

# VFP Öğreniyorum

**Hazırlayan:**  
Tarkan Haser

Haser Yazılım

[www.haser.com](http://www.haser.com)

Revizyon 1.0

# İçindekiler

1. VFP'ya Başlarken.....	3
1.1. Standartlarınız Olsun .....	3
1.1.1. Çalışma Klasörü.....	3
1.1.2. VFP Ayarları.....	4
1.1.2.1. Seçenekler .....	4
1.1.2.2. Makrolar.....	4
1.1.3. Kodlama Standartları .....	5
1.1.3.1. Yazma Standardı.....	5
1.1.3.2. Tasarım Standardı .....	13
1.1.3.3. Objeler Standardı .....	13
2. Projeye Başlarken .....	14
2.1. Analiz Çalışması .....	14
2.2. İlk Ayarların Yapılması .....	15
2.2.1. Veritabanı yaklaşımı .....	15
2.2.1.1. Lokal Veritabanı .....	15
2.2.1.2. Genel Veritabanı .....	15
2.2.2. Programın İlk Kodlarının Oluşturulması .....	16
2.3. Class'lara İlk Adım.....	22
2.3.1. Class Ne Demek?.....	22
2.3.2. İlk Class'lar.....	23
2.3.3. Class'dan Class.....	23
2.3.4. Class'ların Kullanılması .....	24
3. Veritabanına Başlarken .....	25
3.1. Tablo .....	25
3.1.1. Genel Yapı .....	25
3.1.2. Sahalar.....	26
3.1.3. Tablo Büyüklüğü .....	28
3.1.4. İndeks .....	28
3.1.5. Tablo Tipleri .....	30

# 1. VFP'ya Başlarken

## 1.1. Standartlarınız Olsun

Hepimiz yazılım yaparken iyi kötü kendimize özgü standartlar oluşturmuşuzdur. Ancak bu standartları yazılı hale getirmek bir kurallar silsilesi oluşturmak için uğraşmamışızdır. Hacer Yazılımda bir ekip çalışması yapmak zorunda olduğumuzdan yazdığımız kodların bir başka kişi tarafından anlaşılabilir ve devam ettirilebilir olması gerekmektedir. Aslında bu durum tek başına çalışanlar içinde geçerlidir. Hangimiz önceden yazmış olduğu kodun ne işe yaradığını anlamak için çaba harcamadı. Keşke buraya bir açıklama yazsaydım demedi.

Bu ve birçok önemli sebeplerden dolayı ekip olarak bir yazılım kuralı geliştirdik. Aslında piyasada bilinen birçok kural var ancak bizde kendimize özel kurallar geliştirdik ve geliştirmeye de devam ediyoruz. Burada bu kuralların tamamını yazamam ancak ana hatlarını sizlerle paylaşmak istiyorum.

Ayrıca oluşturacağınız kuralların kesinlikle yazılı olmasını tavsiye ederim. Yazdığımız kodların oluşturmuş olduğunuz kurallara uyup uymadığını kurallarınız oturana kadar kontrol edin. İlk başlarda bu size çok zor evet tekrarlıyorum çok zor gelecek sizi yavaşlatacak ancak lütfen yılmayın bu kurallara alışınca önceden nasıl yazıyordum diyeceksiniz.

Aşağıda bu standartları bulacaksınız.

### 1.1.1. Çalışma Klasörü

Projelerini takip edeceğiniz yöneteceğiniz klasörler oluşturun.

C:\

Projeler

A Projesi

B Projesi

Gibi.

En son oluşturduğunuz A Projesi, B Projesi klasörleriniz altına ise şu alt klasörleri oluşturabilirsiniz.

A Projesi

Bmp && Programınızda kullanacağınız resim dosyalarınızı koyacağınız

klasör

Dokuman && İlgili projeye ilgili dokümanlarınızı koyacağınız klasör

Form && Formlarınızı koyacağınız klasör

Menu && Menu dosyalarınızı koyacağınız klasör

Rapor && Rapor dosyalarınızı koyacağınız klasör

Setup && Programın kurulumu oluşturacağınız klasör

Vcx && Klâslerinizi koyacağınız klasör

VeriGenel && Çok kullanıcı programlarda kullanacağınız yani ana bilgisayarda yada başka bir bilgisayarda paylaşma açacağınız veritabanı yada veritabanlarını koyacağınız klasör

VeriLokal && Her bilgisayarın sadece kendisinin kullanacağı veritabanı ya da veritabanlarını koyacağınız klasör

## 1.1.2. VFP Ayarları

Çalışmanızı kolaylaştıracak ayarlarınızı yapmalısınız. Bu konuda size şunları tavsiye edebilirim.

### 1.1.2.1. Seçenekler

Menüden Tools --> Options seçiniz. Açılan pencerede Data sayfasına geliniz. Ignore deleted records seçeneğinizi işaretleyiniz. Bu sayede tablolarımızdaki silinmiş kayıtları devre dışı bırakmış olacaksınız. File Locations sayfasında Default Directory'yi çalışma yapacağınız projenin klasörünü seçiniz. Örneğin: C:\Projeler\A Projesi gibi.

Forms sayfasından ise objelerinizi yerleştirirken size yardımcı olacak olan grid ayarlarınızı değiştirebilirsiniz. Benim tavsiyem 6 pixele 6 pixel. Ayrıca yine bu sayfadan form tasarımının boyutlarını seçebilirsiniz. Bu sayede formunuzun ekran boyutlarından taşıp taşmadığını anlayabilirsiniz. Bu konuda biz 1024x768'i tercih ediyoruz. Ancak unutulmamalıdır ki yapılacak projeye göre bu durum değerlendirilmelidir. Bir form tasarımı yapmak önemli bir olaydır. Formlarınız ne kadar kullanışlı olursa o kadar tutulur. Bir 1024x768 ebatlarında form tasarlayabilmek için sizde ekran ayarlarınızın yüksek çözünürlükte olması iyi olur. Ben ekran çözünürlüğümü 1680x1050 kullanıyorum. Tabii ki bu çözünürlük için 15" den daha büyük ekran tavsiye ediyorum. Form tasarımlarından bahsederken unutulmaması gereken en önemli nokta formun yüksekliğinin ekran çözünürlüğünün yüksekliğinden daha küçük tasarlanması gerektiğidir. Windows'un başlat çubuğu, formun başlığı, screen'daki menü ve durum çubuğu ekranda bir hayli yer kaplayacaktır. Tabi programınızda bir de araç çubukları kullanırsanız yüksekliğiniz bir hayli düşecektir.

Regional sayfasında Use System Settings'i işaretlerseniz bilgisayarınızın tarih, saat ve numaralarla ilgili ayarlarını VFP'da kullanacaktır.

Bu yapmış olduğunuz ayarları Set As Default düğmesine basarak kaydedin. Böylece her VFP'yu açtığınızda bu ayarları yapmak zorunda kalmazsınız.

### 1.1.2.2. Makrolar

Kod yazarken hazır makrolar oluşturabilirsiniz. Bunlar sizin işinizi bir hayli kolaylaştırır.

Makro oluşturmak için;

Tools --> Macros seçin. (Bu pencerede işlemlerinizi kayıt düğmesini var ancak ben burada sizlere hazır makroları tanıtacağım)

New Düğmesine basınız.

**Seçili satırları 'Comment' yapma (yazının başına \*\* koyma)**

Defined Key Bölümde F12 düğmesine basınız. Böylece F12 düğmesine bir makro atamış olacağız.

Macro Contents kısmına {SHIFT+F10}m yazınız.

OK Düğmesinden sonra Set Default düğmesine basınız. Böylece bu makrolarınız her zaman geçerli olur.

**Seçili satırları 'UnComment' yapma (yazının başındaki \*\* işaretini kaldırma)**

Defined Key Bölümde F11 düğmesine basınız. Böylece F11 düğmesine bir macro atamış olacağız.

Macro Contents kısmına {SHIFT+F10}n yazınız.

OK Düğmesinden sonra Set Default düğmesine basınız. Böylece bu makrolarınız her zaman geçerli olur.

**Object List penceresini açma (Form tasarımında kod yazarken Object List açma)**

Defined Key Bölümde F10 düğmesine basınız. Böylece F10 düğmesine bir macro atamış olacağız.

Macro Contents kısmına {SHIFT+F10}j yazınız.

OK Düğmesinden sonra Set Default düğmesine basınız. Böylece bu macrolarınız her zaman geçerli olur.

### **Projenizi çalıştırma**

Programınız eğer ana.prg'den çalışıyorsa;

Defined Key Bölümde F9 düğmesine basınız. Böylece F9 düğmesine bir macro atamış olacağız.

Macro Contents Kısımına {CTRL+F2}do{SPACEBAR}ana.prg{ENTER} yazınız.

OK Düğmesinden sonra Set Default düğmesine basınız. Böylece bu macrolarınız her zaman geçerli olur.

## **1.1.3. Kodlama Standartları**

Kodlama standartları tüm standartlar arasında en önemlisidir. Bu konuyla ilgili olarak kabaca şunları söyleyebilirim. Burada söyleyeceklerim bizim standartlarımızın temelleridir.

### **1.1.3.1. Yazma Standardı**

- **Komut yazımlarında her kelimenin baş harfi büyük yazılmak zorundadır.**

Örneğin: THISFORM yerine ThisForm. Bu kullanım için IntelliSense Manager'in ayarlanması gerekmektedir. Tools-->IntelliSense Manager'i seçin. Açılan menüden FoxCode Default'u No auto-expansion seçiniz. Functions ve Commands seçeneklerini ise Use FoxCode Default olarak seçiniz.

- **Lokal kullanılacak bütün değişkenler Local komutunu ile tanımlanmak zorundadır.**

Bu noktada şöyle düşünebilirsiniz yazmış olduğun procedure içinde hiçbir yere dallanma yok ve eminim ki bu değişkenler başka bir yerde de kullanılmamaktadır ve kullanılmayacak o yüzden Local komutu ile uğraşmamın ne anlamı var ki. Evet, belki de bunu düşündüğünüz an için bu durum doğrudur. Ancak aradan zaman geçip de kodu unuttuktan sonra ya da bir başkası sizin yazmış olduğunuz kodu değiştirdiğinde yada kodda bir dallanma yaptığınızda local olarak tanımlamadığınız değişkenler sorun yaşama olasılığınız yükselir. Böyle bir durumda sorunu test aşamasında yakalama olasılığınız ise çok düşüktür. Ancak kullanıcılardan size verilerinde bir tuhafılık olduğu bilgisi geldiğinde uzun bir araştırmadan sonra hatayı bulma olasılığınız var. Bu durumu özellikle yeni başlayanlar için detaylı açıklamak istiyorum.

