concrete qovluğu isə Abstract qovluğundan miras alan və kodların əsasının saxlanıldığı sinifləri özündə saxlayır.)*

#### Qeyd: Əgər Sistemin yazıldığı dil ilə User eyni dili istifadə edərsə bu zaman user-in service üzərindən yox bir başa BLL hissəsinə bağlanmağını təmin etməklə performansın yüksəldilməsinə səbəb ola bilərik. Bunun üçün BLL və Service üçün eyni mənanı kəsb edən interface yaradılmalıdır. Bunun üçün Solution daxilinə yeni “ClassLibrary” proyekti əlavə edilir.

namespace Northwind.Interfaces

{

public interface IProductService

{

List<Product> GetAll();

Product Get(int productId);

void Add(Product product);

void Delete(int productId);

void Update(Product product);

}

}

Interface əlavə edildikdən sonra BLL proyekti daxilində lazimi işləri yerinə yetirmək lazımdır. Yaradılmış olan “Concrete” qovluğu daxilində “ProductManager” sinfi yaradılır və bu sinif yeni yaradılmış “IProductService” interfeyzindən misal alması təmin edilir və implamentation prosesi icra edilir.

#### Qeyd: Buradan nəzərə almaq lazimdir ki, interface implamet edilən zaman "Product" sinfini gətirə bilmədiyindən onun using-i əlavə edilməlidir.

Diqqət edilməli digər bir məqam isə E.F kodlarının class daxilinə əlavə edilməsidir. Bunun üçün Nortwind daxilindəki “EfproductDal” sinifinin obyekti yaradılaraq bir başa obj.GetAll və digərləri gətirilə bilərdi. Lakin bu yalniş hall hesab edilir. Səbəb baza ile əlaqə kodları dəyişən zaman class daxilindəki kodların dəyişməsi tələbinin yaranmasıdır.

Bu məsələni nəzərə alaraq “EfproductDal” sinifinin obyektini yaratmaq əvəzinə həmin sinifin miras aldığı “IProductDal” interfacesini sinfin constroctor metodu daxilində parametrə mənimsətmək lazımdır. Bu proses icra edilməzsə metodların işləməsində problem yaranır. Proses icra olunduqdan sonra hər bir metodun daxilində ona uyğun metodu çağıraraq məsələnin icrası yerinə yetirilir. Burada eyni zamanda “Validation-Təsdiqləmə”, “Security-Müdafiyə və Təhlükəsizlik”, “Exception Handling-Xətaların yaxalanması” kimi məsələlər də yerinə yetirilir.

namespace Northwind.BusinessLogicLayer.Concrete

{

public class ProductManager : IProductService

{

private IProductDal \_productDal;

public ProductManager(IProductDal productDal)

{

\_productDal = productDal; // Constructor method

}

public void Add(Northwind.Entities.Product product)

{

// Validation - Tesdiqleme

// Security - Tehlukesizlik

// Exception Handling - Sehvlerin yaxalanmasi

\_productDal.Add(product);

}

public void Delete(int productId)

{

\_productDal.Delete(productId);

}

public Northwind.Entities.Product Get(int productId)

{

return \_productDal.Get(productId);

}

public List<Northwind.Entities.Product> GetAll()

{

return \_productDal.GetAll();

}

public void Update(Northwind.Entities.Product product)

{

\_productDal.Update(product);

}

}

}

Növbəti mərhələdə layihənin service hissənə lazımı kodlar əlavə edilir. İlkin olaraq service proyekti olan “Northwind.WcfLibrary” daxilində “Concrete” qovluğu daxilində “Product Service” sinifi yaradılır. Sinif BLL kimi İnterfaces proyektindən miras alır. Bundan sonra sazlama prosesi icra olunduqdan sonra kodların yazılması prosesi üçün BLL daxilində yaradılmış olan “ProductManager”-in obyekti yaradılır. Obyekt yaradılan zaman hansı dillə işləyəcəyini ona bildirmək üçün “ProductManager” daxilində istifadə olunaraq kodlar yazılan interface-nin miras verdiyi sinif təyin edilir. Bu prosesin icrasından sonra kodların çağırılması həyata keçirilir.

namespace Northwind.WcfLibrary.Concrete

{

public class ProductService : IProductService

{

// instance provider ile ezilecek

private ProductManager \_productManager = new ProductManager(new EfProductDal());

// E.F isleyeceyi bildirilir.

public void Add(Northwind.Entities.Product product)

{

\_productManager.Add(product);

}

public void Delete(int productId)

{

\_productManager.Delete(productId);

}

public Northwind.Entities.Product Get(int productId)

{

return \_productManager.Get(productId);

}

public List<Northwind.Entities.Product> GetAll()

{

return \_productManager.GetAll();

}

public void Update(Northwind.Entities.Product product)

{

\_productManager.Update(product);

}

}

}

#### Qeyd: Eyni zamanda Service hissənin icra oluna bilməsi üçün “Northwind.Interfaces” proyekti daxilindəki interface üzərinə “[ServiceContract]”, metodlar üzərinə isə “[OperationContract]” yazılmalıdır.

namespace Northwind.Interfaces

{

[ServiceContract]

public interface IProductService

{

[OperationContract]

List<Product> GetAll();

[OperationContract]

Product Get(int productId);

[OperationContract]

void Add(Product product);

[OperationContract]

void Delete(int productId);

[OperationContract]

void Update(Product product);

}

}

Bu prosesləri icra etdikdən sonra layihəni servisə çıxarmaq lazımdır. Bunun üçün yaradılmış olan “Northwind.WcfIISHost” proyektindən istifadə edilir. Proyekt daxilində yerləşən service.svc faylını silmək lazımdır. Buna səbəb servislərin sayı çoxaldıqca oradakı faylların sayı artmağa başlayır. Bunu nəzərə alaraq Web.config file daxilinə gələrək burada yerləşən “<serviceHostingEnvironment . . ./> ” formatında yazılmış kodun son hissənindəki “/>” işarə silinir və onun əvəzinə “>” işarəsi qoyulur. Bundan sonra “<serviceActivations>” sahəsi açılaraq onun içində servislər yaradılır.

<serviceHostingEnvironment aspNetCompatibilityEnabled="true"

multipleSiteBindingsEnabled="true" >

<serviceActivations>

<add

service="Northwind.WcfLibrary.Concrete.ProductService"

relativeAddress =" Product.svc"/>

</serviceActivations>

</serviceHostingEnvironment>

Növbəti addım olaraq “Northwind.WcfIISHost” proyekti “Set at Startap Project” etmədən öncə proyekt üzərində mausun sağ düyməsi basılaraq “add referance => Project => Northwind. WcfLibrary” ardıcıllığı ilə əlavə edilir və Servist host edilmiş olur. Növbəti addım olaraq User İnterface hissəsinin hazırlanaraq service-dən istifadə edə bilməsi təmin edilməlidir.

İlkin olaraq Controller hissəsinə “ProductController” əlavə edilir. Əlavə edilmiş Controller daxilində ilkin olaraq “GetAll” metodunu istifadə edərək məlumat gətirmək üçün BLL daxilində yerləşər “ProductManager” class-nın obyekti yaradılır *(Service hissəsində olan kimi)*. Yaradılmış obyekt əsasında metod çağırılaraq “Product” sinifinə uyğun formada list halında saxlanılır. Saxlanılmış list obyekt View daxilində ekranda əks edilir.

namespace Northwind.MvcWebUI.Controllers

{

public class ProductController : Controller

{

// GET: Product

ProductManager productManager = new ProductManager(new EfProductDal());

public ActionResult Index()

{

List<Product> products = productManager.GetAll();

return View(products);

}

}

}

Kodun daha doğru versiyası iki səhifə sonra göstərilmişdir.

Məlumatın çəkilərək ekrana gəlməsi üçün ilk əvvə ActionResult-a uyğun View yaradılmalıdır. View daxilində C# kodalarından istifadə edərək məlumatlar ekrana çıxarılır. İlkin olaraq yaradılmış html səhifənin başında məlumatları görə bilmək üçün model – başqa sözlə using əlavə edilir. Sonra isə ona əsasən məlumatların ekrana əks olunması üçün foreach dövründən istifadə edilir.

@model System.Collections.Generic.List<Northwind.Entities.Product>

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

</head>

<body>

<div>

@foreach (var product in Model)

{

@product.ProductName

<br/>

}

</div>

</body>

</html>

Lakin məlumatların ekranda əks oluna bilməsi üçün digər əsas bir amil məlumatların haradan çəkiləcəyini proqrama bildirmək üçün connectionString-in təyin edilməsidir. Bunun üçün Web.Config faylı daxilində aşağıdakı kodlar yazılır.

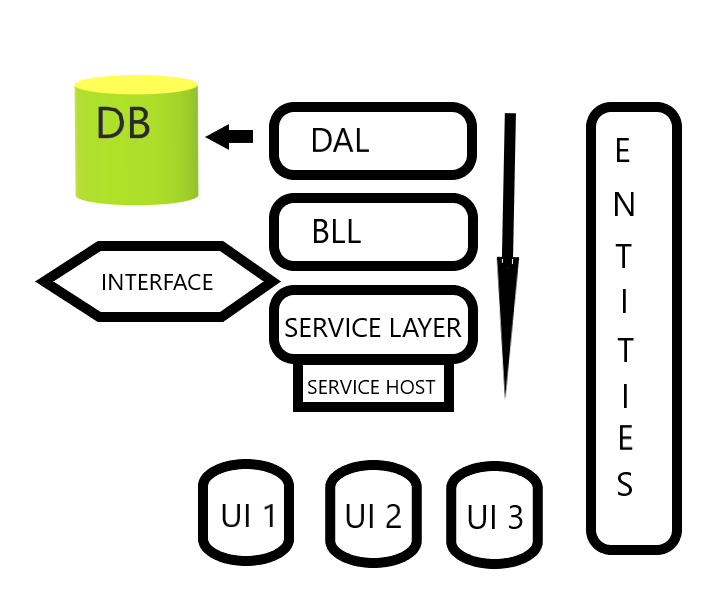
<connectionStrings>

<add

name="NorthwindContext" providerName="System.Data.SqlClient" connectionString=

"Data Source=DESKTOP-K25G1RK;Initial Catalog=Northwind;Integrated Security=True"/>

</connectionStrings>



DOĞRU KOD

namespace Northwind.MvcWebUI.Controllers

{

public class ProductController : Controller

{ //ProductManager \_productManager = new ProductManager(new EfProductDal());

private IProductService \_productService;

public ProductController(IProductService productService)

{

\_productService = productService;

}

public ActionResult Index()

{

List<Product> products = \_productService.GetAll();

return View(products);

}

}

}

Burada UI -i run edən zaman constructor ona verilən dəyərin nə olduğunu anlamadığı üçün error verəcəkdir. Çünki default olaraq constructor metod parametrsiz işləyir və parametr göndərilən zaman nə tip reaksiya verəcəyini təyin edə bilmir. Bu səbəbdən də metodun əzilməsindən yararlanaraq kodu işlək formaya gətirmək olar.

* İlkin olaraq MVC proyek daxilində “Infrastructure” qovluğu yaradılır.
* Qovluq daxilinə “NinjectControllerFactory” adlı sinif əlavə edilir.
* Yaradılmış sinif “DefaultControllerFactory” sinifindən miras alması təmin edilir.
* Sinif daxilində override prosesinin icra olunması üçün “override GetControllerInstance” yazaraq Enter düyməsini basmaq lazımdır.
* Prosesi icra edə bilmək üçün proyektə “Ninject” paketini yükləmək lazımdır.
* Paket yükləndikdən sonra “IKernel” Ninject paketi daxilindəki interface çağrılır.
* Sonra Constuructor metod daxilində aşağıdakı kodlar yazılır.
* Kodların yazılmasından sonra “override GetControllerInstance” metodu daxilində əgər gələn “Type controllerType” dəyər null olarsa, null, yox əgər null olmazsa, o zaman “\_ninjectKernel”-dan gələn dəyəri qaytar
* Son olaraq MVC proyekt daxilində yerləşən “Global.asax” faylı daxilində aşağıdakı kod yazılır.

namespace Northwind.MvcWebUI

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

Yeni kod == ControllerBuilder.Current.SetControllerFactory(new NinjectControllerFactory());

}

}

}

namespace Northwind.MvcWebUI.Infrastructure

{

public class NinjectControllerFactory:DefaultControllerFactory

{

private IKernel \_ninjectKernel;

public NinjectControllerFactory()

{

\_ninjectKernel = new StandardKernel();

AddBllBinding();

}

private void AddBllBinding()

{

Deyisen kod= \_ninjectKernel.Bind<IProductService>().To<ProductManager>()

.WithConstructorArgument("productDal", new EfProductDal());

}

protected override IController GetControllerInstance(RequestContext

requestContext, Type controllerType)

{

return controllerType == null ? null :

(IController)\_ninjectKernel.Get(controllerType);

}

}

}

Məlumatların ekrana çıxarılmasından sonra “css” class-ı və “\_LayoutNorthwind” Layoutundan istifadə edərək “inde” səhifəsi layoutdan məlumat alması ilə “masterpage” strukturunun qurulması təmin edilir. Bu zaman gələn məlumatlar render body hissəsində dəyişilir.

@model System.Collections.Generic.List<Northwind.Entities.Product>

@{

Layout = "~/Views/Shared/\_LayoutNorthwind.cshtml";

}

<!DOCTYPE html>

<div>

@foreach (var product in Model)

{

<div class="item">

<h5>

@product.ProductName

</h5>

<h5>

@product.UnitPrice

</h5>

</div>

<br/>

}

</div>

Front hissədə olan tənzimləmələri yerinə yetirdikdən sonra Controller hissəsində çəkilən məlumatların ekranda toplu formatda yox təyin edilmiş say əsasında gəlməsini təmin etmək üçün aşağıdakı dəyişimi etmək lazımdır.

## Çəkilən məlumatların ekranda səhifələr halında göstərilməsi

int pageSize = 5;

public ActionResult Index(int page=1)// default olaraq ilk "pageSize"-i getir

{

List<Product> products = \_productService.GetAll().

Skip((page-1)\*pageSize).Take(5).ToList();

return View(products);

}

// kodun menasi odur ki, necenci sehifeye getmek isteyirse gelen melumatlarin "(page-1)\*pageSize"-ni at "Take(5)" bunlari goster.