Local ya da public olarak tanımlanmayan (bir tanım yapılmadan kullanılan) tüm değişkenler Private değişken tipinde oluşturulur. Bu şu anlama gelmektedir. Bu değişken tanımlandığı procedure ve procedureden çağrılan tüm alt procedure'lerde kullanılabilir demektir. Bu işlem bilinç bir şekilde yapılmadığı takdirde büyük sorunlara yol açabilir. Bu durumu bir örnekle açıklamak gerekirse; Procedureleriniz içinde kullandığınız bütün değişkenlerini kesinlikle tanıtınız. Local olarak tanımlanmış değişkenler sadece o procedure için geçerli olurlar. Bunu bir örnekle açıklayalım.

#### **Local'siz**

```
*!* *****
```

```
m.lnI = 1
```

```
m.lnI = m.lnI + 1
```

```
MessageBox(Transform(m.lnI)) && 2 görünür
```

```
Do Deger
```

```
MessageBox(Transform(m.lnI)) && 5 görünür
```

```
Return
```

```
Procedure Deger
m.lnI = 1
For m.ln = 1 To 5
    Wait Window Transform(m.lnI) NoWait
EndFor
Return
```

\*!\* \*\*\*\*\*

Yukarıdaki örnekte 1.procedurede tanımlanan lnI değişkeni Deger alt procedureune gittiğinde değeri 5 olarak değişiyor. Alında bu durum istenmeyen bir şeydir. Çünkü 1.procedure ile 2. procedure arasındaki ilişkide lnI değişkenin bir önemi yok.

### Local'li

\*!\* \*\*\*\*\*

```
Local lnI
m.lnI = 1
m.lnI = m.lnI + 1
MessageBox(Transform(m.lnI)) && 2 görünür
Do Deger
MessageBox(Transform(m.lnI)) && 2 görünür
Return
```

```
Procedure Deger
Local lnI
m.lnI = 1
For m.ln = 1 To 5
    Wait Window Transform(m.lnI) NoWait
EndFor
Return
```

\*!\* \*\*\*\*\*

2. örnekte Local komutu ile bu sorun ortadan kaldırılmıştır.

- **Public değişken kullanımı sadece ana (yani programın başlayacağı) kodlama sırasında kullanılmak zorundadır. Hiçbir şekilde formlarda yada benzeri kodlarda public değişken kullanılmayacaktır.**

Bu durumda garip görünebilir ancak bu da en az local değişken kullanımı kuralı kadar önemlidir. Bu durumda çok önemli hatalara yol açabilir. Yine yeni başlayan arkadaşlar için biraz daha detaylı bir açıklama yapalım. Düşünülebilir ki formlarda bir procedureden bir başka procedure bilgi aktarmak ya da formda istediğim zaman kullanacağım bir değişkene ihtiyaç duyulur. Bu gibi durumlarda public değişken dışında ne kullanılabilir ki. Bunu bir örnekle açıklayalım. Tasarlayacağımız formda ad soyad bilgilerinin girildiği iki textbox olsun ve yeni, sil, yükle ve kaydet diye düğmelerimiz olsun. Eğer kullanıcı yükle düğmesinden bir kayıt seçip yüklerse kaydet düğmesine bastığımızda değiştirme yeni düğmesini seçip kayıt girdiyse kayıt ekleme işlemini yapmalıyız. Bu durumda elimizde öyle bir değişken olmalı ki yapılan işlemin kayıt eklememi yoksa değiştirmemi olduğu bilelim. Kaydet düğmesine bastığında işlemimizi ona göre yapalım. Bu işi public bir değişken çözebilir ama bu durum önceden bahsettiğimiz sebeplerden dolayı tehlikelidir. Bunun için formumuza bir özellik (property) eklememiz gerekmektedir. Bunun için menüden Form-->New Property'yi seçiniz. Açılan pencereye IYeniMi yazıp Add düğmesine basın. Artık elimizde yeni bir özellik var. Bu durumda Yeni düğmesinin Click eventine

ThisForm.IYeniMi = .T. Yükle Düğmesinin Click evetine ThisForm.IYeniMi = .F. satırlarını ekleyebiliriz. Şimdi Kaydet düğmesinin Click eventinde

```
If ThisForm.IYeniMi
    ThisForm.YeniKayit
Else
    ThisForm.KayitDegistir
EndIf
```

Gibi bir kod yazabiliriz. Böylece public değişken kullanmaktan kurtulmuş oluruz.

- **Kod hizalama yapılmak zorundadır.**

Yazılan kodun rahat okunabilmesi için kod hizalama çok önemlidir. If, For, Do Case ve atama işlemlerinde TAB karakteri kullanılmalıdır. Bu durumu bir örnekle açıklayalım.

```
LParameters tnSecim As Integer
Local lnBaslangic As Integer, lnBitis As Integer, lnAraDegerSonucu As Integer
m.lnBaslangic = 3
m.lnBitis = 5
m.lnAraDegerSonucu = 0
```

\*!\* Yukarıda ki atamaya dikkat edecek olursanız = işaretleri hizalanmıştır. Bu durum kodun kolay okunmasını sağlar.

```
Do Case
    Case m.tnSecim = 1
        Replace cAd With "Tarkan",
                cSoyad With "Haser" In Kisiler
```

\*!\* Replace komutunda cAd, cSoyad saha isimleri aynı hizadan başladığı gibi With komutları da aynı hizadan başlamaktadır.

\*!\* Ayrıca her saha için bir alt satıra geçilmiştir.

Bu ve benzeri durumlar gereksiz gibi görünse de bu tarz bir yazıma alışınca eski yazımlarınız size çok karışık gelecektir. Lütfen bu noktayı önemseyiniz. Sizlerde replace gibi diğer benzer komutların yazılma standardını oluşturabilirsiniz.

- **With komutunun kullanım olabildiğince kullanma zorunluluğu.**

With komutunu kodlamanızın daha kolay anlaşılır ve yazma işini kolaylaştırır. Ancak en önemlisi kodunuzun hızlandırır. With parantezine alınmış her kod parçası parantezde tanımlanan kısmı sadece bir kere hafızaya alır. Bunu bir örnekle açıklayalım.

```
ThisForm.pgfAna.pagGenel.txtAd.Value = "Tarkan"
ThisForm.pgfAna.pagGenel.txtSoyad.Value = "Haser"
ThisForm.pgfAna.pagGenel.txtTelefon.Value = "(312) 479 10 50"
ThisForm.pgfAna.pagGenel.txtFaks.Value = "(312) 479 10 60"
ThisForm.pgfAna.pagGenel.txtSehir.Value = "Ankara"
```

Kodunda ThisForm.pgfAna.pagGenel objesi her satırda hafızaya yükleniyor ve komut bittiğinde siliniyor. Sonra bir sonraki satırda da benzer işlem yapılıyor. Oysaki bu sizin kodunuzu yavaşlatan bir durumdur. O yüzden bu kodu şu şekilde yazabiliriz.

```
With ThisForm.pgfAna.pagGenel
    .txtAd.Value = "Tarkan"
    .txtSoyad.Value = "Haser"
    .txtTelefon.Value = "(312) 479 10 50"
    .txtFaks.Value = "(312) 479 10 60"
    .txtSehir.Value = "Ankara"
```

```
EndWith
```

Yukarıdaki durumda with parantezinde ThisForm.pgfAna.pagGenel objesi hafızaya bir kere alınır ve paranter EndWith ile kapanan kadar hafızada kalır. Böylece gereksiz kod işletilmemiş olur.

- **Değişken ve Obje isimleri verilerken ön ek kullanılmalı ve anlaşılır isimler seçilmeli. İki kelimedenden oluşan değişken isimleri içinse konuşma diline uygun verilmedir.**

Ön ekten bahsetmeden önce değişken isimlerinin anlaşılır olmasından ve konuşma diline uygunluğundan bahsedelim. Değişkenlerin anlaşılır olması hem sizin kodu sonradan okuduğunuzda hem de başkasının kodu rahat anlamasına yardımcı olur. Örneğin başlangıç tarihi ile bitiş tarihini tuttuğunuz iki değişken için btarih starih gibi iki değişken ismi yerine BaslangicTarihi, BitisTarihi şekliden kullanın. Başlangıç ve tarih kelimelerini birleştirirken Baslangictarih şeklinde yazmayın BaslangicTarihi şeklinde sonuna "i" karakterini de ekleyin ki hem daha manalı hem de ezbere yazımlarda nasıl yazdığımızı kolay hatırlayabilesiniz.

Ön eke gelecek olursak; bununla ilgili birkaç yöntem var ancak biz VFP'nun önerdiği yöntemi kullanıyoruz. Bunları VFP'nun yardımıyla "Variable Naming Conventions" ve "Object Naming Conventions" başlıkları altında bulabilirsiniz. Burada ben biraz yazayım:

#### **Değişken Tanımları**

Temel Format aşağıdaki gibidir;

*[Türü]TipiAdi*

#### **Anlamları**

*Türü*

<b>Önek</b>	<b>Açıklama</b>	<b>Örnek</b>
l	Local	lnSayac
p	Private (varsayılan)	pnDurum
g	Public (global)	gnEskiKayitNo
t	Parameter	tnKayitNo

*Tipi*

<b>Tipi</b>	<b>Açıklama</b>	<b>Örnek</b>
a	Array	aAylar
c	Character	cSoyAdi
y	Currency	ySuankiDeger
d	Date	dYasGunu
t	Datetime	tSonKullanım
b	Double	bDeger
f	Float	fIlgi
l	Logical	lIsaret
n	Numeric	nSayac
o	Object	oGorevli
u	Unknown	uDonusDegeri

## Örnek Tanımları

Temel format aşağıdaki gibidir;

*ÖrnekAdı*

Nesneler için tavsiye edilen örnekler aşağıdaki gibidir.

Örnek	Nesne	Örnek
Acd	ActiveDoc	acdAnaSayfa
Chk	CheckBox	chkSaltOkunur
Cbo	ComboBox	cboIngilizce
Cmd	CommandButton	cmdVazgeç
Cmg	CommandGroup	cmgSecenekler
Cnt	Container	cntListe
Ctl	Control	ctlDosyaListesi
<Kullanici tanimli>	Custom	Kullanici tanimli
Edt	EditBox	edtVeriAlani
Frm	Form	frmDosyaAc
frs	FormSet	frsVeriAlani
grd	Grid	grdFiyatlar
grc	Column	grcSuAnkiFiyat
grh	Header	grhToplamEnvanter
hpl	HyperLink	hplAnaSayfa
img	Image	imgIcon
lbl	Label	lblYardimMesaji
lin	Line	linYatay
lst	ListBox	lstKodlar
olb	OLEBoundControl	olbObject1
ole	OLE	oleObject1
opt	OptionButton	optIngilizce
opg	OptionGroup	opgTipi
pag	Page	pagVeriGuncelle
pgf	PageFrame	pgfSol
prj	ProjectHook	prjBuildAll
sep	Separator	sepAracBolum1
shp	Shape	shpDaire
spn	Spinner	spnDegerler
txt	TextBox	txtVeriAl
tmr	Timer	tmrAlarm
tbr	ToolBar	tbrRapor

- **Değişken isimlerini kullanırken kesinlik m. ile kullanınız.**

m. ön eki hafıza değişkenlerinin kullanıldığını anlatan bir ifadedir. m.lcAd şeklinde kullandığımız bir satır onun bir değişken olduğunu kesin olarak belirtir. m.lcAd yerine lcAd şeklinde bir kullanım onun bir tablo saha başlığı da olabileceği anlamında da gelmektedir. Bu durumda Foxpro yazılan değişkenin saha değişkeni mi olduğuna bakar sonra değişken olup olmadığına bakar. Aynı isimli bir saha kullanıldıysa bir karışıklığa ve zaman kaybına neden olur. Bu tür hataların bulunması ise bir hayli güçtür. O yüzden tüm değişkenler m. ön eki ile kullanılmalıdır.

- **Kod anlaşılabilirliği için gerekli olan her yere uzun kod açıklamaları eklenmelidir.**

Önemli kodları yazdığımız, karışık ve sıkıntılı kodların tamamına uzunun uzun açıklamalar yazılmalıdır. Bu durum bende kabul ediyorum ki çok sıkıcı ancak gerekli :)

- **Kodun optimize edilmesi gerekmektedir.**

Kodlarımızı olabildiğince kısa ve anlaşılır yazmakta fayda vardır. Tekrarlanan kodları kesinlikle ve kesinlikle tek bir procedure altına taşımamızdır. Bunu basit bir örnekle açıklamaya çalışayım. (Bu anlatımı zor bir kodu birçok durum ve örnek var ancak size bir örnek verebileceğim.)

Bir kaydet düğmesi tasarladığımızı varsayalım. Bu kaydet düğmesi hem yeni kayıt hem de eski kaydı değiştiren bir kod içersin.

With ThisForm

```
If Empty(.txtAd.Value)
```

```
    MessageBox("Lütfen ad giriniz!", 48, "Uyarı")
```

```
    .txtAd.SetFocus
```

```
    Return .F.
```

```
EndIf
```

```
If Empty(.txtSoyad.Value)
```

```
    MessageBox("Lütfen soyadı giriniz!", 48, "Uyarı")
```

```
    .txtSoyad.SetFocus
```

```
    Return .F.
```

```
EndIf
```

```
If .YeniMi
```

```
    Local lcAd As String, lcSoyad As String
```

```
    m.lcAd = AllTrim(.txtAd.Value)
```

```
    If Seek(m.lcAd, "Kisiler", "Ad")
```

```
        MessageBox("Daha önceden aynı isimde bir kayıt mevcuttur!" +
```

```
Chr(13) + ;
```

```
        "Lütfen kontrol ediniz!", 48, "Uyarı")
```

```
        .txtAd.SetFocus
```

```
        Return .F.
```

```
    EndIf
```

```
    m.lcSoyad = AllTrim(.txtSoyad.Value)
```

```
    Insert Into Kisiler(cAd, cSoyad) Values(m.lcAd, m.lcSoyad)
```

```
Else
```

```
    *!* Değiştir komutunda adın kullanıcı tarafından değiştirilmediğini
```

```
varsayıyorum.
```

```
    Local lcAd As String, lcSoyad As String
```

```
    m.lcAd = AllTrim(.txtAd.Value)
```

```
    If !Seek(m.lcAd, "Kisiler", "Ad")
```

```
        MessageBox("İlgili kayıt bulunamadı!" + Chr(13) + ;
```

```
        "Bu kayıt bir başka kullanıcı tarafından silinmiş
```

```
        olabilir!" + Chr(13) + ;
```

```
        "Lütfen kontrol ediniz!", 48, "Uyarı")
```

```
        Return .F.
```

```
    EndIf
```

```
    Replace cSoyad With m.lcSoyad In Kisiler
```

```
EndIf
```

```
    MessageBox("İşlem başarılı!", 64, "Bilgi")
```

```
EndWith
```

Gibi bir kod yerine;

```

With ThisForm
    If .Kontrol()
        If .YeniMi
            .YeniKayit
        Else
            .KayitDegistir
        EndIf
    EndIf
EndWith

```

Yazılabilir. Kontrol, YeniKayit ve KayitDegistir adında üç tane yeni method tanımlayıp ilgili kodları bunların içine yazmalıyız. Forma yeni method tanımlamak için Form-->New Method seçeneğini seçip açılan pencereye method isimlerini yazıp add düğmesine basınız. Bu eklemiş olduğunuz yeni methodları properties penceresinde görebilirsiniz. İlgili methodların içine şunları yazabilirsiniz.

Kontrol Methodu

```

With ThisForm
If Empty(.txtAd.Value)
    MessageBox("Lütfen ad giriniz!", 48, "Uyarı")
    .txtAd.SetFocus
    Return .F.
EndIf
If Empty(.txtSoyad.Value)
    MessageBox("Lütfen soyad giriniz!", 48, "Uyarı")
    .txtSoyad.SetFocus
    Return .F.
EndIf
Local lcAd As String
m.lcAd = AllTrim(.txtAd.Value)
If .YeniMi
    If Seek(m.lcAd, "Kisiler", "Ad")
        MessageBox("Daha önceden aynı isimde bir kayıt mevcuttur!" +
Chr(13) + ;
                    "Lütfen kontrol ediniz!", 48, "Uyarı")
        .txtAd.SetFocus
        Return .F.
    EndIf
Else
    ** Değiştir komutunda adın kullanı tarafından değiştirilmediğini
varsayıyorum.
    If !Seek(m.lcAd, "Kisiler", "Ad")
        MessageBox("İlgili kayıt bulunamadı!" + Chr(13) + ;
                    "Bu kayıt bir başka kullanıcı tarafından silinmiş
                    olabilir!" + Chr(13) + ;
                    "Lütfen kontrol ediniz!", 48, "Uyarı")
        Return .F.
    EndIf
EndIf
EndWith

```

YeniKayit Methodu

With ThisForm

```
Local lcAd As String, lcSoyad As String
```

```
m.lcAd = AllTrim(.txtAd.Value)
```

```
m.lcSoyad = AllTrim(.txtSoyad.Value)
```

```
Insert Into Kisiler(cAd, cSoyad) Values(m.lcAd, m.lcSoyad)
```

EndWith

KayitDegistir Methodu

With ThisForm

```
Local lcSoyad As String
```

```
m.lcSoyad = AllTrim(.txtSoyad.Value)
```

```
Replace cSoyad With m.lcSoyad In Kisiler
```

EndWith

- **Select komutunun gerekmedikçe kullanılmaması.**

Select TabloAdi şeklinde kullanımlarda o anki aktif tabloyu değiştirir. Bu sayede ilgili tablonun saha isimlerini direk yazarak kullanma şansımız vardır. Ancak biz bu kullanımı pek tavsiye etmiyoruz. Bizim için aktif tablonun ne olduğunun bir önemi olmamalı. Çünkü bir kodun bir yerinde istemeden bir şekilde aktif tablo değişebilir. Buda kodun çalışmamasına neden olabilir. Bunu bir örnekle açıklayalım.

With ThisForm

```
Select Kisiler
```

```
Do While !EOF()
```

```
    If .SiparisVarMi(cAd)
```

```
        Exit
```

```
    EndIf
```

```
    Skip
```

```
EndDo
```

EndWith

Şeklinde bir kodumuz olduğunu düşünelim. Bu kodun SiparisVarMi alt methodunda bir Siparisler tablosunda bir arama yaptığını düşünürsek bu işlem sırasında seçili tablo sipariş tablosu olursa hatalı bir kod yazmış oluruz.

SiparisVarMi Methodu

LParameters tcAd As String

Select Siparis

Return Seek(m.tcAd)

Bunun yerine şu şekilde yazabiliriz.

With ThisForm

```
Do While !EOF("Kisiler")
```

```
    If .SiparisVarMi(Kisiler.cAd)
```

```
        Exit
```

```
    EndIf
```

```
    Skip In Kisiler
```

```
EndDo
```

EndWith

Yukarıdaki kullanım hiçbir zaman aktif tablonun ne olduğunu önemsemediğinden aktif tablo değişti mi derdiniz yada aktif tabloyu şimdi değiştirsem ne olur gibi bir derdiniz olmaz.

Kod yazım standartları tabii çok detaylı olmalıdır. Ancak ilk başlangıç için bu temel bilgilerin yeterli olacağını düşünüyorum.

### **1.1.3.2. Tasarım Standardı**

Formlarınızı ve raporlarınızı tasarlarken hep bir bütünlük içinde bulunun. Örneğin formlarınızda bir yazı tipi seçin ve hep onu kullanın, düğme boylarınız hep aynı olsun. Önemli şeyleri belirteceğiniz label'ların renkleri hep aynı olsun. Ekran tasarımlarınız bir birine benzesin. Örneğin kapat düğmesi kullanıyorsanız hep sağ alt köşeyi tercih edebilirsiniz. Birinde sağ üst köşe birimde sağ alt köşe olmasın. Çok renkli ekran tasarımlarından uzak durun. Mümkünse projenizde kullanacağınız renk standartlarını belirleyiniz. Bu konu biraz göreceli olup kendi standartlarınızı rahatlıkla oluşturabilirsiniz.

### **1.1.3.3. Obje Standardı**

Kullandığınız her türlü objenin (nesne) form, label, textbox, grid, commandbutton, vs. gibi objeleri kendi standardınıza getirmelisiniz. Bunun için klasları kullanabilirsiniz. Ancak bu yazıda bu kısımda klaslardan bahsetmek pek doğru olmaz. Ancak şunu belirteyim ki biz çok uzun zamandır Foxpro'nun standart objelerini kullanmıyor kendi klaslarımızı kullanıyoruz. Bunun inanılmaz büyük bir faydası var. O kadar rahat ve esneğiz ki bu klasları kullanmadığımız dönemler için çok üzülmişim.

## 2. Projeye Başlarken

### 2.1. Analiz Çalışması

Yeni bir projeye başlarken kodlamadan uzak ama programcılığın en önemli noktası analiz çalışmasıdır. Analiz çalışması yapılmadan başlanılan her program dümeni bozuk bir gemi gibi hedefe ilerlerken sürekli yön değiştirir.