İlkin olaraq “int pageSize” dəyişəni təyin edilir *(Bu dəyişən səhifədə görüləcək məlumat sayını bildirir)*. Eyni zamanda “ActionResult” giriş kəmiyyəti olaraq “int page” verilir *(Default dəyər olaraq = 1 verilir. Bunun məqsədi səhifə ilk çağrılan zaman ilk 5 dəyərin əks etdirilməsini təmin etməkdir)*.

Lakin məlumatların gətirilməsi üçün bir həm gələn məlumat sayını, həmdə səhifə məlumatlarınıda view ilə ötürmək lazım gəlir. Bunun üçün class əsasında məlumatların ötürülməsini təmin edə bilərik.

* Bunun üçün səhifəyə ilk əvvəl məlumatların özü (“Products”) və onlar haqqında olan detalları (“PagingInfo”) bir class (“ProductViewModel”) daxilində göndərdik. (*Yaradılmış sinif Model qovluğu daxilində saxlanılır*)
* Sonra məlumatlar haqqında olan detalları ayrı bir class (“PagingInfo”) kimi yaradaraq daxilinə:
  + hər səhifədə görünəcək məlumat sayı (“ItemsPerPage”)
  + məlumatların ümumi sayı (“ItemsPerPage”)
  + göstərillən məlumatların hansı səhifədə olmasını (“CurrentPage”)

int pageSize = 5;

public ActionResult Index(int page=1)

{

List<Product> products = \_productService.GetAll();

return View(new ProductViewModel

{

Products = products.Skip((page-1)\*pageSize).Take(5).ToList(),

PagingInfo = new PagingInfo

{

ItemsPerPage = pageSize,

ItemsPerPage = products.Count,

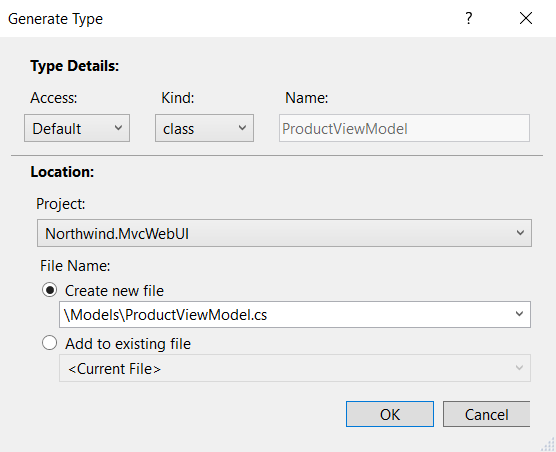
CurrentPage = page

}

});

}

Classların adlarını yazdıqdan sonra üzərilərinə gələrək “Generate Type” pəncərəsindən növünü class, lokasiyasını “Models” qovluğu daxilində seçərək yaradırıq.



namespace Northwind.MvcWebUI.Models

{

public class ProductViewModel

{

public List<Product> Products { get; set; }

public PagingInfo PagingInfo { get; set; }

}

}

namespace Northwind.MvcWebUI.Models

{

public class PagingInfo

{

public int ItemsPerPage { get; set; }

public int TotalItems { get; set; }

public int CurrentPage { get; set; }

}

}

#### Qeyd: Class yaradıldıqdan sonra View hissəsində model dəyişdirilməlidir.

#### Qeyd: Class-lar mütləq formada public edilməlidir.

@model Northwind.MvcWebUI.Models.ProductViewModel

@{

Layout = "~/Views/Shared/\_LayoutNorthwind.cshtml";

}

## Səhifələrin nömrələnməsinin təmin edilməsi.

İlk olaraq Proyekt üzərində yeni qovluq (“HtmlHelper”) və onun daxilinə class (“PagingHelpers”) əlavə edilir. Məqsəd Mvc proyekt üzərində developer üçün yardım məqsədi güdən və “@HTML. …” Kodu ilə yazılan sinifə yeni imkan əlavə etməkdir.

namespace Northwind.MvcWebUI.HtmlHelpers

{

public static class PagingHelpers

{

public static MvcHtmlString Pager (this HtmlHelper html, PagingInfo pagingInfo)

{

// <a href=""> 1 <a/>

var stringBuilder = new StringBuilder();

int totalPage = (int)Math.Ceiling(

(decimal)pagingInfo.TotalItems / pagingInfo.ItemsPerPage);

// sehife sayi tapilir

for (int i = 1; i <= totalPage; i++) // say qeder donur

{

var tagBuilder = new TagBuilder("a"); // a tag yarat

tagBuilder.MergeAttribute("href", String.Format("Index?page={0}", i)); // href linki ver

tagBuilder.InnerHtml = i.ToString(); // araya gelen deyer yazilsin

if (pagingInfo.CurrentPage == i)

{

tagBuilder.AddCssClass("selected");

}

stringBuilder.Append(tagBuilder); // melumatlari bir yerde yaxala

}

return MvcHtmlString.Create(stringBuilder.ToString());

}

}

}

Class daxilində uyğun kod yazıldıqdan sonra onları istifadə etmək üçün “NameSpace” kopyalanır və View daxilində yerləşən “Web.config” faylı daxilində add edilir. Bu prosesden sonra kodu istifadə edə bilərik.

<add namespace="Northwind.MvcWebUI.HtmlHelpers" />

. . .

<div>

@foreach (var product in Model.Products)

{

<div class="info">

<h3>

@product.ProductName

</h3>

<h4>

@product.UnitPrice

</h4>

</div>

<br />

}

<div class="pager">

@Html.Pager(Model.PagingInfo)

</div>

</div>

## Sol paneldə “Category” cədvəlini listləmək və category əsasında səhifələri nömrələmək

Addımlar:

* Entities proyektində class əlavə edilir.
* DAL proyektində “Abstract” cədvəlində interface yaradırıq.

namespace Northwind.DataAccessLayer.Abstract

{

public interface ICategoryDal

{

List<Category> GetAll();

}

}

* “NorthwindContext” sinifi daxilidə “Category” sinifinə uyğun properti yaradılır.

public DbSet<Category> Categories { get; set; }

* “Concrete” qovluğu daxilindəki “E.F.” qovluğuna yeni class əlavə edilir.

namespace Northwind.DataAccessLayer.Concrete.Entity\_Framework

{

public class EfCategoryDal : ICategoryDal

{

NorthwindContext \_context = new NorthwindContext();

public List<Category> GetAll()

{

return \_context.Categories.ToList();

}

}

}

* “Northwind.İnterfaces” Proyektindən servisi əlavə edilir.

namespace Northwind.Interfaces

{

[ServiceContract]

public interface ICategoryService

{

[OperationContract]

List<Category> GetAll();

}

}

* “Northwind.BLL” Proyektində “Concrete” qovluğu daxilində yeni sinif əlavə edilir.

namespace Northwind.BusinessLogicLayer.Concrete

{

public class CategoryManager : ICategoryService

{

private ICategoryDal \_categoryDal;

public CategoryManager(ICategoryDal categoryDal)

{

\_categoryDal = categoryDal;

}

public List<Category> GetAll()

{

return \_categoryDal.GetAll();

}

}

}

* Tələbə uyğun olaraq “Northwind.WcfSevice” Proyektindən “Concrete” qovluğu daxilində yeni sinif əlavə edilməli idi. Lakin bu nümunədə bu hissəni boş keçməklə tətbiq ediləcəkdir. Məntiqi nəticə olaraq bu zaman birbaşa UI hissəsinə gəlmək kifayət edir.
* Controller hissəsində uyğun adla controller yaradılır.

namespace Northwind.MvcWebUI.Controllers

{

public class CategoryController : Controller

{

private ICategoryService \_categoryService;

public CategoryController(ICategoryService categoryService)

{

\_categoryService = categoryService;

}

public ActionResult List()

{

return PartialView(\_categoryService.GetAll());

}

}

}

#### Qeyd: Məlumatları list formatında gətirmək üçün iki yol vardır. Birinci yol pəncərə layout üzərində olduğuna görə məlumatı orada çekmək. Digər bir üsul isə “PartialView”-dan istifadə etməkdir.

@model System.Collections.Generic.List<Northwind.Entities.Category>

@foreach (var category in Model)

{

@Html.RouteLink(category.CategoryName, new

{

controller = "Product",

action = "Index"

} )

}

* Layout hissəsində məlumatların listlənməsi üçün Sol blok hissəsinə aşağıdakı kodlar yazılır.

<div class="sidebar">

<h2>Categories</h2>

<ul>

<li>

@{ Html.RenderAction("List", "Category");}

</li>

</ul>

</div>

#### Qeyd: Bu kodları yazmağımıza baxmayaraq controller daxilindəki constructor metod giriş kəmiyyətinin nə olduğunu almağı üçün “NinjectCottroller”-dən onun məlumatlarını daxil etmək lazım gəlir.

. . .

private void AddBllBinding()

{

\_ninjectKernel.Bind<ICategoryService>().To<CategoryManager>().

WithConstructorArgument("categoryDal", new EfCategoryDal());

}

. . .

#### UNUTMA: Cateqoryaların listlənmiş formatda görülməsi üçün “Category”controller daxilindəki “List” view əvəzinə “Product” controlleri daxilindəki “index” səhifəsini işlətmək lazımdır.

Category məlumatlarının ekranda əks edilməsi prosesi tamamlandıqdan sonra, category əsasında məlumatların gəlməsini təmin etmək lazımdır. Bu səbəbdən də “Product” controller daxilində index “actionresult”-na bəzi dəyişikliklər etmək lazım gələcəkdir.

. . .

int pageSize = 5;

public ActionResult Index(int page=1, int category = 0)

{

List<Product> products = \_productService.GetAll().Where

(o => o.CategoryID == category|| category == 0).ToList();

return View(new ProductViewModel

{

Products = products.Skip((page-1)\*pageSize).Take(5).ToList(),

PagingInfo = new PagingInfo

{

ItemsPerPage = pageSize,

ItemsPerPage = products.Count,

CurrentPage = page

}

});

}

Controller daxilində qırmızı hissədə göstərilmiş dəyişikliyi edərək “category=0” olan zaman bütün məlumatlar əks halda isə həmin category-ə uyğun məlumatlar gətiriləcəkdir.

Category-nin üzərinə toxunan zaman məlumatlara uyğun səhifələrin listlənməsi üçün “Category => List” hissəsinə gələrək aşağıdakı dəyişikliyi etmək lazım gələcəkdir.

@model System.Collections.Generic.List<Northwind.Entities.Category>

<li>

@Html.RouteLink("AllList",new

{

controller = "Product", //Bütün məlumatları səhifədə göstərmək üçün.

action = "index",

category = 0

},

new { @class = ViewBag.CurrentCategory == 0 ?"Selected":null}) )

</li>

@foreach (var category in Model)

{

<li>

@Html.RouteLink(category.CategoryName, new

{

category = category.CategoryID, //İd esasında məlumatları səhifədə göstərmək üçün.

controller = "Product",

action = "Index"

},

new { @class = category.CategoryID == ViewBag.CurrentCategory?"Selected":null})

</li>

}

Mavi çərçivə içərisindəki kod: Əgər categoryİd controllerdən gələn dəyər ilə bərabərdirsə, o zaman CSS formatı ver. Əks halda format vermə.

Controller hissəsindən məlumat göndərmək üçün “Category” controllerində aşağıdakı kod dəyişikliyi edilməlidir.

. . .

public ActionResult List(int category = 0)

{

ViewBag.CurrentCategory = category;

return PartialView(\_categoryService.GetAll());

}

#### Qeyd: “ViewBag” ifadəsi controller hissəsindən view hissəsinə məlumat göndərilməsi üçün istifadə edilir.

Məlumatların category əsasında gəlməsinə baxmayaraq səhifələmə və keçid arasında link problemi olduğunu nəzərə alaraq aşağıdakı yerlədə bəzi dəyişikliklər edərək bu problemi aradan qaldırmaq olar.

* Models qocluğu daxilindəki “Pagingİnfo” sinifi daxilinə “CurrentCategory” adı altında bir ədəd property əlavə edilir.

public class PagingInfo

{

public int ItemsPerPage { get; set; }

public int TotalItems { get; set; }

public int CurrentPage { get; set; }

public int CurrentCategory { get; set; }

}

* Əlavə edilmiş property Controller daxilində də yazılaraq dəyər mənimsədilir.

. . .

return View(new ProductViewModel

{

Products = products.Skip((page-1)\*pageSize).Take(5).ToList(),

PagingInfo = new PagingInfo

{

ItemsPerPage = pageSize,

ItemsPerPage = products.Count,

CurrentPage = page,

CurrentCategory = category

}

});

* Mənimsədilmiş dəyər əsasında “HtmlHelpers” qovluğu daxilindəki “PagingHelpers” sinfində link üzərinə category hissəsini əlavə etdikdən və əvvəlində controller ilə daxilindəki actionresoultun adlarını əlavə etdikdən sonra səhifələr arası keçid problemi həll olmuş olur.

#### Qeyd: Burada nəzərə alınmalı bir amil olaraq qeyd etmək olar ki, Href linki yaradan zaman aşağıfakı halları nəzərə almaq lazımdır:

#### "Index?page={0}" – category arası keçidində “controller” adı silinir məlumat gəlmir.

#### "Product/Index?page={0}"- yuxarıdakı hadisə baş verir.

#### "/Prodoct/Index?page={0}"-Controller ve sehife adi daim olaraq qalır. (**DOĞRU YAZILIŞ**)

. . .

for (int i = 1; i <= totalPage; i++) // say qeder donur

{

var tagBuilder = new TagBuilder("a"); // a tag yarat

tagBuilder.MergeAttribute("href", String.Format

("/Product/Index?page={0}&category={1}", i, pagingInfo.CurrentCategory));

tagBuilder.InnerHtml = i.ToString(); // araya gelen deyer yazilsin

if (pagingInfo.CurrentPage == i)

{

tagBuilder.AddCssClass("selected");

}

stringBuilder.Append(tagBuilder); // melumatlari bir yerde yaxala

}

. . .

## Məlumatların yanında Add buttonu yaratmaq və səbətə əlavəetmə prosesi

Məlumatların category əsasında gələrək sıralanması prosesini bitirdikdən sonra həm ümumi siyahıda, həm də category-ə görə sıralanan zaman sağ tərəflərində add buttonunun əlavə edilməsi prosesini reallaşdıran kodları yazaq.