Yapacağınız proje müşteriniz için özel bir çalışma olabilir. O zaman müşterinin neler istediğini, programdan ne beklediğini çok iyi analiz etmek lazım. Müşterinizle gidip konuştunuz. Konuşma sırasında onu zorlayın ne bekliyor iyice anlayın. Yapılacak program, müşteri için önemli bir sistem oluşturabilir. Bu durumda programın sihirli bir değnek olmadığını programla beraber sistemi kullanacak kişilerin de programı desteklemesi gerektiğini veri girişlerinin sağlıklı olması gerektiğini vurgulayın. Sadece bir kişi ile görüşerek analiz çalışması yapmayın. Programı kullanacak kişilerle de görüşünün. Onlarında sorunlarını beklentilerini öğrenin.

Analiz sonunda hiç sıkılmadan bir rapor hazırlayın. Rapor iki kısımdan oluşmalıdır. Bir müşteriye verilecek olan rapor iki kendimizin kullanacağı rapor. Müşteriye vereceğimiz raporda sistem akış diyagramları, yapılacak formların kaba ekran tasarımları ve bu formların nasıl çalışacağı (en önemlisi kayıt ekleme, düzenleme ve silme koşulları). Bu hazırlamış olduğunuz müşteri raporunu özel bir toplantı ile müşteriye anlatın. Sorun ve sıkıntıları konuşun. Açık ve net olun.

Sistem akışı ve programla ilgili konular kadar programın çalışacağı bilgisayarlar ve bilgisayar ağ yapısı da önemlidir. Programınızın çalışacağı minimum özellikleri belirleyin. Bu hem terminal bilgisayarlar hem ana bilgisayarlar hem de ağ yapısı olmalıdır. Bunları belirlerken bilgisayarın hafızasından ekran çözünürlüğüne, ana bilgisayarın domanin yapısına kadar yazılı olarak aynı raporda müşteriye sunun. Tabi ki müşterinizin bu konuyu da iyice anlamasını sağlayın.

Müşterinizle kesinlikle ekinde hazırladığınız rapor olan bir sözleşme imzalayın.

Kendinize hazırlayacağınız rapor; veritabanı yapısal analizi, yapılacak olan formların çalışma mantığı ve ekran tasarımları, tüm programın sistem akış diyagramları ki dikkat edin diyagramı yazıp tek bir diyagram kastetmiyorum detaylı bir şekilde bir çok diyagram hazırlayın ve en önemlisi iş planını belirleyin.

İş planınızı gün gün hazırlayın. Hangi gün ne iş yapılacak ve ne kadar zamanda yapılacak ve kimin tarafından yapılacak? Bu iş planına projenin ara test aşamalarını ve proje sonunda tam test aşaması ile bu sıkıntıları çözecek zaman planlamasını da eklemeyi unutmayın. İş planınızda fazla mesai planlamayın. Günlük çalışma saatlerine uyun. İş planınıza gerçekleşen zamanları yazın. Erken ya da geç bitirimlerde neden olduğunu not düşün. Erken yada geç bitirmeler bir hatalı plan yapıldığını gösterir. Bu durumlardan ders alıp bir sonraki proje planlamasını ona göre yapın.

**PROJE PLANLAMASIZ ÇALIŞMAYIN!..**

Yukarıda bahsetmiş olduğum analiz çalışması sizlere büyük bir külfet gibi görünebilir. Ancak bu işlemten sonra projenizin yarısını bitmiş sayabilirsiniz. Projenizi bitirdiniz ve teslim ediyorsunuz. Müşterinizle; ben böyle istemedim, şunu yapmamışsın yada bunu neden böyle yaptın gibi konuşmalar olmasını istemiyorsanız, yada proje tesliminden sonra olabilecek her türlü sıkıntı için bu rapor size kaynak olacaktır.

Şu bir gerçek ki proje teslimlerinde her zaman müşterinizin fazladan istekleri ve talepleri olabilir. Bu istekler sizin için çok basit olabileceği gibi zorda olabilir. Bunları yapıp yapmamak size kalmıştır. Ancak projenin sıhhatli yürüebilmesi için mantıklı en azından çok zaman almayacak istekleri yapın ancak müşterinizden bu istekleri yazılı ve imzalı olarak alın.

Son olarak tekrar vurgulamak istiyorum, bu tür analiz çalışması 1 gün, 1 hafta yada 2 hafta sürebilir bu projenin büyüklüğüne göre değişir ancak kesinlikle böyle bir çalışma yapılmalıdır. Lütfen bunu es geçmeyiniz

## **2.2. İlk Ayarların Yapılması**

### **2.2.1. Veritabanı yaklaşımı**

Programınız tek veya çok kullanıcıli olabilir ancak hep çok kullanıcıliymiş gibi tasarım yapmanızı tavsiye ederim. Çok kullanıcıli bir tasarımda, veritabanının ana bilgisayarda tutulması gerekir ki tüm kullanıcılar ana bilgisayarda paylaşım açılmış olan veritabanına bağlanabilsinler. Bu noktada konuyu biraz daha detaylandıralım. Tasarım sırasında herkesin kullanacağı bir veritabanına ihtiyaç duyulduğu gibi sadece o kullanıcının ihtiyaç duyabileceği veritabanı da olabilir. Şöyle ki o bilgisayarda yapılan ayarların saklanabileceği bir veritabanı olabilir. (Bu ayarlar bir ini dosyasında da tutulabilir ama bana göre veritabanından okumak ve yazmak çok daha kolay.)

Bu noktada anlaşılıyor ki en az iki veritabanına ihtiyacımız var. Haseer Yazılım olarak biz bu iki veritabanının ismini Genel ve Lokal olarak sabitledik Bundan sonraki anlatımlarda ben de hep bu isimleri kullanacağım.

#### **2.2.1.1. Lokal Veritabanı**

Bu veritabanının amacı lokalde tutulması gereken ayarların saklanacağı veritabanıdır. Ben tabi ki burada minimum gereken bilgileri belirteceğim. En önemlisi ana bilgisayarda paylaşım açılan veritabanının adresinin saklanacağı bilgidir. Bunun için Bölüm 01 - VFP Başlarken 1.1.1 Çalışma Klasöründe belirtmiş olduğum klasör yapısında VeriLokal klasörünün altına Lokal isminde bir veritabanı oluşturun. Bu veritabanına MakineAyar isminde bir tablo oluşturun ve tablo yapısı şu şekilde olsun. Saha Adı : mVeriYolu Saha Tipi M (Memo) Oluşan bu tabloya tek bir boş kayıt ekleyin. (Daha sonra bilgi gireceğiz)

#### **2.2.1.2. Genel Veritabanı**

VeriGenel klasörünün altına Genel adında bir veritabanı oluşturun. Bu veritabanının tabloları ise uygulamanıza bağlı olarak değişecektir. Bunu tabi ki sizler belirleyeceksiniz.

## 2.2.2. Programın İlk Kodlarının Oluşturulması

İlk kodlamadan önce tabi ki yeni bir proje oluşturmalısınız. Bunun için File-->New... seçiniz açılan pencereden Project seçip New file düğmesine basınız. Açılan Create penceresinde yapacağınız projenin kök klasörüne projenin adını vererek kaydedin. Şimdi karşınızda projenizi rahatlıkla yönetebileceğiniz bir Project Manager penceresi açılmıştır.

Bu pencereye önceden oluşturduğumuz Genel ve Lokal veritabanlarını ekleyin. Data sayfasında Databases'i seçip Genel ve Lokal veritabanlarını ekleyin. Şimdi ilk kodlama için hazırız. Code sayfasında Programs'ı seçip New... düğmesine basınız. VFP'da oluşturacağınız programın ilk satırlarını bu program code sayfasına yazacağız. Oluşturduğunuz exe çalıştığında ilk bu dosya içindeki kodları işletecektir. O yüzden bu prg dosyasına biz "Ana" ismini verdik. Bu Ana.prg dosyasını yine projenizin kök klasörüne kaydetmelisiniz.

Şimdi bu Ana.prg dosyası içindeki ilk satırları açıklamaları ile yazalım.

```
!* Ana.prg dosyasının başı

!* Set Escape komutu program çalışırken ESC tuşuna basıldığında
!* programın kırılıp kırılmamasını ayarlar.
!* Program exe'den çalışıyorsa kırılmasını istemeyiz.
!* Ancak programı VFP içinden çalıştırıyorsak kodlamamızda oluşabilecek
bir
!* hata nedeni ile programı kırmak isteriz. O zaman ESC tuşuna ihtiyaç
duyarız.
!* Bu yüzden bu ayarı otomatik seçimli olarak ayarladık.
!* Versiyon() fonksiyonu ile programın exe'den mi yoksa VFP içinden mi
!* çalışıp çalışmadığına bakıyoruz. Eğer exe ise ESC 'Off' konumuna
!* VFP içinden çalışıyorsa 'On' konumuna getiriyoruz.
If Versiyon(2) = 0
    Set Escape Off
Else
    Set Escape On
EndIf

!* Kullanıcıları özgür bırakın. Bunun ilk adımı kullanıcının ayarlarını
kullanmaktır.
!* Kullanıcının yapmış olduğu tarih, saat, basamak ve dijital sembolünü
sizde kendi
!* programınızda kullanın. Bunun için aşağıdaki komutu uygulamanız
yeterlidir.
Set SysFormats On

!* VFP'nun bazı komutları sonuçlarını aktif ekranda gösterir.
!* Bizce bu durum tasarım aşamasında işe yarayabilir ancak
!* exe'den çalışan bir program için bu pek istenmez.
!* Örneğin Talk 'On' konumunda iken SUM komutunu kullandığınızda aktif
ekrana sonucu
!* listeler. Bu sizin ekran tasarımınızın üzerine çıkan çirkin yazılar
demektir.
!* Bu yüzden Talk komutunu ağırlıklı olarak 'Off' konumunda kullanmanızı
tavsiye ederim.
!* Ancak ufak bir hatırlatmada yapmak istiyorum bazı komutların
işletilmesi uzun sürebilir.
!* Örneğin bir SQL Select komutu 5-10 saniye civarında sürebilir. Bu
durumda ekrana
!* bir işlem ilerleyiş çubuğu (system progress bar) çıkmasını
isteyebilirsiniz.
```

\*!\* Bu durumda Talk komutunu 'On' yapmalısınız. Biz bu özel durumlar gerçekleştiğinde  
\*!\* ilk önce 'On' işlemden sonrada 'Off' hale getiriyoruz. Ancak varsayılan olarak hep  
\*!\* 'Off' kullanıyoruz. Talk'u etkileyen komutları VFP'nun yardım menüsünde bulabilirsiniz.  
Set Talk Off

\*!\* Foxpro'nun bir tablosundan kayıt sildiğinizde o kayıt gerçek anlamda tablodan silinmez.  
\*!\* Sadece silindiğini belirten bir işaret konur. Bu işareti Browse penceresinde ve Grid'de  
\*!\* sağ taraftaki ince sütunda görebilirsiniz. Eğer Deleted 'Off' konumunda ise silinmiş  
\*!\* kayıtları tablo içinde görürsünüz ancak bu sağ taraftaki ince sütunda silinmiş kayıtları  
\*!\* için siyah görünür. Kaydı silmek için bu sütunu kullanabileceğiniz gibi silinmiş kaydı  
\*!\* geri almak içinde kullanabilirsiniz. Bunun ilgili kaydın bu sütununa tıklayabilirsiniz.  
\*!\* Özetle silinen kayıtlar özel komutlar uygulanıncaya kadar tablo içinde saklı kalır.  
\*!\* İşte bu kayıtlar üzerinde genelde işlem yapılmaz o yüzden görmemiz ve kullanmamız  
\*!\* pek gerekmeyecektir. (Orta ve ileri düzeylerde bu kayıtlar kullanılacaktır ama kullanmak  
\*!\* istediğimizde Deleted 'Off' 'On' yaparak işimizi çözebiliriz. Biz varsayılan olarak  
\*!\* 'On' kullanıyoruz.  
Set Deleted On

\*!\* Programımızın çok kullanıcılı olmasına karar vermiştik. Bu yüzden kullanacağımız tabloları  
\*!\* kişiye özel değil de birden fazla kişi kullanacağından paylaşımaya uygun olarak açmalıyız.  
\*!\* Aşağıdaki komut kişiye özel kullanımı kapatan komuttur. Bu durumu kapatarak çok kullanıcı  
\*!\* program ayarlarına giriş yapmış oluruz. Ancak unutmayın ki tablolarınızı yeniden yapılandırmak  
\*!\* indekslemek gibi işlemlerde kişiye özel komumun açık olması gerekir. Yada tablonuzu açarken  
\*!\* özel olarak kişiye özel olduğunu belirtmeniz gerekir. Bunun için Use komutunun yardımına  
\*!\* bakabilirsiniz.  
Set Exclusive Off

\*!\* Çok kullanıcılı programlarda aynı kayıt üzerinde birden fazla kullanıcı işlem yapabilir.  
\*!\* Kayıt üzerinde işlem yaparken o kayıt işlem bitene kadar o kullanıcı için kilitlenmelidir.  
\*!\* Diğer bir kullanıcı aynı kayıt üzerinde işlem yapabilmek için o kaydın kilidinin açılmasını  
\*!\* beklemek zorundadır. Aşağıdaki komut işte bu bekleme süresini ayarlar.  
Set Reprocess To 3 Automatic  
\*!\* Çok kullanıcılı ortamlarda kayıtlar üzerinde işlem yaparken birden fazla kaydı kilitlemek istersiniz.  
\*!\* Paylaşımaya açılmış tablolarda otomatik kilitleme yada elle kilitleme işlemleri için aşağıdaki  
\*!\* Komutu kullanmak zorundasınızdır.  
Set MultiLocks On

```

** Tablolarımızda rahatlıkla Türkçe karakter kullanabiliriz. Ancak bu
tablolarda bir sıralama (index)
** yaptığımızda Türkçe sıralamaya girebilmesi için aşağıdaki komutu
kullanmamız gerekir. Eğer bu komutu
** kullanmazsak İ, Ş vb. karakterler sıralamada sona kalırlar. Bu durumda
kullanıcıların sıralı bir
** tablodan bir şey bulması sıkıntı olur. Ancak Türkçe sıralamanın
olabilmesi için programın
** kurulu olduğu bilgisayarın Türkçe sıralama bilgisinin yüklü olması
gerekmektedir. Bunu şu şekilde
** kontrol edebilirsiniz. Denetim Masası (Contol Panel) - Bölgesel Ayarlar
(Regional Settings) -
** Gelişmiş (Advance) bölümünde Unicode kısmında Türkçe seçilmelidir. Tabi
bu ayar her bilgisayarda
** tanımlı olmayabilir bu yüzden Set Collate komutunu kullandığımızda
programımızın bir hata mesajı
** vermemesini isteyebiliriz. Bu yüzden Try Catch bloğu içinde bu komutu
yazıyorum. Try Catch
** VFP 8 ile gelen bir komuttur. VFP 8 öncesi için On Error komutu ile bir
çözüm yapılabilir.
** Yeni başlayan arkadaşlar için Try Catch bloğunu çok az anlatmak
istiyorum. Komutun aşağıdaki gibidir.
** Try
** [Yapılmak istenen komutlar]
** Cacth
** [Hata oluştuğunda yapılmak istenen komutlar]
** Finally
** [Hata olsa da olmasa da yapılmak istenen komutlar]
** EndTry
Try
    Set Collate To "TURKISH"
Catch
EndTry

** Kodlama sırasında yapılan en büyük yanlışlıklardan biri ise kodlamanın
içine tam dosya yolunun
** yazılmasıdır. Örneğin bir dosya açmak için kullanılan Use komutunu eğer
şu şekilde olursa
** Use C:\Deneme\VeriLokal\Ayarlar In 0 programın VeriLokal klasörünün C
diskinde Deneme klasörünün
** altında olma zorunluluğunu getirirsiniz. Kullanıcı programı farklı bir
dizine kurabilir. Daha
** kullanıcıya gelmeden siz projenizi başka bir klasöre taşıyabilirsiniz.
Bu durumda program çalışmaz
** hale gelir. Oysa ki sizin için önemli olan programın nerede çalıştığı
değil VeriLokal klasörünün
** altındaki Ayarlar dosyasının açılmasıdır. Sizin tek şartınız exe'nin
çalıştığı klasör altında
** VeriLokal klasörü olması ve bu klasörün altında Ayarlar dosyanızın
olmasıdır. Bunun için varsayılan
** klasörünüzü ayarlamanız gerekir. Eğer varsayılan klasörünüz exe'nin
çalıştığı klasör olursa komutu
** Use VeriLokal\Ayarlar In 0 şeklinde kullanabilirsiniz. Bu tarz bir
kullanımda sizin sorunlarınıza
** çözüm olur. Varsayılan klasörü ayarlamak için Set Default komutunu
kullanmak gerekir. Ancak buradaki
** önemli soru program o an hangi klasörde çalışmakta. Bunu da sys(16,1)
komutu ile bulabiliriz.
** Özetle varsayılan klasörü ayarlamak için aşağıdaki kodu yazmamız
yeterli olur.
Set Default To (JustPath(Sys(16,1))

```

\*!\* 2.2.1 Veritabanı yaklaşımı kısmında açıkladığım üzere Lokal ve Genel isimli iki veritabanımız var.  
\*!\* Bu veritabanlarından Lokal olan exe'nin çalıştığı VeriLokal klasörünün altında olmak zorunda.  
\*!\* Genel veritabanımız VeriGenel klasörünün altında ve paylaşıma açılmış bir yerde olacaktır. Bu adres  
\*!\* program kodlamasında belli değildir. Bu yüzden bu adresin kullanıcı tarafından belirtilmesi  
\*!\* gerekir ki bizde VeriGenel klasörünün adresini bilip altındaki tabloları açabilelim. Eğer biz  
\*!\* Genel veritabanının adresini bilirsek örneğin Use komutunu şu şekilde kullanabiliriz.  
\*!\* Use VeriGenel\CariKart In 0 gibi. Burada dikkat edilecek nokta şudur VeriGenel klasörünün tam  
\*!\* yolunu yine yazmadım. VFP'ya VeriGenel klasörünün altındaki CariKart tablosunu aç dedim. Bu durumda  
\*!\* VFP ilk önce varsayılan klasör altında bu dosyayı arayacaktır. Eğer bulamazsa önceden tanımlı klasörler  
\*!\* altında bu dosyayı arayacaktır. Önceden tanımlı klasörleri ise Set Path komutu ile ayarlayabiliriz.  
\*!\* Şimdi bu paylaşıma açılmış Genel veritabanının adresinin kullanıcıdan alınması ve önceden tanımlı  
\*!\* klasörlerin ayarlanması görelim. Bunun için VeriYoluKontrol isimli procedure kullanacağım.  
\*!\* Bu procedure'u açıklamasıyla beraber aşağıda bulabilirsiniz.  
Do VeriYoluKontrol

\*!\* Foxpro ile tanışmadan önce VB kullanıyordum. VB'de yaptığım exe'leri çalıştırdığımda Windows ekranında  
\*!\* tasarlamış olduğum ekranları tek başına görüyordum. Yani VB'nin ekranı görünmüyordu. Ancak Foxpro'da  
\*!\* yaptığım exe'de form Foxpro'nun tasarım ekranının içinde açıldığında çok şaşırılmış ve hiç hoşuma gitmemişti.  
\*!\* Foxpro'nun bu tasarım ekranını nasıl kapatırım diye çok uğraştığımı hatırlıyorum. Oysa şimdi bu tasarım  
\*!\* ekranı olmadan bir program düşünemiyorum. Foxpro'nun tasarım ekranı dediğim bu ekranın adı \_Screen'dir.  
\*!\* Bu ekran o kadar kullanışlı ve o kadar çok işe yarıyor ki sormayın. O yüzden Foxpro'a yeni başlayan  
\*!\* arkadaşlara kesinlikle bu ekranı kullanmanızı tavsiye ederim. Bu \_Screen ekranını programınız ilk ekranı  
\*!\* olarak düşünün. Bunun üzerine menüler, toolbarlar ekleyip ana ekranınızı oluşturabilirsiniz. Bu kadar  
\*!\* gevezelikten sonra bu ekranı nasıl kendinize uyarlayabilirsiniz kısaca bir bakalım. Foxpro'nun simgesini,  
\*!\* başlığını değiştirip programın tam ekran açılmasını sağlayalım.  
\*!\* Not: Aşağıdaki simge atamasına dikkat edilecek olursa simgenin tam adresi verilmemiştir.  
\*!\* 1.1.1 Çalışma Klasörü'nde belirtmiş olduğum yapıya göre BMP klasörünün altındaki simge kullanılmıştır.

```
With _Screen
    .Icon          = "BMP\Icon.ico"
    .Caption       = "Benim ilk programım!.."
    .WindowState   = 2
EndWith
```