Butonların əlavə edilməsi üçün ilkin olaraq hansı məlumata clik edildiyini bilmək lazımdır ki, onun əsasında səbətə əlavə edilmə məsələsini təmin etmək mümkün olsun. Bunun üçün hər bir məlumat üçün “form” yaradaraq bu məsələni həll etmək olar. Bu zaman Index view daxilində aşağıdakı kodlarıı əlavə etmək lazımdır.

<!DOCTYPE html>

<div>

@foreach (var product in Model.Products)

{

. . .

using (Html.BeginForm("AddToCard", "Cart")) // Cart - controller, AddToCart - action

{

@Html.Hidden("productId",product.ProductID)

// Idni yaxalamaq ucun ist edilir.

<input type="submit" value="Add to cart"/> // button

}

<br />

}

. . .

@Html.HiddenFor(o => product.ProductID)

hiddenden ferqi gelen Id-ni oz teyin etdiyi ad altinda saxlamasidir/ Bu ise cari anda isi tam eks tedirmeye biler\*/

Index view daxilində dövrün içərisinə form əlavə edildikdən sonra burada hansı dəyərin olduğunu anlamaq üçün onların “ID”-lərini yaxalamaq olar. Bunun səbəbbi ilə hmtl kodunun yaratmış olduğu “HİDDEN” metodundan istifadə edə bilərik. *(Hidden ilə eyni funksiyanı icra edən “hiddenfor” funksiyasından da istifadə etmək mümkündür. Lakin nəzərə almaq lazımdır ki, hiddenfor for funksiyası məlumatı yaxalayan zaman özü tərəfindən dəyişəni adlandırır. Bu hal isə cari vəziyyətdə o qədər də istafədə imkanı yaratmır).*

Növbəti addımda Using daxilində yazılmış olan controller adı və Actionunun əlavə edilməsini lazımdır. (*Səpət mexanizması action daxilində yaradılacaq*) Uyğun olaraq bu proses üçün “Cart-səbət” adında bir class yaradılır.

#### Qeyd: Sebet yaradılan zaman nəzərə almaq lazımdır ki, bu səbətdən eyni zamanda müxtəlif platformalardan girişlər olacağı üçün onun MVC-nin Models qovluğu əvəzinə “Entities” proyekti daxilində yaradılması daha doğru olacaqdır.

namespace Northwind.Entities

{

public class CartLine

{

public Product Product { get; set; }

public int Quantity { get; set; }

}

Məlumatları yaxalamaq üçün CartLine adında class yaradılaraq içərisində “Product” tipində obyekt və sayı yaxalamaq üçün dəyər yaradılır.

public class Cart

{

List<CartLine> lines = new List<CartLine>();

Gələn hər bir CartLine tipindəki məlumatlar List formatında yaxalanaraq üzərində əməliyyatlar icra edilə bilər.

public void AddToCart(Product product,int quantity)

{

CartLine cartLine = lines.FirstOrDefault

(o => o.Product.ProductID == product.ProductID);

if (cartLine == null)

{

lines.Add(new CartLine

{

Product = product,

Quantity = quantity

});

}

else

{

cartLine.Quantity += quantity;

}

}

İlkin olaraq səbətə əlavə etmə prosesi zamanı Product tipində məhsul və onun sayı əlavə edilir. Bu zaman metod daxilində əvvəlcə həmin məhsulun öncədən əlavə edilib edilməməsi, başqa sözlə deyilərsə varlığı yoxlanılır. Əgər məhsul öncədən səbətə əlavə edilməyibsə bu zaman giriş dəyəri kimi verilən kəmiyyətlər “CartLine” classına əlavə edilir. Yox əgər məhsul əvvəlcədən mövcuddursa, bu zaman öncəki sayın əvəzinə yenisi əlavə edilir.

public void RemoveFromCart(Product product)

{

lines.RemoveAll(o => o.Product.ProductID == product.ProductID);

}

Səbətə əlavə edilmiş məhsulun səbətdən silinməsi tələbi yarana bildiyindən buradan seçilmiş məhsulun “ID” nə əsasən onun haqqında olan məlumatlar səbətdən silinə bilər.

public decimal Total

{

get

{

return lines.Sum(o => o.Product.UnitPrice \* o.Quantity);

}

}

Eyni zamanda səbətə məhsul əlavə edildikcə səbətdə olan məhsulların ümumi dəyəridə ekranda əks etdirilməlidir.

public void clear()

{

lines.Clear();

}

Son olaraq müştəri səbət daxilindəki bütün məhsulların silinməsi tələbi irəli sürə bilər. Bunun üçün Clear metodu faydalı olacaqdır.

public List<CartLine> Lines

{

get { return lines; }

}

Yekunda isə bütün məlumatlar list formatında listələnir.

}

}

Clasın yaradılma prosesi tamamlandıqdan sonra controller daxilində uyğun məlumatların doldurulması prosesini icra etmək olar. Məlumatların icra edilməsi üçün ilkin olaraq controller daxilinə göndərilən ID əsasında bazadan həmin ID uyğun məhsul haqqında məlumatın çəkilməsi lazımdır.

#### Session Nədir? İstifadəçi veb səhifəni açdığıdı andan etibarən başlayan və brauzer bağlanana qədər olan zaman intervalına “session” deyilir. Data saxlamaq üçün istifadə edilir. (*Cari məsələdə istifadə user girişi etmədən səbətə məhsul əlavə edə bilməsi üçün istifadə olunur*)

namespace Northwind.MvcWebUI.Controllers

{

public class CartController : Controller

{

private IProductService \_productService;

public CartController(IProductService productService)

{

\_productService = productService;

}

public ActionResult Index()

{

var cart = (Cart)Session["cart"];

return View(cart);

}

public ActionResult AddToCart(int productId)

{

Product product = \_productService.Get(productId);

var cart = (Cart)Session["cart"]; // obyekt oldugu ucun Cart-a cast edilir.

if (cart == null)

{

cart = new Cart();

Session["cart"] = cart;

}

cart.AddToCart(product, 1);

return RedirectToAction("Index", cart);

}

}

}

Button əlavə edən zaman göndərdiyimiz “ID” controllerin giriş kəmiyyəti olaraq daxil olur. İlkin olaraq “ID” əsasında məlumatlar çəkilir. Sonra “Session” obyektinin boş olub olmaması yoxlanılır (Boş olarsa Yeni Cart tipi mənimsədilir.) və metod ilə əlavə edilir. Məlumatlar isə “İndex” adındakı view daxilində əks edilir və parametr olaraq view üçün “cart” göndərilir.

***Yaradılmış view kodları:***

@model Northwind.Entities.Cart

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_LayoutNorthwind.cshtml";

}

<h2>Product in the cart</h2>

<table>

<tr>

<td>Product Name</td>

<td>Unit Price</td>

<td>Quantity</td>

<td>Tutar</td>

</tr>

@foreach (var line in Model.Lines)

{

<tr>

<td>@line.Product.ProductName</td>

<td>@line.Product.UnitPrice</td>

<td>@line.Quantity</td>

<td>@((line.Quantity \* line.Product.UnitPrice).ToString("c"))</td>

<td>

@using (Html.BeginForm("RemoveFromCart", "Cart"))

{

@Html.Hidden("productId", line.Product.ProductID)

<input type="submit" value="Remove From Cart"/>

}

</td>

</tr>

}

</table>

ToString("c") pul vahidi tipində göstərməni təmin edir.

Məlumatların əlavə edilməsindən sonra onların çıxarılması tələbi yaranarsa bu prosesin icrası üçün view daxilində button əlavə edilir. Əlavə edilmiş button digərində olduğu kimi “ID”-ni saxlayaraq Cart controlleri daxilindəki action-a göndərərək onun işləməsini təmin edir.

. . .

public ActionResult RemoveFromCart(int productId)

{

Product product = \_productService.Get(productId);

var cart = (Cart)Session["cart"];

//if (cart.Lines.Count == 0)

//{

// ModelState.AddModelError("", "Sebetde mehsul yoxdur");

//}

//else

//{

cart.RemoveFromCart(product);

//}

return RedirectToAction("Index", cart);

}

. . .

## Sifariş et düyməsi

Məhsulların səbətə əlavə edilməsindən sonra onların sipariş edilməsi məsələsi meydana çıxır. İlkin olaraq bu tələbi reallaşdıra bilmək üçün “Cart” condrolleri daxilində istifadə olunan “Index” view-a növbəti səhifəyə keçid üçün bir ədəd “istinad” yerləşdirilir.

<p align="right" class="btn-primary">

@Html.ActionLink("Checkout Now", "Checkout");

Kodun ekvivalenti => @\*<a href="/Cart/Checkout">Checkout Now</a>\*@

</p>

Növbəti addımda istinad verilən səhifə üçün “Cart” controlleri daxilində “Checkout” adında “action” yaradılır. Yaradılmış action geriyə şifarişçini məlumatlarının əldə edilməsi üçün lazım olan view əks etdirir. Bu səbəbdən də view yaratmazdan öncə məluatarı özündə saxlayacaq class yaradılmalıdır. Class yaradılan zaman onun global xarakter daşıdığı və müxtəlif platformlardan müraciət qəbul edə biləcəyini nəzərə alaraq “Entities” proyekti daxilində yaradılır.

Nəzərə almaq lazımdır ki, məlumatların əlavə edilməsindən sonra onların button vasitəsi ilə təsdiq prosesi reallaşdırılacaq və bunun ilə məlumatların yüklənməsi prosesi icra olunacaq. Nəticə etibarı ilə controller daxilində yaradılmış olan metodun eyni zamanda “Post” metodu da yaradılmalıdır ki, button klik edən zaman məlumatların əlavə edilməsi prosesi həyata keçirilə bilsin.

[HttpGet]

public ActionResult Checkout()

{

return View(new ShippingDetails());

}

[HttpPost]

public ActionResult Checkout(ShippingDetails shippingDetails)

{

if (ModelState.IsValid) // Model dogru olarsa

{

// Manager-dan DB-e melumat elavesi kodlari burada olacaq

return View("Complated");

}

else

{

return View(shippingDetails);

}

}

“Post” metodunun işləməmişdən oncə get metodu ilə məlumatları “ShippingDetails” class-ı daxilində saxladığımızdan Post metodu üçün giriş kəmiyyəti olaraq bu classdan istifadə edilir. Lakin burada ilk öncə Class daxilində tətbiq edilmiş olan “**validation**”-lar əsasında məlumatların təsdiq prosesi yerinə yetirilərsə, başqa sözlə desək “(ModelState.IsValid) // Model dogru olarsa” kodların əlavə edilməsi prosesi icra olunacaqdır. Əks halda error mesajları ilə eyni səhifə yenidən yüklənəcəkdir.

***Məlumatları əks etdirən class:***

namespace Northwind.Entities

{

public class ShippingDetails

{

[Required(ErrorMessage ="Ad girmək vacibdir")] boş buraxıla bilməz

[Display(Name = "Ad Soyad")] ekranda əks olunacaq ad

public string Name { get; set; }

[Required()]

[MinLength(10)] minimum 10 simvol

[MaxLength(50)] maksimum 50 simvol

public string Adress1 { get; set; }

[MaxLength(50)]

public string Adress2 { get; set; }

[MaxLength(50)]

public string Adress3 { get; set; }

[Required(ErrorMessage = "Şəhər girmək vacibdir")]

public string City { get; set; }

public string Country { get; set; }

public bool IsGift { get; set; }

}

}

“Checkout” üçün view yaradılması prosesində bir başa modeli nəzərə alaraq class daxilindəki property-lərə uyğun atributları “EditorForModel” metodundan istifadə edərək yaratmaq olar.

***Checkout View kodları:***

@model Northwind.Entities.ShippingDetails

@{

ViewBag.Title = "Checkout";

Layout = "~/Views/Shared/\_LayoutNorthwind.cshtml";

}

<h2>Checkout Now</h2>

@using (Html.BeginForm())

{

@Html.EditorForModel()

<input type="submit" value="Complate Chechout"/>

}

***Complate View kodları:***

@{

ViewBag.Title = "Complated";

Layout = "~/Views/Shared/\_LayoutNorthwind.cshtml";

}

<he>Succses</he>

## Header hissəsindəki səbət haqqında məlumatlar

Sözü gedən bu sahə hedar hissəsinin sağ künçündə yerləşdilir və səbətə məlumat əlave etdikcə həm məhsulun sayı, həm də ümumi məbləği göstərir.

İlkin olaraq header hissəsində bu sahəni yaratmaq üçüb “Layouth” hissəsində kod əlavəsi edilir.

<div class="fas">

<div class="header" height="200px">

<div class="fas" align="right">

@{Html.RenderAction("CartSummary", "Cart"); }

@\*her hansi bir "action"-i ise salmaq ucun\*@

</div>

<div class="fas" align="left">

Northwind DB

</div>

</div>

</div>

Layout hissəsində kod əlavəsi edildikdən sonra Kodda görüldüyü kimi “Cart” cotrolleri daxilində “CartSummary” adında action yaradılmalı və məlumatlar bu action əsasında ekranda əks etdirilməlidir. Məlumatların əks etdirilməsi üçün həmin controller daxilində istifadə edilmiş “session” məlumatlarından istifadə etmək olar.

public PartialViewResult CartSummary()

{

var cart = (Cart)Session["cart"];

if (cart==null)

{

cart = new Cart();

}

return PartialView(cart);

}

Action əsasında View yaradan zaman “PartialView” kimi yaradılır. (*Əgər proses dəfələrlə təkrar olunacaqsa, bu zaman partialview istifadə edilir. Misal olaraq veb səhifə daxilində istifadə edilən şəkillər toplusu. Bu toplunu hər səhifə üçün təkrar-təkrar eyni qalereya yaratmaq lazımsız və vaxt itkisidir. Bu zaman köməyə “partialview” gəlir. “Partialview” təklikdə heç bir prosesi icra etmir, yalnız işlədiyi səhifə daxilində fəaliyyət göstərir.*)

***Partialview kodları:***

@model Northwind.Entities.Cart

<div align="right">

<span class="caption" align="right">

<b>Your Cart "(@Model.Lines.Sum(x => x.Quantity))"</b> // sayın cəmlənməsi

@Model.Total.ToString("c") // məbləgin toplamı

</span>

@Html.ActionLink("Checkout", "Index", "Cart"); // istinad linki

</div>

## ModelBinding anlayışı və nümumə.

ModelBinding: View-dan post ilə göndərilən dəyərlərin Controller tərəfində yaxalananaq Action-ların giriş parametrlərinə çevrilməsi prosesidir. Əgər Controller hissəsindəki action metod giriş parametri alırsa və bu parametr View-dan post ilə göndərilərsə, bu zaman ilk olaraq Model Binder-ə düşər. Model Binder isə öz nüvbəsində kənardan gələn bu dəyəri giriş kəmiyyətinə bərabərləşdirər. Beləliklə View-dan gələn dəyərlə controller tərəfində istifadə edilər. Bunun üçün mövcud Model Binding mexanizmindən və ya developer tərəfindən yazılan model binding-dən istifadə edilə bilər.

Mövcud proyekt üzərində “Session” yaxalayaraq səbət prosesi icra edildi. Lakin hər dəfəsində eyni kod təkrar edilirdi. Bu səbəblə Model binding daxilində eyni kod yazılır və controller hissəsində kodlar silinərək sadecə giriş parametri verilir.

* MVC proyektinə yeni qovluq əlavə edilir “ModelBinders”.
* Qovluq daxilində class yaradılır “CartModelBinder.cs”.
* Yaradılan class-ın “IModelBinder”interface-dən miras alması təmin edilir. (using System.Web.Mvc;)
* Intarface əsasında implamentation prosesi icra olunur və metod daxilində uyğun kodlar yazılır.
* Kodların işləyə bilməsi üçün “Global.asax” daxilində qeydiyyat edilməsi lazımdır.
* Cotroller daxilində təkrarlanan “session” kodları əvəzlənir və metodlara giriş parametri olaraq “Cart” sinfinin obyekti verilir.

***ModelBinding class kodları:***

namespace Northwind.MvcWebUI.ModelBinders

{

public class CartModelBinder : IModelBinder

{

public object BindModel(ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

var cart = (Cart)controllerContext.HttpContext.Session["cart"];

if (cart == null)

{

cart = new Cart();

controllerContext.HttpContext.Session["cart"] = cart;

}

return cart;

}

// controllerContext - cari cotrollerdə "HttpContext" yaxalamaga komək edəcək bir mühitdir.

}

}

***Kodların Global.asax daxilində qeyd edilməsi:***

namespace Northwind.MvcWebUI

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

. . .

System.Web.Mvc.ModelBinders.Binders.Add(typeof(Cart), new CartModelBinder());

// Əgər cotroller daxilində giriş kəmiyyəti olaraq "Cart" tipi isdənilərsə, ona dəyər olaraq "CartModelBinder" bərabərləşdir.

}

}

}

***Cotrollerdə dəyişilmiş actionlar:***

namespace Northwind.MvcWebUI.Controllers

{

public class CartController : Controller

{

. . .

public ActionResult Index(Cart cart)

{

return View(cart);

}

public ActionResult AddToCart(Cart cart, int productId)

{

Product product = \_productService.Get(productId);

cart.AddToCart(product, 1);

return RedirectToAction("Index", cart);

}

public ActionResult RemoveFromCart(Cart cart, int productId)

{

Product product = \_productService.Get(productId);

cart.RemoveFromCart(product);

return RedirectToAction("Index", cart);

}

public PartialViewResult CartSummary(Cart cart)

{

return PartialView(cart);

}

}

}

## AdminPage əlavə edilməsi

İlkin olaraq Controller hissəsinə “Admin” adında yeni cotroller əlavə edilir. Yaradılmış kontrollerin service xidmətini icra edən “IProductService” interface əsasında məlumat mübadiləsini təmin etmək üçün obyekti və constructor metod yaradılır. Sonra yaradılmış obyektdən istifadə edərək məlumatları ekranda göstərmək üçün GetAll metodu index action-nun view-na göndərilir. Göndərilən məlumatarı saxlamaq üçün Index və həmin səhifəni master page etməkə imkan verən AdminLayout yaradılır.

@model System.Collections.Generic.List<Northwind.Entities.Product>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Products</h2>

@Html.ActionLink("Create New", "CreateNew")

<table>

<tr>

<td>Product Id</td>

<td>Product Name</td>

<td>Unit Price</td>

</tr>

@foreach (var item in Model)

{

<tr>

<td>@item.ProductID</td>

<td>@item.ProductName</td>

<td>@item.UnitPrice</td>

<td>@Html.ActionLink("Edit", "Edit", new { productId = item.ProductID })</td>

<td>@Html.ActionLink("Delete", "Delete", new { productId = item.ProductID })</td>

</tr>

}

</table>

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Admin Panel</title>

</head>

<body>

<div>

@RenderBody()

</div>

</body>

</html>

Index səhifəsi daxilində yaradılmış olan “Edit”, “Create” və “Delete” linklərini işlək hala gətirmək lazımdır. İlkin olaraq Create hissəsinin yaradılması üçün Controller daxilində “CreateNew” adında action yaradılır. Bu action daxilində “Product”-ı göndərmək lazımdır ki, view daxilində class atributlarına uyğun olan məlumatları yaxalayaraq sonradan onları post metodu vasitəsi ilə insert etməyə imkan olsun. Məlumatları sürətli formada hazırlamaq üçün “beginfor” daxilində “editorForModel” istifadə edərək avtomatik olaraq yaradılmadan istifadə etmək olar.

[HttpGet]

public ActionResult CreateNew()

{

return View(new Product());

}

[HttpPost]

public ActionResult CreateNew(Product product)

{

if (ModelState.IsValid)

{

productService.Add(product);

return RedirectToAction("Index");

}

return View(product);

}

@model Northwind.Entities.Product

@{

ViewBag.Title = "CreateNew";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Create New</h2>

@using (Html.BeginForm())

{

Html.EditorForModel()

<input type="submit" value="Save">

}

Eyni zamanda həm get, həm də post metodunun eynisi “Edit” üçün olacağından eyni kodları kopyalamaq olar. Lakin controller daxilində Get metodunu yazan zaman “Create”-dən fərqli olaraq burada view daxilində “Product” əvəzinə cari “id” əsasında olan “Product product” göndərmək lazımdır. Edit üçün yaradılan view kodları üçün “Create” view-nun kodlarından istifadə etmək olar.

[HttpGet]

public ActionResult Edit(int productid)

{

Product product = productService.Get(productid);

return View(product);

}

[HttpPost]

public ActionResult Edit(Product product)

{

if (ModelState.IsValid)

{

productService.Update(product);

return RedirectToAction("Index");

}

return View(product);

}

@model Northwind.Entities.Product

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.EditorForModel()

<input type="submit" value="Save">

}

Həm “Create”, həm də “Edit” edən zaman ProductId-nin ekranda gəlməməsini təmin etmək üçün “**System.Web.Mvc**”-ın “HiddenInput” atributundan istifadə etmək lazımdır. Bunun üçün Entities proyekti daxilindəki Product Classı daxilində “” kodunu yazmaq tələb olunur. Eyni zamanda “**System.ComponentModel.DataAnnotations**” atributlarından istifadə edərək qaxil edilən məlumatların hər birinin yazılması tələbi və digər vasitələrdən istifadə edə bilərik.

using System.Web.Mvc;

using System.ComponentModel.DataAnnotations;

namespace Northwind.Entities

{

public class Product

{

[HiddenInput(DisplayValue = false)]

public int ProductID { get; set; }

[Required]

public string ProductName { get; set; }

[Required]

public decimal UnitPrice { get; set; }

[Required]

public int CategoryID { get; set; }

}

}

## AdminPage səhifəsindəki imkanlara giriş məhdudiyyəti

URL anlayışından məlumatı olan istənilən şəxs admin səhifəyə daxil ola və məlumatları dəyişə bilər. Bunun qarşısını almaq və sadecə seçilmiş adamların həmin məlumatları dəyişiklik etməsinə icazə olmasını təmin etmək lazımdır. Bunun üçün MVC Framework daxilində olan “Authorize” imkanından istifadə edə bilərik.

[Authorize]

[HttpGet]

public ActionResult CreateNew()

{

return View(new Product());

}

Proyekti işlədikdən sonra admin səhifənə rahatlıqla giriş olmasına baxmayaraq digər imkanları istifadə etmək istənilən zaman error səhifəsi yüklənir. Burada diqqət edilməli bir digər məqam isə URL hissəsidir (<http://localhost:51562/Account/Login?ReturnUrl=%2fAdmin%2fCreateNew>). Bu URL-dən görünür ki, Account controlleri daxilində ki, Login səhifəsinə yönəlməyə çalışır. (*Bu hadisənin olması avtomatik olmazsa, bu zaman “web.config => configuration => System.web” daxilində* “authentication” tag-i yaradılır.)

<system.web>

<compilation debug="true" targetFramework="4.6.1" />

<authentication mode="Forms">

<forms loginUrl="~/Account/Login" timeout="2880"/>

</authentication>

<httpRuntime targetFramework="4.6.1" />

</system.web>

Buradan belə bir məna yaranır ki, əgər sistemə giriş etməyən bir user üzərində “Authorize” annotation-ı olan bir action-ı icra etmək istəyərsə, bu zaman həmin istifadəçini Admin controller-i daxilindəki login actionuna yönəlt- başqa sözlə desək sistemə girişini təmin et. Məhz istifadəçi sistemə giriş etdikdən sonra onun bu imkandan istifadə edib-edə bilməməsi təmin ediləcəkdir.

Yuxarıda qeyd edilmiş URL-nin işləməsini təmin etmək səbəbi ilə Controller qovluğuna “Account” controller əlavə edilir. Yaradılmış controller üçün ilk metod olaraq “login” action-ı yaradılır. Login Action-ı daxilinə user məlumatları göndərmək lazımdır. Bunun üçün Entities proyekti daxilində Yeni bir class yaradılır. Sonra həmin class əsasında məlumatlar üzərində proseslər təmin edilir.

namespace Northwind.Entities

{

public class User

{

[Required]

public string UserName { get; set; }

[Required] [DataType(DataType.Password)]

public string Password { get; set; }

}

}

Istifadəçinin giriş zamanı doğruluğunu iki formada yoxlamaq olar. Əsas hal burada baza üzərindən məlumatların yoxlanılmasıdır. Lakin Hard kod vasitəsi ilə yazılan məlumatların yoxlanış prosesidə ola bilər. Məlumatın harada yerləşməsindən asılı olmayaraq ilkin halda məlumatı yoxlamaq üçün bir metoddan istifadə etmək lazımdır. Proyekt daxilində əvvəlcə interface yaradılır. Yaradılmış interface həm BLL və service hissəsində metodların yazılması prosesi icra edilir. Eyni zamanda BLL daxilindəki method məlumatları DAL-ın Abstract qovluğu daxilindəki interface-dən, o isə öz nüvbəsində DAL-ın Concrete qovluğu daxilindəki uyğun class-dan məlumatları əldə edəcəkdir.

using Northwind.Entities;

namespace Northwind.Interfaces

{

public interface IAuthenticationManager

{

User Authenticate(User user);

}

}

namespace Northwind.BusinessLogicLayer.Concrete

{

public class AuthenticationManager:IAuthenticationService

{

private IAuthenticationDal \_authenticationDal;

public AuthanticationManager(IAuthenticationDal authenticationDal)

{

\_authenticationDal = authenticationDal;

}

public User Authenticate(User user)

{

return \_authenticationDal.Authenticate(user);

}

}

}

namespace Northwind.DataAccessLayer.Concrete.Entity\_Framework

{

public class EFAuthenticationDal : IAuthenticationDal

{

public User Authenticate(User user)

{

//Db üzərindən işləyəcək

if (user.UserName == "Khan" && user.Password == "123")

{

return user;

}

return null;

}

}

}

Təbii olaraq bu kodların cotroller daxilində işləyə bilməsi üçün son olaraq “Infrastructure => NinjectControllerFactory” controller-i daxilində aşağıdakı kodu yazmaq lazımdır.

\_ninjectKernel.Bind<IAuthenticationService>().To<AuthenticationManager>()

.WithConstructorArgument("authenticationDal", new EFAuthenticationDal());

***Controller kodları:***

namespace Northwind.MvcWebUI.Controllers

{

public class AccountController : Controller

{

private IAuthenticationService authenticationService;

public AccountController(IAuthenticationService \_authenticationService)

{

authenticationService = \_authenticationService;

}

// GET: Account

public ActionResult Login()

{

return View(new User());

}

[HttpPost]

public ActionResult Login(User user, string returnUrl)

{

User validateUser = authenticationService.Authenticate(user);

if (validateUser == null)

{

ModelState.AddModelError("Xəta", "İstifadəçi adı və ya şifrəsi yanlışdır");

}

if (ModelState.IsValid)

{

FormsAuthentication.SetAuthCookie(user.UserName, false);

return Redirect("http://localhost:51562/Admin/Index");

}

return View(new User());

}

}

}

***View kodları:***

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Login</title>

</head>

<body>

<div>

@using (Html.BeginForm())

{

@Html.ValidationSummary() @\*Error mesaji\*@

@Html.EditorForModel()

<input type="submit" value="Login" />

}

</div>

</body>

</html>

## Sistemi Service üzərindən host etmək.

İndiyə kimi olan bütün müraciətlər hər biri BLL üzərindən həyata keçirilirdi. Lakin Korporativ iş mühitində bu məlumatlar və proseslər əsasə web service-lər üzərindən icra olunur. Eyni zamanda Service üzərindən müraciət etmək daha təhlükəsiz və sürətli olduğunu demək lazımdır. Uyğun olaraq Solution daxilində yerləşən WcfLibrary daxilində servicə uyğun classlar yaradılaraq onların “svc” fie-larını yaratmaq lazımdır.

namespace Northwind.WcfLibrary.Concrete

{

public class AuthenticationService : IAuthenticationService

{

AuthenticationManager authenticationManager = new AuthenticationManager

(new EFAuthenticationDal());

public User Authenticate(User user)

{

return authenticationManager.Authenticate(user);

}

}

}

#### Qeyd: Service class-ı daxilində constructor metodan istifadə edilə bilməz. Onun ilə əlaqədar olaraq bir başa Manager classının obyekti yaradılmışdır.

namespace Northwind.WcfLibrary.Concrete

{

public class CategoryService : ICategoryService

{

CategoryManager categoryManager = new CategoryManager(new EfCategoryDal());

public List<Category> GetAll()

{

return categoryManager.GetAll();

}

}

}

#### UNUTMA: Burada birbaşa olaraq BLL -də yerləşən Manager classının obyekti yaradılmaq əvəzinə instance provider istifadə edərək çağırılma təmin edilməlidir.