\*!\* Yukarıda \_Screen ekranından bahsetmiştim. Ben bu ekrana bir menü koyup programı sadece bu menü ile  
\*!\* başlatmayı tercih ediyorum. Kullanıcı yapacağı işlemi menülerden seçecektir. Bu yüzden programa uygun

```
!* bir menü tasarlamak gerekir. Menü tasarımı bir başka bölümde
incelenebilir. Burada tasarlanmış bir menünün
!* çalıştırılmasına bakalım. Tasarlanan menünün adı AnaMenu olup Menu
klasörünün atındadır. Bu program parçası
!* çalıştığında Foxpro'nun menüsü kaybolur ve bizim tasarladığımız menü
gelir.
```

```
Do Menu\AnaMenu.mpr
```

```
!* Artık programımız kullanıcın seçimine göre yönlenecek sayfaya geldi. Bu
durumda programın burada kullanıcı
!* komutlarını beklemesi gerekir. Bunun için aşağıdaki komut kullanılır.
Foxpro bu komutu görünce sonraki
!* komutları biz isteyene kadar çalıştırmaz. Zaten bu komuttan sonra da
program sonlanması gerekir ve _Screen
!* ekranında yapmış olduğumuz değişiklikleri geri alacağız. Programın
Read Events komutundan sonra ki
!* komutlara geçebilmesi için Clear Events komutu kullanılmadır. Bu komutu
menüde çıkış seçeneğine koyabilirsiniz.
!* Hatırlarımda ilk yaptığım exe'ye Read Events yazmadığım için program
açılmasıyla kapanması bir olmuştu.
!* Çünkü Foxpro Read Events'i görmediği için ana.prg dosyasının sonuna
geliyor ve bütün komutları işlettiği için
!* programı bitiriyordu.
```

```
Read Events
```

```
!* Read Events'den sonra yazılan komutlar artık programın son
komutlarıdır. Exe'den çalışan bir kod olarak
!* düşünürsek bundan sonra bir şey yazmak anlamsız. Ancak bizler tasarım
sirasında programı test amaçlı bir çok
!* kez çalıştırıp kapatıyoruz. Yukarıda Foxpro'nun tasarım ekranı olan
_Screen ekranının ayarları değiştirmiştik.
!* Şimdi bu ayarları geri almamız gerekir ki tasarımıımıza rahatlıkla devam
edelim. Hatırlayın Foxpro'nun menüsünün
!* yerine kendi menümüzü koymuştuk. Şimdi bu ayarları geri alalım.
```

```
Set SysMenu On
```

```
Set Sysmenu To Default
```

```
Set Talk On
```

```
Return
```

```
!*****!
!* Procedure ve Function      !*
!*****!
```

```
!* Procedure   : VeriYoluKontrol
!* Açıklama    : Genel veritabanının yerini kontrol eder veya kullanıcıdan
yolunu ister.
!*            Belirtilen yolu önceden tanımlı klasöre ekler. Eğer Genel
veritabanının
!*            yolunda bir sorun varsa programı kapatır.
```

```
Procedure VeriYoluKontrol
```

```
Local lcYol As String
```

```
!* Set Default komutu ile exe'nin çalıştığı yol varsayılan yol olarak
ayarlandığından
!* exe'nin bulunduğu klasörün altında VeriLokal klasöründeki MakineAyar
tablosunu
!* aşağıdaki gibi tam yol yazmadan açabiliriz. MakineAyar tablosunun
içinde mVeriYolu
!* adında bir memo saha vardır. Bununla ilgili olarak 2.2.1.1 Lokal
Veritabanı kısmına
```

```

** tekrar bakabilirsiniz. Bu mVeriYolu sahasında Genel veritabanının yolu
tutulmaktadır.
** Örneğin : C:\PROJELER\DENEME\VERIGENEL
    Use VeriLokal\MakineAyar In 0 Shared

** PathKontrol fonksiyonunda MakineAyar.mVeriYolu sahasında yazılı olan
yolu doğruluğu
** kontrol edilir. Eğer hatalı ise kullanıcıya Genel veritabanının yolu
sorulur.
** Bu işlemlerin sonunda yol doğru ise .T. hatalı ise .F. gönderir. .F.
geldiğinde
** bizim kullanacağımız tabloların adresi bilinmediğinden programın
çalışmasının bir anlamı
** kalmadığından programdan çıkılır. PathKontrol fonksiyonunu aşağıda
açıklamasıyla birlikte
** bulabilirsiniz.
    If !PathKontrol()
        Quit
        Return
    EndIf

** mVeriYolu sahasındaki yol doğru ise Set Path komutu ile bu klasör
varsayılan klasörler arasına
** eklenir. Bizim örneğimizde C:\PROJELER\DENEME\VERIGENEL idi. Buradaki
önemli nokta şudur.
** Bizim için önemli olan klasör VERIGENEL klasörü değil bu bu klasörün
üst klasörüdür yani C:\PROJELER\DENEME
** Çünkü biz bu klasör altına RAPOR isimli bir başka klasör daha oluşturup
raporlarımızı bu klasörün
** altına koyacağız. O zaman rapor yazdırmak için Report Form
Rapor\CariKart.frx to Printer gibi
** tam yol yazmadan komut kullanabiliriz.
    m.lcYol = Left(AllTrim(MakineAyar.mVeriYolu), Rat("\",
AllTrim(MakineAyar.mVeriYolu)) - 1)
    Set Path To (m.lcYol)
EndProc

** Function      : PathKontrol
** Açıklama      : MakineAyar.mVeriYolu sahasında yazılı olan yolu doğruluğu
**              : kontrol edilir. Eğer hatalı ise kullanıcıya Genel
veritabanının yolu sorulur.
**              : Bu işlemlerin sonunda yol doğru ise .T. hatalı ise .F.
gönderir.
Function PathKontrol
    If File(AddBS(AllTrim(MakineAyar.mVeriYolu)) + "Genel.dbc")
        Return .T.
    Else
** MakineAyar isimli bir form vardır. Bu form kullanıcıdan Genel
veritabanının yolunu sorar ve
** mVeriYoluna kaydeder. Bu formu şeklini ve çalışma mantığını buradan
yazamıyorum. Ama özetle
** GetFile komutu ile kullanıcıdan Genel veritabanının yerini bulup
seçmesini isteyen bir formdur.
        Do Form Form\MakineAyar
        Return File(AddBS(AllTrim(MakineAyar.mVeriYolu)) +
"Genel.dbc")
    EndIf
EndFunc

** Ana.prg dosyasının sonu

```

Ana.prg dosyasını kaydettiğinizde project manager'in Code sayfasında ana isimli bir kalem belirecektir. Bu kalın harflerle yazılmış olacaktır. Bunun dışındaki kalemler (formlar veya diğer prg dosyaları) kalın harflerle yazılmış olmayacaktır. Bu şu anlama geliyor Foxpro ilk olarak bu kodları işletecek. İlk işleteceği kodları başka bir prg yada form yapmak isterseniz o kalemin üzerinde farenin sağ tuşuna basın açılan menüden Set Main komutunu seçebilirsiniz. Ancak unutmayın burada ki anlatımlarda kesinlikle Ana.prg dosyası ilk işletecek kod olmalıdır.

## 2.3. Class'lara İlk Adım

Programlamaya daha yeni başlarken Class gibi bir konuya neden girdiniz demeyin. Keşke birisi de bana zamanında hemen class'larla çalış deseydi. Benim düştüğüm bu hataya sizlerde düşmeyin diye ben de sizlere hemen class'lardan kabaca bahsedip ilk class'larımızı oluşturmayı görelim.

### 2.3.1. Class Ne Demek?

Class'ın ne demek olduğunu yazmak çok cümle kurmak demek. Ben sizlerin aklını çok karıştırmadan basit anlamda (obje) class ne demek onu açıklamaya çalışayım. Foxpro'nun içinde hazır olarak gelen objelerden bazıları CommandButton, TextBox, EditBox, Form vb. bu objelerin özellikleri (properties) ve olayları (methots-events) vardır. Bu objeler bizim için önceden tasarlanmış objelerdir. Biz onların bu özelliklerini kullanarak kendi programımızı oluşturmakta kullanabiliriz. Örneğin bir CommandButton'u formun üzerine koyar Click Event'ine formun kapatılmasını sağlayan bir kod yazarız ya da forma bir grid ekler properties'inden DeletedMark özelliğini kapatır, HighLightStyle'ını 2 yapabiliriz. Şimdi şöyle düşünelim, programımızın her formunda bir kapat düğmesi olacak ve her kullandığımız gridin DeletedMark özelliği kapalı, HighLightStyle 2 olacak. Bu durumda her form için bu işlemleri tek tek yapabileceğimiz gibi classları kullanabiliriz. Tabi ki class sadece bu demek değildir. Bunun için bir farklı örnek daha vermeye çalışayım. MS SQL veritabanı ile iletişim kuran, ekleme, düzenleme ve silme işlemlerini yapabilen bir obje de tasarlayabilirsiniz. Özetle var olan objeleri değiştirip veya olmayan yeni objeler oluşturup geliştirebildiğiniz bir ortamdır. İşte bu oluşturduğunuz classları programınızda kullanmanız sizlerin kodlamayı çok daha çok yapmanızı ve çok daha kolay programınızı yönetmenizi sağlar.

Bu konuyu biraz daha açıp pekiştirmeye çalışayım. Formlarımızda kapat adından bir düğmemiz olsun bu düğmeye basılınca formumuz kapatılsın. Bunu yapan bir class oluşturduğumuzu varsayalım. Bu kapat class'ını formlara ekledik. Ekleyince tekrar kapat düğmesinin click event'ine kod yazmamıza gerek kalmaz. Çünkü bu kapat class'ının özelliğinde var. Bu durumda ilk avantajımız her form için bir form kapatan kod yazmamıza gerek kalmadı. Burada şu düşünülebilir ben bir kere kodu commandbutton'a yazar ve o düğmeyi diğer formlara kopyalarım bu durumda da tekrar kod yazmaktan kurtulmuş olurum. İşte class bu noktada çok büyük bir avantaj sağlıyor. Formlara düğmeleri ekledikten sonra Kapat düğmesinin Caption'ı koyu harflerle yazdırmaya karar verdiniz. Bu durumda sadece class'da değişiklik yapmanız yeterli olur ama diğer yöntemde her forma gidip tek tek değiştirmeniz gerekir.

Class'ları sadece küçük kod parçaları ve özelliklerini hızlı bir şekilde değiştirmeye yarayan bir ortam olarak düşünmeyin. Örneğin bir TextBox'dan özel bir class oluşturduğunuz bu yeni class içine bir e-posta adresi yazıldığında ve çift tıklandığında yeni e-posta iletisi açan özelliği olsun. Bu classı programınızın bir çok yerinde kullandınız. Sonra bir boş vaktinizde aklınıza

web adresi yazıldığında ve çift tıklandığında ilgili web adresini açmasını istediniz. Bunu aynı class'a eklediğinizde programınızda bu class'ı kullandığınız her yer otomatik olarak değişmiş ve gelişmiş olacaktır.