<serviceActivations>

<add service="Northwind.WcfLibrary.Concrete.ProductService"

relativeAddress=" ProductService.svc"/>

<add service="Northwind.WcfLibrary.Concrete.CategoryService"

relativeAddress=" CategoryService.svc"/>

<add service="Northwind.WcfLibrary.Concrete.AuthenticationService"

relativeAddress=" AuthenticationService.svc"/>

</serviceActivations>

Yaradılmış service-ləri host edə bilmək üçün onların hər biri uyğun formatda web.config daxilində qeyd edilir.

#### Qeyd: Service üzərindən məlumat mübadiləsi yerinə yetirə bilmək üçün “Solution => Properties => Startup Project => Multiple startap project” hissəsindən həm service, həm də UI-in eyni zamanda “run” edilməsi təmin edilməlidir.

Service və UI eyni zamanda işə salınmasına baxmayaraq hal-hazırda sistem service üzərindən yox BLL üzərindən məlumat çekməsini təmin edir. UI-in service-ə müraciət edərək onun üzərindən əlyetənliyi təmin etmək üçün UI üzərində yaradılmış olan “infrasturucture” dovluğuna yeni class əlavə edilir. Yaradılan class satatik olaraq yaradılır və Proxy-nin əldə edilməsini təmin edəcəkdir.

namespace Northwind.MvcWebUI.Infrastructure

{

public static class WcfProxy <T>

{

public static T CreateChannel()

{

string address = String.Format("http://localhost:51537/{0}.svc?wsdl",

typeof(T).Name.Substring(1));

var binding = new BasicHttpBinding();

var channel = new ChannelFactory<T>(binding, address);

return channel.CreateChannel();

}

}

}

Nəzərə almaq lazımdır ki, sistem daxilində xeyli sayda service ola bilər və hər biri üçün ayrı-ayrı kod yazmaq əvəzinə class generic tip edilir (“WcfProxy <T>”) və buradan gələn service-ə uyğun olaraq onun əvvəlcə metod daxilində “adress”-i(*wsdl adresi*), sonra “binding”-i (*hansı protokol (tcp, http və s.) və ya təhlükəsizlik rejimində olması* ) və Bu ikisi əsasında “channel” yaradılır. (A*dress*B*inding*C*hannel*)

Qeyd edilmiş əməliyyatlar icra edildikdən sonra, UI-in service-ə müraciətini təmin etmək üçün “NinjectController” hissəsinə keçid etməklə “” yeni metodunu yaratmaq lazımdır. Beləliklə istifadəçi metod ilə service üzərindən müraciət etmiş olacaqdır.

private void AddServiceBindings()

{

\_ninjectKernel.Bind<IProductService>().ToConstant

(WcfProxy<IProductService>.CreateChannel());

\_ninjectKernel.Bind<ICategoryService>().ToConstant

(WcfProxy<ICategoryService>.CreateChannel());

\_ninjectKernel.Bind<IAuthenticationService>().ToConstant

(WcfProxy<IAuthenticationService>.CreateChannel());

}

#### Qeyd: UI-də yerləşən web.Config file daxilindəki connectionstrings-i kopyalayaraq Host proyekti daxilindəki web.Config daxilinə də əlavə etmək lazımdır.

# Klavye qısa yolları

* **Ctrl + K** **+D** – Kodu Səliqəli Formaya Salır.
* **Ctrl + K +C** – Kodu Commentə Alır.
* **Ctrl + K +U** – Kodu Commentdən Çıxarır.
* **Ctrl + C** – Kodu Kopyalayır.
* **Ctrl + X** – Kodu Kəsir.
* **Ctrl + V** – Kodu Yapıştırır.
* **Ctrl + Z** – Bir Əvvəlki Vəziyyətə Qaytarır.
* **Ctrl + Y** – Yeri Qayıtdıqdan Sonra Bir İrəli Koda Qayıdır.
* **Ctrl + A** – Bütün Kodu Seçir.
* **Ctrl + D** – Kursor olan sətrin eynisini aşağı atır.
* **Alt + Shift** – Sütun boyunca seçmək
* **Alt + Shift + Enter** – Full ekran (*Çıxmaq üçün eyni kod*)

# Dərs 1. İlkin tanışlıq və ilkin anlayışlar

VS code vasitəsilə html kodlarını yazmaq mümkündür. İlk əvvəl yeni fayl əlavə edilir. Əlavə edilmiş fayl “index.html” adlandırılaraq bu faylın bir html kodlarını əks etdirən fayl olması qeyd edilir (*Frontend proyektlərin hazırlanması zamanı fayl adı əsasən index.html kimi yazılır. Digər hallarda isə müxtəlif tipli xətalarla qarşılaşmaq mümkündür*).

Fayl yaradıldıqdan sonra html-in skletini təşkil edən kodu editor imkanı ilə “!+Tab” kobmbinasiyasini edərək yaratmaq olar.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

</body>

</html>

“DOCTYPE html” brauzerlərin bu kodları html kodları kimi anlamasını təmin edən açar sözdür. “<” işarəsi ilə başlayan və ondan sonra yazılan sözlə birlikdə olan konbinasiyaya “tag” deyilir. Htlm kodu “<html> “ tag-i ilə başlayır və bitir.

Tag-lar iki yerə bölünür: Bağlanan və bağlanmayan. “html” tag-i bağlanan təqlər sinifinə aiddir. Bağlanan tag-lar “<html> </html> “ kimi yazılır.

#### Qeyd: Əgər bir tag daxilində başqa bir tag açılıbsa, daxilindəki tag mütləq formada ondan əvvəl olan tag bağlanmamış bağlanmalıdır.

Html kodu iki hissədən “head” və “body” hissəsindən ibarət olur. Head hissəsində yazılan kodlar səhifınin hansı qaydalar əsasında işləyəcəyini təyin edir. Burada “title” tag-i ilə səhifələrin başlıq hissələrini vermək olur. Eyni zamanda bu sahədə CSS kodlarıda yazıla bilər. Body hissəsi isə bütün kodların yer aldığı sahədir.

Body hissəsində olan tag-lar:

**1. div** – ekran boyunca səhəni tutan bir atributdur. Div daxilində müxtəlif tipli tag-lar istifadə etmək olar. Eyni zamanda kod başlanğıcında müxtəlif atributlardan istifadə edərək divlərə mextəlif dizayn imkanları yaraqmaq mümkündür.

**2. span** – cari səhr hissəsində sadecə tag daxilində yazılmış kodu əhatə edir. Misal olaraq mətin daxilində bir sözü fərqləndirmək üçün istifadə edilə bilər.

**3. b** – sözün qalın yazılmasını təmin edir.

**4. i** – sözün itallik yazılmasını təmin edir.

**5. u** – sözün altında xətt yazılmasını təmin edir.

**6. p** – Paraqraf tag-i div tag-i kimi ekranın tamını tutan bir atributdur.

**7. br** – qeyd olunduğu yerdən məlumatların və ya mətnin bir sətr aşağı düşməsini təmin edir.

**8. img** – səhifə daxilində şəkil yükləmək üçün istifadə olunur. Əsasən “png” və “jpg(jpeg)” tipli fayllardan istifadə edilir. Bunun üçün img tag-nin “src” atributu vasitəsi ilə ya lokal unvan yada online URL verilə bilər.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body style="background-color: tomato;">

    <div>

        Div umumi olaraq ekranin ayrilmis olan hissesini tam olaraq goturur

    </div>

    <span>Cari setr daxiline span daxilindeki erazini secir</span>

    <p>Paraqraf div-de oldugu kimi ekranin umumu hissesini alir</p>

    <br> <span>br metnin yeni setre enmesini temin edir</span>

    <br> <b>Qalin soz</b> <br> <i>Itallik soz</i> <br> <u>altdan xett</u>

<div><img src="images/ms.jpg" width="70%" style="height: 500px;">

</div>

<div><img src="https://www.stickpng.com/assets/images/59db69c73752880e93e16efb.png"> </div>

</body>

</html>

#### Qeyd: “p” və ya “div” tag-ları daxilində “lorem” yazaraq hazır mətn gətirmək mümkündür.

#### Qeyd: Səhifə daxilində istifadə olunan faylları uyğun adlı qovluqlarda saxlamaq daha məqsədə uyğun hesab edilir.

#### 

# Dərs 2. Css - class, style, id anlayışları

Bütün tag-lar üçün ümumi olan 3 atribut vardır: Class, style, id. Bu atributlar vasitəsi ilə css kodlarını tətbiq edərək tag-ların hər birinə ayrı-ayrılıqda kod yazmaq əvəzinə onları bir yerdə yazaraq hər dəfə lazım olduqca çağırmanı təmin edə bilərik.

İlkin olaraq Head hissəsində “style” tag daxilində css kodlarının yazılmasından istifadə etmək olar. Bunun üçün hər bir atrubutu uyğun olaraq aşağıdakı kimi kodlar yazaraq cağırmaq olur.

**1. Class** – Tag daxilində class çağırılaraq onun daxilində yazılmış kodlarının tətbiq edilməsi üçün head sahəsində qeyd edilmiş class adı daxilində mütləq formada “.className” kimi yazılışdan istifadə edilməlidir. Bu yazılış əsasında yazılan kodarı isə tag-lar daxilində asanlıqla çağırmaq mümkündür.

#### Qeyd: Nəzərə almaq lazımdır ki, Tag daxilində birdən çox css classı və js classları çağırmaq olar. Bunun üçün sadecə olaraq hər bir classı boşluq ilə yazmaq tələb olunur.

#### Qeyd: Eyni dizaynı özündə əks etdirən classlar olarsa onları “,”-dən istifadə edərək yazmaq kifayətdir. Bu zaman yazılan classlar daxilində onların digər fərdi xüsusiyyətləri olarsa, onlarda ayrı formatda yazıla bilər.

**2. Tag** – lar üçün css kodları yazan zaman onların əvvəlinə hansı tag-a təqbiq olunacağı və class adı yazılır. “p.p1”

**3**. id – lər adlandırılan zaman hərf ilə başlamalıdır və unikal olmalıdır. İd yaratmaq üçün “#Id” formatında yazılır.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="">

    <style>

    p.p1, .nn{

        border: yellowgreen;

        background: tomato;

        padding: 25px;

    }

.nn{

    margin: 12px 22px 10px 15px;

}

.pp{

    margin-top: 12px;

    margin-left: 22px;

    margin-bottom: 10px;

    margin-right: 15px;

}

#i1{

    color: crimson;

}

    </style>

</head>

<body style="background: wheat;">

    <div class="nn">Lorem ipsum dolor sit amet consectetur adipisicing elit. Fugiat quis odit dolorem nihil! Quia nisi, commodi inventore libero suscipit doloribus a fuga recusandae doloremque debitis enim laboriosam repellat, dolore veniam.</div>

    <p id="i1">Lorem ipsum dolor sit amet consectetur adipisicing elit. Quia amet cupiditate provident rerum voluptas placeat modi excepturi, nisi laboriosam vitae asperiores maiores praesentium magnam pariatur velit cumque, eos incidunt enim.</p>

    <div><p></p></div>

    <div class="p1" >

        <p class="nn">Lorem ipsum dolor, sit amet consectetur adipisicing elit. Similique repellat pariatur saepe, quaerat, ipsa, perspiciatis est odit eaque ea velit quo. Et obcaecati hic enim quas mollitia nihil aspernatur tenetur?</p>

        <img src="" alt="">

    </div>

    <script src=""></script>

</body>

</html>

**Panding** daxil olduğu yerdən(*div*) “top, left, buttom, right” olmaqla məsafə vermək üçün istifadə edir. Burada Sadecə “pandig:15px” yazaraq hər tərəfdən eyni məsafə, “panding-top:15px” yazaraq yalnız istənilən yerdən, “panding: 15px 10px 20px 40px” yazaraq yuxarıdakı ardıcıllığa uyğun şəkildə məsafə vermək və son olaraq “panding: 15px 10px” (*ekvivalentdir=>15px 10px 15px 10 px*) yazıla bilər.

**Marging** ekrandan kənarlarından təyin etilmiş məsafə qədər boşluq vermək üçün istifadə edilir. Yazılma tərzi “panding”-də olduğu kimidir.

#### Qeyd: Kodları sürətli formatda yazmaq üçünEMMAT-lardan istifadə etmək olar. Bunun üçün “div.nn>(p>span)+img” tipli kodları yazaraq aşağıdakı kodun yaranmasını görmək olar.

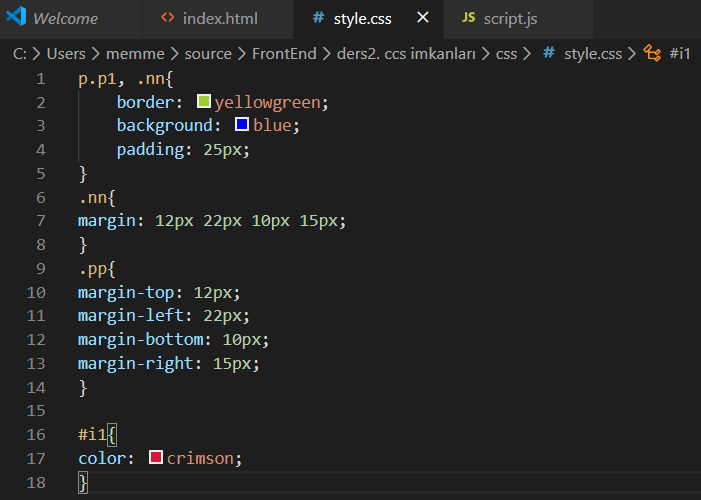
    <div class="nn">

        <p><span></span></p>

        <img src="" alt="">

    </div>

#### Qeyd: Yazılmış kodları fərqli olaraq css file daxilində yaradaraq onları “link” tag-i vasitəsi ilə çağırmaq mümkündür.



## Səhifə daxili və digər səhifələrə istinad

Bunun üçün “a” tag-dən istifadə edilir. Bunun üçün “Href” daxilində keçid ediləcək ünvan və onun adlandırılacağı məlumat yazılır. Burada əgər istinad verdiyinin linkin yeni səhifədə açılmasını istəyiriksə bu zaman “target= “\_blank” atributundan istifadə edilir.

    <a href="https://jsfiddle.net/" target="\_blank"> Tapsiriqlari burada yaz</a>

    <a href="about.html">about </a>

Eyni zamanda səhifə daxilində “anchor-lövbərlərdən” istifadə etmək olar. Anchor-ların məqsədi işarələnmiş hissəyə bir başa yönəlməni təmin edir. (*Misal olaraq şəkil haqqındakı məlumatları səhifənin sonuna ataraq orada anchor istifadə etməklə bir başa ora keçid etmək olar.*) Bunun üçün lövbər atılan yerə ad verilir və istinad həmin ada uyğun olaraq yaradılır.

<body style="background: wheat;">

    <a name="top"></a>

. . .

    <a href="#top">Evvele qayit</a>

</body>

## Başlıqlar

Başlıqlar “h” tag-ları ilə ifadə olunur. 1-dən 6-ya kimi ölçüləri var. Uyğun olaraq getdikcə ölçüləri kiçilir.

#### Qeyd: Başlıqlar mətn hissəyə aid deyil. Uyğun olaraq onları “p” tag-ı içində yazmaq olmaz. Eyni zamanda “h” tag-dan sonra “br” tag-dən istifadə etməyə ehtiyac yoxdur.

    <h1>lorem</h1>

    <h2>lorem</h2>

    <h3>lorem</h3>

    <h4>lorem</h4>

    <h5>lorem</h5>

    <h6>lorem</h6>

Əlavə olaraq bu təqlərə style imkanlarıda vermək mümkündür.

    <h1 style="text-align: center;font-size: 50px;">lorem</h1>

## Səhifəni ayıran horizontal xətt

Əgər səhifənin istənilən bir yerində horizontal xətt qoymaq tələbi yaranarsa, bu zaman “hr” tag-dan istifadə edilir.

    <hr style="border-width: 5px;">

## Text daxilində sistat

Əgər text daxilində sitat gətirmək lazım gələrsə, “Blockquote” tag-dan istifadə edilir.

blockquote{

    background: #000; color: yellow; padding: 3px;

    width: 50px;    float: right;

}

p{

margin-bottom:15px ;    clear: both;

}

    <p>

        Lorem ipsum dolor sit amet consectetur adipisicing elit. Obcaecati nobis laboriosam eaque alias illum dolorem totam quis rerum nostrum eveniet unde odio cumque, exercitationem similique facere veritatis vero ratione assumenda?<blockquote>SITAT</blockquote> Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora corporis molestiae ullam qui, facilis atque quisquam excepturi eligendi voluptas reprehenderit rerum id temporibus! Eaque reiciendis porro quibusdam possimus ipsum illo. Lorem ipsum dolor, sit amet consectetur adipisicing elit.

    </p>

## Code daxilindəki bütün icraları göstərmək

Normal “p” tag-i daxilində “enter” klavişindən istifadə edən zaman o sadecə olaraq bir ədəd boşluq kimi anlaya bilir. Lakin elə bir hall yarana bilər ki, biz yazdığımız mənt file üçün və ya hər hansı bir başqa proses üçün code necə yazılıbsa(enter klavişi neçə dəfə klik edilibsə), onların hər biri ekranda əks edilsin. Bu zaman “pre” tag-dan istifadə edilir.

    <pre>

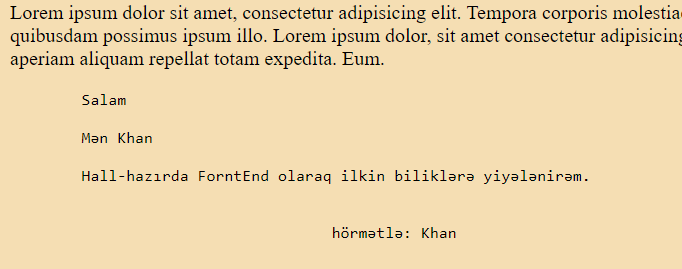
        Salam

        Mən Khan

        Hall-hazırda ForntEnd olaraq ilkin biliklərə yiyələnirəm.

                                    hörmətlə: Khan

    </pre>



## Text formaları

**Bold** – “**b**” və “**strong**” tag-ları

**İtalic** – “**i**” və “**em**” tag-ları

# Dərs 3. Emmet (qısaltma) anlayışı və bəzi html tag-ları

Emmet anlayışı qısaldılmış formada kodların daha sürətlə yazılmasını təmin edən əsas prosesdir. Bunun üçün Emmeti daha geniş şəkildə öyrənmək üçün <https://emmet.io/> saytından istifadə etmək olar. Misal olaraq aşağıdakı kodu yazaraq “enter” düyməsini sıxdıqdan sonra növbəti html kodları ekranda əks ediləcəkdir.

    div#page>div.logo+ul#navigation>li\*5>a{item$}

<div id="page">

    <div class="logo"></div>

    <ul id="navigation">

        <li><a href="">item1</a></li>

        <li><a href="">item2</a></li>

        <li><a href="">item3</a></li>

        <li><a href="">item4</a></li>

        <li><a href="">item5</a></li>

    </ul>

</div>

#### Qeyd: Kod Daxilində vertikal (yuxarıdan aşağı) seçim etmək üçün shift+alt kombinasiyasından istifadə etmək lazımdır. Əlavə olaraq shift+alt və hərəkət oxlarından istifadə edərək söz-söz seçmək, başqa sözlə desək bir sonrakı sözü simvol sayları fəqrli olarsa belə sütun kimi seçmək imkanı yaradır.

## Vertikal olaraq avto rəqəmlər daxil etmək.

Qeyd etmək lazımdır ki, vertikal olaraq seçim etdikdən sonra onların sonuna və ya hər hansı tələb olunan sahəyə ardıcıl və yaxud müxtəlif addımla rəqəmlərin yazılması tələbi formalaşa vilər. Bunun üçün VS code editorunun “TEXT PASTRY” (yoxdursa əlavə etmək lazımdır) tool-dan istifadə etmək olar. Bunun üçün ctrl+shift+p kombinasiyasından istifadə edərək açılan search bar hissəsinə “text pastry 1 to X” yazmaqla şeçilmiş sahəyə ardıcıl rəqəmləri yazmaq mümkündür.

**Yazının üzərində xətt vermək** – “**s**” tag-i

**Yazını olduğundan böyük göstərmək** – “**big**” tag-ı

**Yazını olduğundan kiçik göstərmək** – “**small**” tag-ı

**Yazını sağdan sola hərəkət etdirmək** – “**marquee**” tag-ı

**Yazını indexsə yazmaq**  – “**sub**” tag-ı

**Yazını üstlü qüvvətə yazmaq** – “**sup**” tag-ı

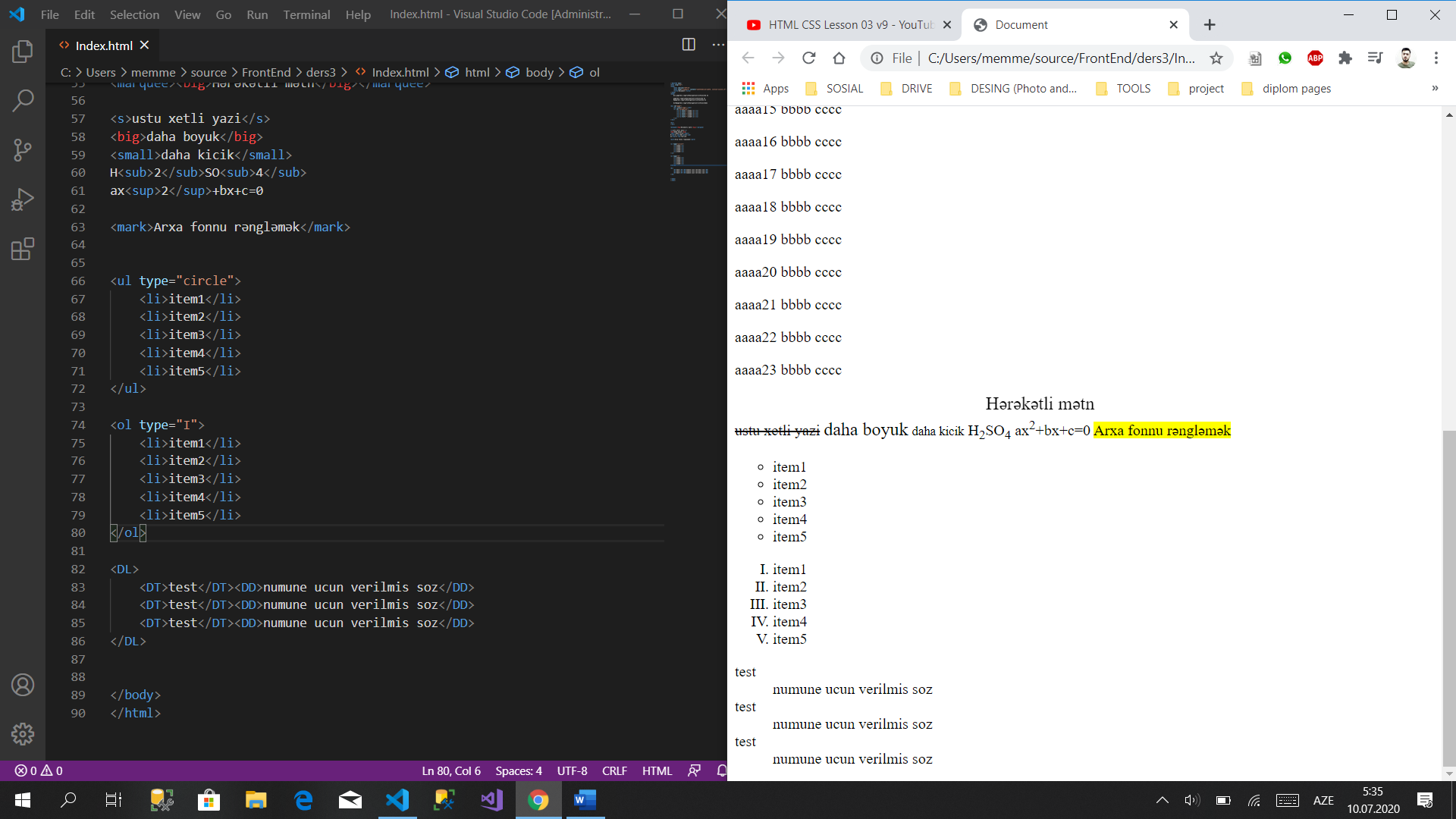
**Yazını arxa hissəsini rəngli etmək** – “**mark**” tag-ı

**Yazını işarələr əsasında sıralamaq (Bullets) –** “**ul**” tag-i daxilində “**li**” tag-i

**Yazını rəqəmlər əsasında sıralamaq (Numbering) –** “**ol**” tag-i daxilində “**li**” tag-i

#### Qeyd: “ul” və “ol” tag-ları daxilində “type” atributundan istifadə edərək sıralama vasitələrini dəyişmək mümkündür.

**Yazının başlıq və onun açıqlamasını abzasdan yazmaq – “DL**” tag-i daxilində “**DT**” tag-i baş, “**DD**” tag-i isə açıqlama hissəsi üçün uyğun tag-dan istifadə edilir.



<DL>

    <DT>test</DT><DD>numune ucun verilmis soz</DD>

    <DT>test</DT><DD>numune ucun verilmis soz</DD>

    <DT>test</DT><DD>numune ucun verilmis soz</DD>

</DL>

## İmg tag-i və onun atributları

“**scr**” – şəkilin hansı ünvanda yerləşməyi link olaraq buraya daxil edilir.

“**width**” – şəkilin öz ölçüsü ilə yox təyin edilmiş ölçü ilə gəlməsini təmin edir. Bu zaman hündürlük avomatik olaraq uyğunlaşır.

“**alt**” – bu atributun istifadəsi o zaman işə yarayır ki, şəkilin çekilməsi zamanı hər hansı bir səbəbdən şəkil gəlməzsə bu zaman bu atribut daxilində yazılmış ifadə əks edilir və şəkilin bərpası daha rahat olur.

“**title**” – bu atribut vasitəsi ilə şəkil əks olunan zaman mausu onun üzərində saxladığımız zaman atribut daxilində yazılmış ifadə kiçik pəncərə formasında qısa müddət üçün əks etdirilir.

<img src="images/ms.jpg" alt="vs logosu" title="VS LOGO" width="200px" style="float: left; margin: 10px 15px 15 px 0;">

#### Şəkilin əlavə edilməsi və onun bir paraqraf daxilində yazılan zaman səliqəsiz görünməsinin qarşısını almaq üçün “style” atributu daxilində şəklin sol tərəfə sabitlənməsi üçün “float” ifadəsindən istifafə edilir.

## Table tag-i və onun atributları

Cədvəlin hazırlanması üçün ilkin olaraq “table” tag-i, onun içində sətri ifadə edən “tr” tag-i və xanaları ifadə edən “td” tag-dan istifadə edilərək yaradılır. Əgər sətr daxilində bir neçə xana olması tələb olunarsa, bu zaman göstərilmiş kod kombinasiyasina uyğun olaraq kodlama prosesi icra edilməlidir.

Əlavə olaraq qeyd etmək lazımdır ki, cədvəl daxilində sütunları ifadə edən sütun başlıqlarından istifadə etmək üçün “tr” daxilində “th” tag-dan istifadə edərək başlıqların digər xanalardakı mətnlərə nisbətən daha qalın olmasını təmin etmək olar.

Table tag=i daxilində müxtəlif atributlardan istifadə etmək mümkündür. “border” xənar xətti əks etdirir və ona bərabər edilən dəyər cədvəlin kənar xəttinin qalınlığını ifadə edir, “cellspacing” xanalar arası boşluq, “cellpadding” xana daxilində mətnin xanadan uzaqlığını ifadə edir.

“rowpan” ifadə edilən qədər sətrin birləşdirilməsini, “colspan” ifadə edilən qədər sütunun birləşdirilməsini təmin edir.