### 2.3.2. İlk Class'lar

İlk class'larınızı Foxpro'nun standart objelerinden oluşturabilirsiniz. Bu objeler üzerinde bir kaç ufak değişiklik yapabileceğiniz gibi (font, color vb) ilk başta hiç bir değişiklik yapmadan da kullanabilirsiniz. Bunun için Project Manager'da Classes sayfasına gelip New.. düğmesine basınız. Açılan pencerede Class Name kısmına yeni class'mızın adını yazmalıyız, based on kısmında referans objeyi seçmelisiniz, store in kısmında ise oluşturacağımız tüm bu classların saklanacağı kütüphanenin yoluyla beraber adını yazınız. Benim sizlere bu konuda şöyle bir tavsiyem olacak. İlk önce Temel kütüphane isminde içinde Foxpro'nun temel objelerini barındıran classlar oluşturun. Bu classların ismini verirken orijinal foxpro isimlerinin başına "Temel" isminin "t" harfini ekleyin. Bir örnek yapalım:

Class Name : tCheckBox

Based On : CheckBox

Store In : c:\projeler\deneme\vcx\temel.vcx dikkat edin klasör yapımıza uygun olarak proje klasörünün altındaki VCX klasörünün altına kaydediyorum.

Şimdi açılan Class Designer penceresinde istediğimiz değişikliği yapabiliriz. Eğer bir değişiklik yapmayacaksak kaydedip kapatın. Böylece ilk class'ınızı oluşturmuş oldunuz. Şimdi diğer objeler (ComboBox, CommandButton vs.) içinde benzer işlemleri yapın. Classlarda bazı objeler için size bir ipucu vereyim. Örneğin OptionGroup objesi bünyesinde OptionButton içerir. Bu durumda OptionGroup objesinin kullanacağı OptionButton objesinde kendi classımız olması gerekir. Bunun için ilk önce tOptionButton classını ilk önce oluşturun. tOptionGroup classını tasarlariken ButtonCount özelliğini 0 yapın. MemberClass özelliğini seçip açılan pencereden VCX\Temel.vcx'i seçip içinden de tOptionButton'u seçin. Classınızı kaydedin. Şimdi tOptionGroup objesini formun üzerine koyduğunuzda düğmesiz gelecektir. ButtonCount'ı 2 yapın göreceksiniz ki eklenen OptionButton'lar tOptionButton olacaktır.

Kesinlikle Form içinde bir class yapmayı unutmayın. Çünkü bütün formlarımız bu tForm classından oluşacaktır. Yeni form tasarlamak istediğimizde (projet manager'da Docs sayfasında Forms kalemi seçili iken New.. düğmesine basıldığında yeni form tasarımı açılır) açılan formun tForm classı olması için Tools --> Options'dan Forms sayfasına geliniz. Template Classes kısmında Form onay kutusunu işaretleyiniz. Açılan pencereden VCX\Temel.vcx kütüphanesinden tForm'u seçin Set As Default düğmesine basınız. Artık her yeni formumuz tForm classından olacaktır.

### 2.3.3. Class'dan Class

İlk classlarımızı Temel.vcx ismindeki kütüphanede oluşturduk. Şimdi bu classları kullanarak üst düze bir class oluşturalım. Bunun için yukarıda hep örnek olarak verdiğim kapat düğmesini nasıl yapacağınızı anlatayım.

Class Name : cmdKapat

Based On : tCommandButton (bunun için Based On'un sağındaki "..." başlıklı düğmeye

basarak VCX\Temel.vcx kütüphanesini seçiniz.)

Store In : c:\projeler\deneme\vcx\ozel.vcx (ozel adında yeni bir kütüphane oluşturuyorum)

Açılan class Designer penceresinde düğmenin Caption'ını Kapat olarak ayarlayın. Click Event'ine aşağıdaki kodu yazın ve classı kaydedin.

```
If Type("ThisForm.Parent") = "0"  
    ThisFormSet.Release  
Else  
    ThisForm.Release  
EndIf
```

### 2.3.4. Class'ların Kullanılması

Yukarıdaki açıklamaların sonucunda Temel ve Ozel isimli iki kütüphanede classlarımız oluştu. Şimdi bu classları nasıl kullanacağımızı kısaca gösterelim. Form classımızı 2.3.2 İlk Class'lar başlıklı yazının son paragrafında bahsetmişim. Diğer classları ise iki türlü kullanabilirsiniz. 1. yol form tasarımında project manager'da Classes sayfasından istediğimiz classı seçip sürükleyip formun üzerine bırakabiliriz. 2. yol olarak foxpro'nun standart toolbarında view clases düğmesinden Add.. komutunu seçip istediğimiz kütüphaneyi seçebilirsiniz. Seçtiğinizde standart toolbar seçili kütüphanenin classlarını gösterecektir. Bu toolbardan istediğimiz classı seçip formun üzerine bırakabiliriz.

Classlardan kesinlikle korkmayınız. Gördüğünüz üzere classları oluşturmak ve kullanmak çok kolay. Zaten class kullanımında 2. yol foxpro'nun standart obje kullanımı ile aynıdır.

Form ve diğer tüm objeleri kendi classımız olarak değiştirdiğimizden artık çok rahatız. Bu rahatlığı ilerleyen günlerde sizlerde farkedeceksiniz.

## 3. Veritabanına Başlarken

Foxpro'yu seçmemin en büyük nedeni verileri yönetmedeki ustalığı ve programcıya sağladığı kolaylıklar olmuştur. Foxpro'nun veri yönetme özelliklerinden önce veriler nasıl saklanır saklarken nerelere dikkat etmemiz gerekir bunlara kısaca bir bakalım.

### 3.1. Tablo

#### 3.1.1. Genel Yapı

Verileri iki boyutlu matrislerde saklayabiliriz. (Matris boyutları 2'den fazla olabilir ama bu farklı bir durumdur.) Bu matrislere tablo diyebiliriz. Tablolar sütun ve satırlardan oluşur. Bu tabloları kullanırken aktif satırı işaret eden bir işaretçi (pointer) vardır. Tablo konusunu bir örnekle açıklamaya çalışayım. Amacımız ad ve soyadı bilgisini tutmak olsun. Bunun için cAd, cSoyadi saha (field) başlıklı bir tablomuz olsun.

<i>cAd</i>	<i>cSoyadi</i>
Tarkan	Haser
Özgür	Yıldırım

Şimdi bu yukarıdaki bilgilerin Foxpro'nun tablo yapısında nasıl görüldüğüne adım adım bakalım.

Kayıt No	Silinmiş mi?	<i>cAd</i>	<i>cSoyadi</i>
1	Hayır	Tarkan	Haser
2	Hayır	Özgür	Yıldırım

Yukarıdaki tabloya dikkat edecek olursak iki sütun daha vardır. Kayıt No ile Silinmiş mi? sütunları. Kayıt no sütununda kaydın numarası, Silinmiş mi sütununda ise kaydın tablodan silinip silinmediğini tutan bilgi vardır. Biz kaydı silmek istediğimizde kayıt fiziksel olarak tablodan çıkartılmaz sadece o kaydın silindiğini işaret eden Silinmiş mi sütunu işaretlenir. Biz tabloya yeni bir kayıt eklediğimizde bu kayıt tablonun sonuna yazılır. Kayıt No otomatik olarak sıradaki numara olur ve silinmiş mi sütunu hayır olur.

Kayıt No	Silinmiş mi?	<i>cAd</i>	<i>cSoyadi</i>
1	Hayır	Tarkan	Haser
2	Hayır	Özgür	Yıldırım
3	Hayır	Mustafa	Kılıç

Foxpro içinden bu tabloya baktığımızda kayıt no sütunu göremeyiz. Bu sütunu sizlere konuyu daha iyi açıklayabilmek için gösteriyorum. Silinmiş mi sütunu ise satırların (Browse ekranında) yanında beyaz bir kutu gibi görebilirsiniz. Silinmiş kayıtları ise siyah bir kutu olarak görebilirsiniz. Bu detaya daha sonra tekrar deyeceğim.

Bu tablonun içinde bir de o an aktif olan kaydı gösteren bir işaretçi vardır.

İşaretçi	Kayıt No	Silinmiş mi?	cAd	cSoyadi
->	1	Hayır	Tarkan	Haser
	2	Hayır	Özgür	Yıldırım
	3	Hayır	Mustafa	Kılıç

Tabloyu ilk defa açtığınızda bu işaretçi birinci kaydı gösterir. Biz bazı komutlar ile bu işaretçinin yerini değiştirerek ilgili kayda ulaşabiliriz. İşaretçiyi Browse ekranında en solda ok şeklinde görebilirsiniz. Bir tablodan bilgi okumak için bu işaretçinin yeri önemlidir. İşaretçi hangi kaydı gösteriyorsa biz o kayıt üzerinde işlem yaparız.

### 3.1.2. Sahalar

Bir tablo oluştururken kullandığımız saha tipleri çok önemlidir. Tablolarımızı optimum büyüklükte tutmamız gerekir. Gereksiz büyüklüklerden kaçınmamız gerekir. Örneğin işlem tiplerini tuttuğumuz iki basamaklı bir saha için integer yerine numeric (2,0) kullanmalıyız.

Saha Tipleri	Saha Genişliği	Dijit	Veri Tipi
W, Blob	-	-	<b>Blob</b>
C, Char, Character	N	-	<b>Karakter</b>
Y, Currency	-	-	<b>Para</b>
D, Date	-	-	<b>Tarih</b>
T, DateTime	-	-	<b>Tarih Saat</b>
B, Double	-	d	<b>Hassas Sayısal</b>
G, General	-	-	<b>Genel</b>
I, Int, Integer	-	-	<b>Tam Sayı</b>
L, Logical	-	-	<b>Mantıksal</b>
M, Memo	-	-	<b>Not</b>
N, Num, Numeric	N	d	<b>Sayısal</b>
F, Float	N	d	<b>Sayısal</b>
Q, Varbinary	n	-	<b>Değişken Binary</b>
V, Varchar	n	-	<b>Değişken Karakter</b>

Burada tüm sahalara detaylı açıklamasını yapmayacağım. Ancak önemli olan bir iki saha tipinden bahsetmek istiyorum.

#### **C, Char, Character :**

Örneğin cAd C (20) tipinde bir sahanız olsun. Bunun anlamı 20 karakter uzunluğun bir cAd sahanız vardır. cAd sahasına 20 karakterden daha az veya daha çok veri yazamazsınız

demektir. Tabii hemen şunu düşünebilirsiniz, ben "Ali" ismi yazsam 3 karakter olur nasıl olurda 20 karakterden daha az bir isim yazamam ki. Siz Ali'de yazsanız Foxpro Ali isminin sonuna 17 tane boşluk karakteri ekleyerek "Ali" ismini "Ali " haline getirip tablonun cAd sahasına yazar. Dolayısıyla bu sahadan okuduğunuz her bilginin boyu 20 karakterdir. Bu özelliği lütfen unutmayın çünkü; indeks ve karşılaştırma işlemlerini bu özelliğe göre yapacağız.