<table border="2" cellspacing="5" cellpadding="5">

    <thead>

        <tr>

            <th>basliq1</th>

            <th>basliq2</th>

            <th>basliq3</th>

        </tr>

    </thead>

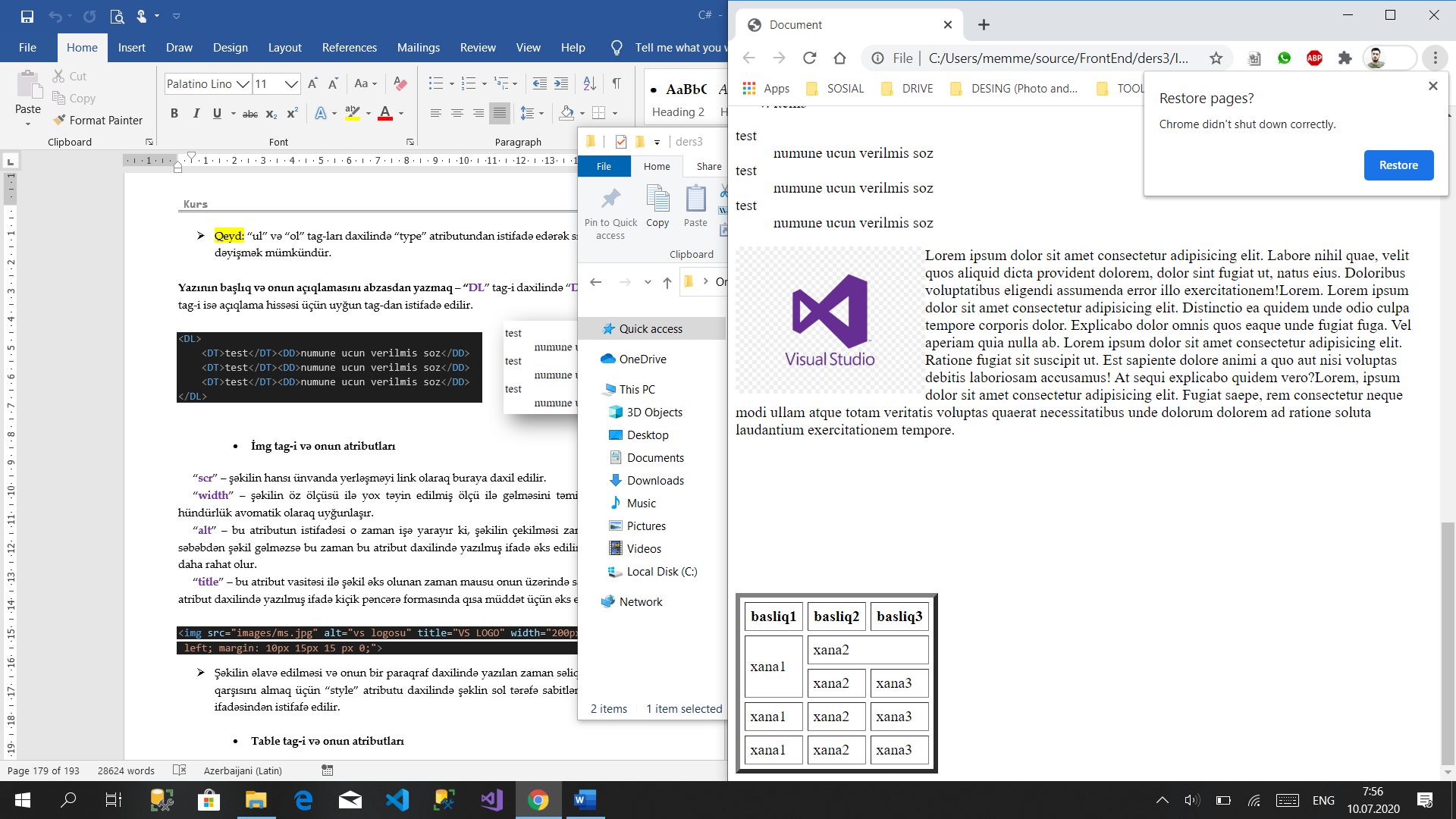
    <tr>

        <td rowspan="2">xana1</td>

        <td colspan="2">xana2</td>

    </tr>

    <tr>

        <td>xana2</td>

        <td>xana3</td>

    </tr>

    <tr>

        <td>xana1</td>

        <td>xana2</td>

        <td>xana3</td>

    </tr>

    <tr>

        <td>xana1</td>

        <td>xana2</td>

        <td>xana3</td>

    </tr>

</table>

# Dərs 4.

# PROYEKT HAZIRLANMASI ZAMANI YARANAN ÇƏTİNLİKLƏR

## Formlar arası keçid və məlumat mübadiləsi.

Formlar arası keçid etmək üçün aşağıdakı kod kobinasiyalarından istifadə etmək mümkündür.

this.Hide(); // this cari form menasina gelir

Form2 f2 = new Form2(); // form2-nin obyekti yaradilir

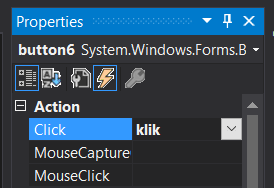
f2.Show(); // form2 acilan zaman form1 baglanir

f2.ShowDialog(); // form2 acilan zaman form1 aktiv qalir

#### Qeyd: Bir formdakı komponentlərin digər formda görülə bilməsi üçün Formlardaki komponentlərin MODİFİERS imkanları private halından public halına keçirilməlidir.

## Kilit ekran ve s. Progrsmlarda Buttonların “button\_click” events-ləri

Kilit ekran, kalkulyator və s. kimi *(1-9 və başqa formada)* buttonların istifadəsi lazım gələrsə, hər bir button daxilinə yazmaq əvəzinə buttonların CLİCK EVENTS-indən istifadə edərək *(hər birinə eyni məlumat yazılmalıdır)* aşağıdakı kombinasiyadan istifadə etmək olar.

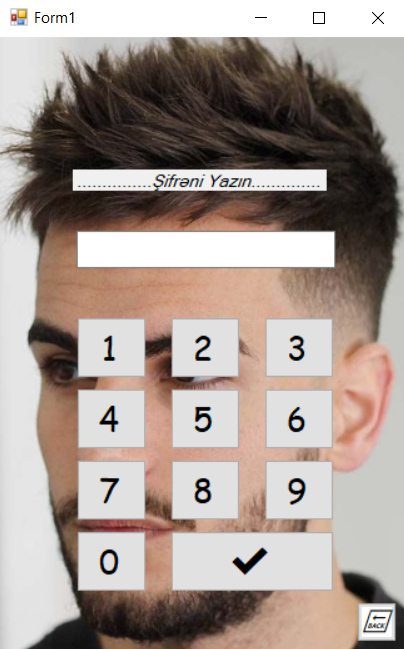


private void klik(object sender, EventArgs e)

{

Button duyme =(Button)sender;

textBox1.Text += duyme.Text;

 }

## Parol yazılan zaman simvollar \*\* görülməsi

TextBox-a girilən dəyərlərin məxfiliyik tələbləri olarsa, bu zaman daxil edilən simvolların ekranda görülməməsini təmin etmək üçün TEXTBOX-un PROPERTİES-dən “UseSystemPasswordChar” hissəsini TRUE edə bilər.

## Escape Sequance

1. \’ – string daxilində tek dırnaqda yazmaq

string soz = "\'a\'";

‘a’

1. \” – string daxilində cüt dırnaqda yazmaq

string soz = "\"a\"";

“a”

1. \\– string daxilində \ simvolu yazmaq üçün

string soz = "\\a";

\a

1. @ – string daxilində bir ədəd \ simvolu yazısını iki ədəd kimi ifadə edir. *(path qeyd edəndə)*

string path1 = @"C:\Users\memme\source\log\ders13\" + filename + ".txt";

1. \n – string daxilində yazılan zaman özündən sonrakı məlumatı bir sətr aşağı endirir.

string club = "Real\nMadrid";

Real

Madrid

1. \r – string daxilində yazılan zaman özündən əvvəl gələn məlumatı silir.

string club = "Real\rMadrid";

Madrid

1. \a – string daxilində istənilən yerdə qeyd edilərək açılan pəncərədə bildiriş səsi verir

string club = "Real\aMadrid";

## Klaviaturadan ENTER və ESCAPE düymələrinə basan zaman işləyən button - ları təyin etmək.

### I yol

Form-un Properties-dəki “Misc” bölməsi daxilində yerləşən AcceptButton – CancelButton hissəsindən lazımi buttonları seçmək.

### II yol

Form-un Load hissəsinə yazmaqla. AcceptButton = button2; CancelButton = button2;

## EYNI ANDA TOPLU INSERT

#### Qeyd: Eyni vaxtda insert əmri ilə iki və daha çox sətr əlavə etmək.

INSERT KITABLAR VALUES ('VERITABANI YONETIM SISTEMI 2', 'TURGUT OZSN', 33.50, '2002-10-23')

('VERI YONETIMI', 'ASIL MUTLU', 25.20, '2008-06-23') . . .

## SELECT ILE INSERT

INSERT DBO.KITABLAR\_RENG(KITAB\_AD) SELECT KL.KITAB\_AD FROM DBO.KITABLAR KL

## Dama Taxtası

namespace \_40.ButtonlarlaDamaTaxtasi

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

Button[,] buttons = new Button[8, 8];

int top = 0; int left = 0;

for (int i = 0; i < buttons.GetUpperBound(0); i++)

{

for (int j = 0; j < buttons.GetUpperBound(1); j++)

{

buttons[i, j] = new Button();

buttons[i, j].Width = 50;

buttons[i, j].Height = 50;

buttons[i, j].Left = left;

buttons[i, j].Top = top;

left += 50;

if ((i+j)%2 == 0)

{

buttons[i, j].BackColor = Color.Black;

}

else

{

buttons[i, j].BackColor = Color.White;

}

this.Controls.Add(buttons[i, j]);

}

top += 50;

left = 0;

}

}

}

}

# INTERVIEW SSENARI

1. **VARIABLES (TYPES) VALUE TYPES & REFERANS TYPES,(DEYISHENLER)**
2. **OPERATORS *(XETTI ALQORITM)*** 
   1. ARIFMETIK
   2. MENTIQI
   3. MENIMSETME
   4. QARSHILASDHIRMA
3. **OPERATORS *(BUDAQLANAN)***
   1. IF,
   2. SWITCH
4. **LOOPS *(DOVRI ALQORITMLER)*** 
   1. FOR,
   2. FROEACH,
   3. WHILE,
   4. DO WHILE
5. **ARRAYS and METHODS**
6. **COLLECTIONS and METHODS**
   1. ARRAYLIST,
   2. LIST
7. **STRINGS and METHODS**
8. **TRY CATCH - EXCEPTION**
9. **METHODS**
   1. OVERLOAD,
   2. OVERRIDE
10. **OOP** 
    1. CLASS,
    2. ENCAPTULATION,
    3. INHERITANCE,
    4. POLYMORPHIZM,
    5. CONSTURCTOR
11. **ADO.NET**
    1. CONNECTED METHODU,
    2. DISCONNECTED METHODU
12. **WEB SERVICES**
    1. ASMX,
    2. WCF
13. **ASP.NET-MVC *(MODEL, VIEW , CONTROLLER)***
14. **ORM - OBJECT RELATIONAL MAPPING *(ENTITY FRAMEWROK)***

MÜNDƏRİCAT

[Types and Variables 4](#_Toc40629972)

[DƏRS 1. (OPERATORLAR) 6](#_Toc40629973)

[ Arifmetik 6](#_Toc40629974)

[ Mənimsətmə 6](#_Toc40629975)

[ Qarşilaşdirma 6](#_Toc40629976)

[ Məntiqi 6](#_Toc40629977)

[ Xüsusi 6](#_Toc40629978)

[DƏRS 2. (OPERATORLAR) 7](#_Toc40629979)

[ Dəyişən tipləri 7](#_Toc40629980)

[ Şərt Operatorları (IF, SWITCH) 7](#_Toc40629981)

[DƏRS 3. (İÇ-İÇƏ IF FUNKSİYASİ) 9](#_Toc40629982)

[ CheckBox haqqında ilk məlumatlar 9](#_Toc40629983)

[DƏRS 4. (DÖVR OPERATORLARI) 9](#_Toc40629984)

[ Null və Empty anlayışı 9](#_Toc40629985)

[ Dövr operatorları ( For, Foreach, While, Do while) 9](#_Toc40629986)

[ WHILE => Verilmiş aralıqdakı dəyərlərin cəmi 9](#_Toc40629987)

[ Do WHILE 9](#_Toc40629988)

[ ForEach 10](#_Toc40629989)

[DƏRS 5. (İÇ-İÇƏ DÖVR OPERATORU) 11](#_Toc40629990)

[ \n açar sözü 11](#_Toc40629991)

[ Dövr daxilində istifadə edilən keyword-lər ( CONTİUNE, BREAK, RETURN) 11](#_Toc40629992)

[DƏRS 6. (ARRAYS-MASSİVLƏR) 12](#_Toc40629993)

[TAPŞIRIQ 12](#_Toc40629994)

[ Array yaratmağın növləri 12](#_Toc40629995)

[ Çox ölçülü array 13](#_Toc40629996)

[DƏRS 7. (ARRAY METHODS) 14](#_Toc40629997)

[ IndexOf \*\*\* 14](#_Toc40629998)

[ LastIndexOf \*\*\* 14](#_Toc40629999)

[ Contains 14](#_Toc40630000)

[ Length 14](#_Toc40630001)

[ Resize \*\*\* 14](#_Toc40630002)

[ CreateInstance \*\*\* 15](#_Toc40630003)

[ SetValue 15](#_Toc40630004)

[ GetValue 15](#_Toc40630005)

[ Sort \*\*\* 15](#_Toc40630006)

[ Reserve \*\*\* 15](#_Toc40630007)

[TAPŞIRIQ 16](#_Toc40630008)

[DƏRS 8. (COLLECTİONS) 17](#_Toc40630009)

[ Generic 17](#_Toc40630010)

[ NonGeneric 17](#_Toc40630011)

[ARRAYLİST 17](#_Toc40630012)

[ARRAYLİST METHODS 17](#_Toc40630013)

[ Add 17](#_Toc40630014)

[ AddRange 17](#_Toc40630015)

[ Remove 18](#_Toc40630016)

[ RemoveAt 18](#_Toc40630017)

[ RemoveRenge 18](#_Toc40630018)

[ Insert 18](#_Toc40630019)

[ InsertRange 18](#_Toc40630020)

[ Count 18](#_Toc40630021)

[ Capacity 19](#_Toc40630022)

[ ToTrimSize 19](#_Toc40630023)

[ Clear 19](#_Toc40630024)

[ CopyTo 19](#_Toc40630025)

[ Sort, Reserve, IndexOf, LastIndexOf, Contains 19](#_Toc40630026)

[DƏRS 9 və 10. (LIST və STRING) 20](#_Toc40630027)

[LIST və METHODS 20](#_Toc40630028)

[STRING METHODS 21](#_Toc40630029)

[ ToString 21](#_Toc40630030)

[ IndexOf 21](#_Toc40630031)

[ LastIndexOf 21](#_Toc40630032)

[ Contains 21](#_Toc40630033)

[ Length 21](#_Toc40630034)

[ Trim 21](#_Toc40630035)

[ TrimStart 22](#_Toc40630036)

[ TrimEnd 22](#_Toc40630037)

[ IsNullOrEmpty \*\*\* 22](#_Toc40630038)

[ IsNullOrWhiteSpace \*\*\* 22](#_Toc40630039)

[ ToUpper 22](#_Toc40630040)

[ ToLower 22](#_Toc40630041)

[ Compare \*\*\* 23](#_Toc40630042)

[ Copy \*\*\* 23](#_Toc40630043)

[ ToCharArray 23](#_Toc40630044)

[ Replace 23](#_Toc40630045)

[ Remove 23](#_Toc40630046)

[ Concat \*\*\* 23](#_Toc40630047)

[ SubString 24](#_Toc40630048)

[ Split 24](#_Toc40630049)

[ Join \*\*\* 24](#_Toc40630050)

[TAPŞIRIQ 24](#_Toc40630051)

[DƏRS 11. (RANDOM və TİMER) 25](#_Toc40630052)

[ Random 25](#_Toc40630053)

[ Timer 26](#_Toc40630054)

[TAPŞIRIQ 26](#_Toc40630055)

[DƏRS 12. (Try Catch və Dialog Windows) 27](#_Toc40630056)

[ Try Catch - Exception 27](#_Toc40630057)

[ Dialog windows 29](#_Toc40630058)

[SAVE File Dialog 29](#_Toc40630059)

[OPEN File Dialog 30](#_Toc40630060)

[DƏRS 13. (DateTime Class, StreamWriter and StreamReader, File Class Methods) 31](#_Toc40630061)

[ DateTime Class və DateTimePicker 31](#_Toc40630062)

[ StreamWriter 32](#_Toc40630063)

[ StreamReader 32](#_Toc40630064)

[ FileClassMethods 33](#_Toc40630065)

[WriteAllText 33](#_Toc40630066)

[AppendAllText 33](#_Toc40630067)

[ReadAllText 33](#_Toc40630068)

[Exists 33](#_Toc40630069)

[Move 33](#_Toc40630070)

[Replace 34](#_Toc40630071)

[DƏRS 14. (Directory Class) 35](#_Toc40630072)

[ CreateDirectory 35](#_Toc40630073)

[ Delete 35](#_Toc40630074)

[ Exists 35](#_Toc40630075)

[ GetCreationTime 35](#_Toc40630076)

[ GetCurrentDirectory 35](#_Toc40630077)

[ GetFiles 35](#_Toc40630078)

[ GetLastAccessTime 35](#_Toc40630079)

[ Move 36](#_Toc40630080)

[ GetLogicalDrives 36](#_Toc40630081)

[ GetParent 36](#_Toc40630082)

[ GetDirectories 36](#_Toc40630083)

[DƏRS 14. (Logs – [AppendAllText, StreamWriter, Nlog] ) 37](#_Toc40630084)

[ AppendAllText 37](#_Toc40630085)

[ Nlog 38](#_Toc40630086)

[ StreamWriter 39](#_Toc40630087)

[DƏRS 16. (Methods ) 41](#_Toc40630088)

[TAPŞIRIQ 42](#_Toc40630089)

[ ref - Keyword- ü 43](#_Toc40630090)

[ out - Keyword- ü 43](#_Toc40630091)

[DƏRS 17. (Method’s Overload and Class ) 44](#_Toc40630092)

[ Class 44](#_Toc40630093)

[ Overload 46](#_Toc40630094)

[ params - Keyword- ü 47](#_Toc40630095)

[DƏRS 18. (OOP – Encapsilation, inheritence ) 48](#_Toc40630096)

[ Encapsilation 48](#_Toc40630097)

[ İnheritance 48](#_Toc40630098)

[DƏRS 19. (OOP – Polymorphyzm, Constructor metodu, static-nonstatic class və metodlar, Access Modifires ) 50](#_Toc40630099)

[ Override 50](#_Toc40630100)

[ Constructor metodu. 51](#_Toc40630101)

[ Static və NonStatic class yaxud metod anlayışı. 52](#_Toc40630102)

[ Access Modifiers 53](#_Toc40630103)

[DƏRS (Interface) 54](#_Toc40630104)

[ Nümunə 57](#_Toc40630105)

[ SOLID, Interface Segragation (Inteyfeyslərin doğru planlanması) 59](#_Toc40630106)

[SQL TİPLƏRİ 62](#_Toc40630107)

[DƏRS 20. (SQL – ilkin anlayşlar [SQLServer ilə]) 63](#_Toc40630108)

[DƏRS 21. (DDL, DML) 65](#_Toc40630109)

[ DDL 65](#_Toc40630110)

[ DML 66](#_Toc40630111)

[ EYNI ANDA TOPLU INSERT 66](#_Toc40630112)

[ SELECT ILE INSERT 66](#_Toc40630113)

[DƏRS 22. (SELECT, WHERE, LIKE, BETWEEN, IN, AS, COUNT GROUP BY, ORDER BY, DISTINCT, JOINS) 67](#_Toc40630114)

[ WHERE 67](#_Toc40630115)

[ IN 67](#_Toc40630116)

[ LIKE 67](#_Toc40630117)

[ ORDER BY 67](#_Toc40630118)

[ AND, OR, BETWEEN, >, <, < > 67](#_Toc40630119)

[ AS 67](#_Toc40630120)

[ COUNT 68](#_Toc40630121)

[ DISTINCT 68](#_Toc40630122)

[ GROUP BY 68](#_Toc40630123)

[ ƏLAVLƏR ( '[^A]%', '[A-D]%', '[ADV]%' ) 68](#_Toc40630124)

[ SQL JOINS 68](#_Toc40630125)

[INNER JOIN 69](#_Toc40630126)

[LEFT JOIN 69](#_Toc40630127)

[RIGHT JOIN 69](#_Toc40630128)

[FULL JOIN 69](#_Toc40630129)

[ ƏLAVLƏR 69](#_Toc40630130)

[INNER JOIN EKVİVALENTİ 69](#_Toc40630131)

[DƏRS 23. (PROCEDURE, VİEWS) 71](#_Toc40630132)

[ Procedure 71](#_Toc40630133)

[ Views 72](#_Toc40630134)

[VIEW - ENCRYPTION 73](#_Toc40630135)

[DƏRS 24-25. (TRIGGER) 74](#_Toc40630136)

[İNSERT – İCRAÇI USER NAME 74](#_Toc40630137)

[DELETE – SİLMƏK QADAĞASI 75](#_Toc40630138)

[Satış Edən Zaman Bir Cədvələ Daxil Edilən Məlumat Əsasən Bazada Dəyişiklik. 75](#_Toc40630139)

[ Əlavələr 76](#_Toc40630140)

[Instead of 76](#_Toc40630141)

[Multiple actions Trigger 76](#_Toc40630142)

[Şərt funksiyalı Trigger 76](#_Toc40630143)

[Enabled-Disabled Trigger 76](#_Toc40630144)

[DƏRS 26. (ADO.NET) – DataGridView-CellClick { INSERT, UPDATE, DELETE, SELECT, SCALAR } 79](#_Toc40630145)

[ ADO.Net – INSERT (Connected) 79](#_Toc40630146)

[ADO.Net – INSERT (Daha düzgün forması) 81](#_Toc40630147)

[ ADO.Net – SELECT (Connected) 83](#_Toc40630148)

[ DataGridView – CellClick 83](#_Toc40630149)

[ ADO.Net – UPDATE (Connected) 84](#_Toc40630150)

[ ADO.Net – DELETE (Connected) 85](#_Toc40630151)

[ ADO.Net – SCALAR (Connected) 86](#_Toc40630152)

[DƏRS 27. (ADO.NET DISCONNECTED-ADAPTER) 87](#_Toc40630153)

[ ADO.Net – INSERT (DisConnected) 87](#_Toc40630154)

[ ADO.Net – SELECT-GET (DisConnected) 87](#_Toc40630155)

[ ADO.Net – UPDATE (DisConnected) 88](#_Toc40630156)

[ ADO.Net – DELETE (DisConnected) 88](#_Toc40630157)

[İki cədvəldə join daxil edərək məlumat mübadiləsi ilə INSERT, SELECT, DELETE, UPDATE və SCALARA 89](#_Toc40630158)

[ ADO.Net – INSERT (JOİN) 90](#_Toc40630159)

[ ADO.Net – SELECT (JOİN) 91](#_Toc40630160)

[ ADO.Net – UPDATE (JOİN) 91](#_Toc40630161)

[ ADO.Net – DELETE (JOİN) 92](#_Toc40630162)

[ ADO.Net – SCALAR (JOİN) 92](#_Toc40630163)

[ DataGridView-CellClick 92](#_Toc40630164)

[DƏRS 28. (ADO.NET -yekun) Buttons Command 93](#_Toc40630165)

[ Axdarılan dəyərə uyğun olaraq çəkilmiş məlumatları textbox-lara yazmaq. 93](#_Toc40630166)

[ Selected Row 94](#_Toc40630167)

[ Selected Cell 94](#_Toc40630168)

[ Current Row 94](#_Toc40630169)

[ ExecuteScalar deyerinin var tipine menimsetmek 95](#_Toc40630170)

[DƏRS 29. (Web Service – anlayışı, ASMX) 97](#_Toc40630171)

[ ASMX – toplama, vurma 98](#_Toc40630172)

[DƏRS 30. (WS-ASMX) ADO.Net – insert, select, update, delete {Prosedur ilə} 102](#_Toc40630173)

[ Prosedurlar 102](#_Toc40630174)

[SELECT 102](#_Toc40630175)

[INSERT 102](#_Toc40630176)

[UPDATE 102](#_Toc40630177)

[DELETE 102](#_Toc40630178)

[ Web Services hisəsi 103](#_Toc40630179)

[ Client hissəsi 107](#_Toc40630180)

[DƏRS 31. (WS-WCF) Cbar.az - XML 108](#_Toc40630181)

[ Mərkəzi bankın gündəlik valyuta məzənnəsinin çekilməsi 108](#_Toc40630182)

[ WCF – interface anlayışı 110](#_Toc40630183)

[DƏRS 32. (MVC-Model View Controller) 113](#_Toc40630184)

[DƏRS 33-34. (Master page – model class) 115](#_Toc40630185)

[ Master page 115](#_Toc40630186)

[ Model Class 117](#_Toc40630187)

[DƏRS 35. (Model class-kiçik nümunə) 121](#_Toc40630188)

[DƏRS 36. (ORM – Object Relational Mapping) LinqToSql 123](#_Toc40630189)

[ Linq To SQL 124](#_Toc40630190)

[DƏRS 37. (ORM – Code First) 126](#_Toc40630191)

[ CodeFirst Engin Demiroğ 128](#_Toc40630192)

[DƏRS 38. () 129](#_Toc40630193)

[DƏRS 39. (Object to XML, ExcellExport) 130](#_Toc40630194)

[ Excel Export 132](#_Toc40630195)

[ Excel Export (DataGridView -> Excel) 134](#_Toc40630196)

[Kurumsal Mimaride ASP.NET MVC Dersleri 135](#_Toc40630197)

[ Layihələndirmə 135](#_Toc40630198)

[ Textini tapşırığın hazırlanması 136](#_Toc40630199)

[ Çəkilən məlumatların ekranda səhifələr halında göstərilməsi 145](#_Toc40630200)

[ Səhifələrin nömrələnməsinin təmin edilməsi. 147](#_Toc40630201)

[ Sol paneldə “Category” cədvəlini listləmək və category əsasında səhifələri nömrələmək 148](#_Toc40630202)

[ Məlumatların yanında Add buttonu yaratmaq və səbətə əlavəetmə prosesi 153](#_Toc40630203)

[ Sifariş et düyməsi 157](#_Toc40630204)

[ Header hissəsindəki səbət haqqında məlumatlar 159](#_Toc40630205)

[ ModelBinding anlayışı və nümumə. 160](#_Toc40630206)

[ AdminPage əlavə edilməsi 162](#_Toc40630207)

[ AdminPage səhifəsindəki imkanlara giriş məhdudiyyəti 165](#_Toc40630208)

[ Sistemi Service üzərindən host etmək. 168](#_Toc40630209)

[Klavye qısa yolları 170](#_Toc40630210)

[Dərs 1. İlkin tanışlıq və ilkin anlayışlar 171](#_Toc40630211)

[Dərs 2. Css - class, style, id anlayışları 173](#_Toc40630212)

[ Səhifə daxili və digər səhifələrə istinad 175](#_Toc40630213)

[ Başlıqlar 176](#_Toc40630214)

[ Səhifəni ayıran horizontal xətt 176](#_Toc40630215)

[ Text daxilində sistat 176](#_Toc40630216)

[ Code daxilindəki bütün icraları göstərmək 177](#_Toc40630217)

[ Text formaları 177](#_Toc40630218)

[PROYEKT HAZIRLANMASI ZAMANI YARANAN ÇƏTİNLİKLƏR 179](#_Toc40630219)

[ Formlar arası keçid və məlumat mübadiləsi. 179](#_Toc40630220)

[ Kilit ekran ve s. Progrsmlarda Buttonların “button\_click” events-ləri 179](#_Toc40630221)

[ Parol yazılan zaman simvollar \*\* görülməsi 180](#_Toc40630222)

[ Escape Sequance 180](#_Toc40630223)

[ Klaviaturadan ENTER və ESCAPE düymələrinə basan zaman işləyən button - ları təyin etmək. 180](#_Toc40630224)

[I yol 180](#_Toc40630225)

[II yol 180](#_Toc40630226)

[ EYNI ANDA TOPLU INSERT 181](#_Toc40630227)

[ SELECT ILE INSERT 181](#_Toc40630228)

[ Dama Taxtası 181](#_Toc40630229)

[INTERVIEW SSENARI 182](#_Toc40630230)