## V, Varchar

Bu sefer cAd V (20) tipinde bir sahanız olsun. Bunun anlamı en fazla 20 karaktere kadar veri tutabilirsiniz demektir. Siz "Ali" yazarsanız FoxPro bunu yine "Ali" olarak saklar. Character tipinde olduğu gibi sonuna 17 tane boşluk karakteri eklemeyiz.

## N, Num, Numeric

Bu saha tipi tutacağı sayısal rakamın dijit işareti dahil toplam boyu ve dijit sayısını belirterek oluşturulur. Örneğin en fazla tutacağımız sayı 999,99 ise nSayı N (6,2) şeklinde bir saha belirleyebiliriz. Burada "," dahil toplam boy 6, ","den sonra ise 2 dijit kullanılmıştır.

## F, Float

N, Num, Numeric saha tipi ile aynıdır. Eski Foxpro versiyonları ile uyumluluk için vardır.

## B, Double

Matematiksel işlemlerde özellikle dijitten kaynaklanan sorunlardan kurtulmak ve daha doğru sonuçlar elde etmek için bu saha tipini kullanabilirsiniz. Bu cümle biraz kapalı oldu. Bu durumu bir örnek ile açıklamaya çalışalım. Bir işlem sonucunda 22 rakamını 7'ye bölmemiz ve çıkan sonucu tabloda saklamamız gerekmektedir. Bu sonucu ise bir başka zaman kullanmamız gerektiği bir durumu düşünelim.

$22/7 = 3,1428571428571428571428571428571$  kendini tekrarlayan bir dijite sahip bir sayı olur. Bundan sonraki kısımları ise bir gerçek örnek ile yapalım.

```
Create Cursor curDeneme (nSonuc1 B(2), nSonuc2 N(6,2))
Insert Into curDeneme Values (22 / 7, 22 / 7)
```

```
? curDeneme.nSonuc1          && 3,14 olarak görünür. Çünkü bu saha Double
olup 2 dijit olarak ayarlanmıştı.
? curDeneme.nSonuc1 * 10000   && 31428,57 olarak görünür. Dikkat edin bu
çarpma işlemi 3,14 ile değil 3,1428571428571428571428571428571 ile
olmuştur.
```

```
? curDeneme.nSonuc2          && 3,14 olarak görünür. Çünkü bu daha Numeric
olup 2 dijit olarak ayarlanmıştır.
? curDeneme.nSonuc2 * 10000   && 31400,00 olarak görünür. Dikkat edin bu
çarpma işlemi 3,14 ile yapılmıştır. Doubledaki gibi değil.
```

```
? curDeneme.nSonuc1 * 7      && 22,00 olarak görünür
? curDeneme.nSonuc2 * 7      && 21,98 olarak görünür.
```

Hassas işlemlere çok nadir ihtiyaç duyuyorum. Ancak yine de Double saha tipini sıkça kullanıyorum. Toplam boyu 8'i geçen tüm sayısal tipler için double tipini tercih ediyorum.

Çünkü tabloda N(12,2) saha tipi 12 byte yer tutarken B(2) 8 byte yer tutmaktadır. neresinden baksanız her kayıt için 4 byte kar etmiş oluyorum. Bu yüzden N(8,2) üzeri tipler için B(2) tercih ediyorum. Bu sahayı kullanırken de hassasiyete ihtiyacım yoksa Round fonksiyonunu kullanıyorum. Yukarıdaki örnek için ;

```
Insert Into curDeneme Values (Round(22 / 7, 2), 22 / 7)
```

### 3.1.3. Tablo Büyüklüğü

Tablolarımızı oluşturmaya başladığımızda dikkat etmemiz gereken noktalardan biri ise bu tabloya kayıtlar eklenmeye başladığında tablo boyumuz ne olacak? Büyük dosyalar üzerinde işlem yapmak her zaman sıkıntı olmuştur. O yüzden sizde oluşturduğunuz tablonun boyunun ne kadara çıkacağını tahmin ederek bir hesap yapın. Tablo oluştururken "Table Designer" formunda Width sahasını göreceksiniz. Bu saha size o sahanın boyunu byte cinsinden söyler. Sahaların bu boylarını toplarsanız size bir kaydın toplam uzunluğunu verir. Bu bilgiyi "Table Designer" formunun "Table" sayfasında da bulabilirsiniz. Length : xxx yazar. Örneğin bu uzunluk 2056 byte olsun. Bu durumda 30000 kayıt için bu tablo  $30000 * 2056 +$  tablonun header kısmı. Tablonun header kısmı çok küçük olduğundan göz ardı edilebilir. Bu durumda  $30.000 * 2.056 = 61.680.000$  byte = 60.234,38 KByte = 58,82 Mbyte olur.

Bu sonuçtan nereye ulaşmak istiyoruz. Tablolarını tasarlarken optimum alan kullanmaya çalışın. Örneğin Numeric (9) ve üzeri sahalara yerine B(2) -double- kullanın. Çünkü double alanlar 8 byte yer tutar. Ancak double ile ilgili yaptığın yukarıdaki açıklamayı unutmayın. Numeric(8) den küçük tam sayı değerleri için Integer saha tipini kullanın. Saha boyutları ile ilgili olarak foxpro yardım kısmından "Visual FoxPro Data and Field Types" başlığına bakın.

### 3.1.4. İndeks

İndeks ile ilgili yazıyı yazarken "tabloları indekslemek" tabirini kullandım sonra kendi kendime dedim ki bu hatalı bir tabir çünkü bu yapılan işlem indekslemek değil "indeks oluşturmak". O yüzden yazdıklarımı yeni baştan derledim. Garip bir başlangıç oldu ama bu başlangıcın sizleri etkilediğini umuyorum. İndeks oluşturmak ne demek, ne işe yarar? İndeksleri kabaca şu şekilde tanımlayabiliriz.

- Kayıtları daha hızlı sorgulamamıza yardımcı olur
- Kayıtları daha hızlı güncellememize yardımcı olur
- Kayıtlarda daha hızlı arama yapmamıza yardımcı olur
- Diğer tablolarda ilişki (relation) kurmamızı sağlar
- Kayıtları sıralamamıza yardımcı olur

Peki neden "tablolar indekslemek" yerine "indeks oluşturmak" tabirini kullandım? Tabloyu indekslemek mevcut kayıtlara göre bir indeks oluşturmak anlamına gelir ancak indeks oluşturmak farklıdır. Biz bir indeks oluşturduğumuzda eklenen, değiştirilen ve silinen her kayıt otomatik olarak indekslenir. Bizim tekrar tabloyu indekslememize gerek kalmaz. Özetle tabloya her eklediğimiz indeks kalıcı olur ve kayıtlarda bir değişiklik olduğunda yeniden indekslenmesine gerek kalmaz.

İndeksi gerçekten bir kitap indeksi ya da içindekiler sayfası olarak düşünebilirsiniz. Örneğin kalınca bir kitap içinde içindekiler sayfasına bakarak istediğiniz konunun kaçınıcı sayfada olduğunu bulup hızlı bir şekilde o sayfadaki bilgiye ulaşabilirsiniz. İşte programda

kullandığımız bir çok komutta bu indeksleri kullanarak uygun kayıt yada kayıtlara hızlı bir şekilde erişebilir. Daha geçen gün başıma gelen bir olayı sizlerle paylaşmak istiyorum. Bir projemizde MS SQL kullanıyoruz ilgili müşterimiz bize yavaşlık sorunu ile geldi. Her geçen gün programın yavaşladığını söyledi. Gerçekten de öyle bir form var ki yaklaşık 10 saniyede açılıyor. O formun kullandığı tabloya baktım yaklaşık kayıt sayısı 1.500.000 olmuş. Select SQL komutum gerçekten çok yavaş çalışıyor ama komutta optimize edecek bir yer yok. Sonradan fark ettim ki tablonun indeksleri verilmemiş. İndeksleri oluşturduktan sonra formun açılması 1 saniye bile sürmez oldu.

Yukarıdaki örnekten de anlaşıldığına göre indeks oluşturmak en az kodlama kadar önemlidir. 4 indeks tipi vardır.

- Primary (Birincil)
- Candidate (İkincil)
- Regular (Normal)
- Binary (ikili)

### **Primary**

Birincil indeks tabloda bir tane olabilir. Tabloda indeks koşuluna uyan tek bir kayıt olabilir anlamına gelir. Örneğin bir stok kartında mal kodu sahası primary indeks yapılabilir. Çünkü stok kartı tablosunda aynı mal koduna sahip birden fazla kayıt olmamalıdır.

### **Candidate**

Primary indeks tabloda bir tane olabilir demiştik ama bizim indeks koşulan uyan tek kayıt olma durumu primary sahası dışında da olabilir. Bu durumda bu sahalara için candidate kullanabiliriz. Örneğin stok kartında stok adı yine tek isteyebiliriz. Bu durumda stok adı kısmının indeksini candidate yapabiliriz.

### **Regular**

Normal indeks. İndeks koşuluna uyan birden fazla kayıt olabilir.

İndekslerle ilgili genel kanı şudur; indeks kayıtlarımızı sıralar browse komutunu ile yada grid'e kayıtları sıralı şekilde getirir o kadar. Biraz daha iyisi seek komutu ile arama yaparken kullanırım bunun içinde ilk iş olarak ilgili indeksi tayin ederim (set order) sonra ararım. Evet bu söylenenler doğru ancak hatalı bir düşünce tarzıdır. İndeksler bağımsız kullanılabilen sistemlerdir. Örneğin arama yapmak için ilgili indeksi tayin etmenize gerek yoktur. Seek fonksiyonunda hangi indekse göre arama yapacağınızı söyleyebilirsiniz. Siz farkında olmadan da indeksleri kullanabilirsiniz. Örneğin bir replace komutunda. Nasıl mı? Foxpro içinde bazı komutların daha hızlı çalışabilmesi için bir optimizasyon yöntemi kullanılmıştır. Adı Rushmore optimizasyon yöntemidir. Rushmore yöntemini kullanan komutlar indeksleri kullanarak işlemi daha çabuk yaparlar. Bu komutlar Select (SQL), Update (SQL), Delete (SQL), Average, Blank, Browse, Calculate, Change, Copy to, Copy to Array, Count, Delete, Display, Edit, Export, Index, Label, List, Locate, Recall, Replace, Replace From Array, Report, Scan, Sort, Sum, Total.

Örneğin ;

```
Replace nBirimFiyat With nBirimFiyat * 1.1 For cKodMal = "153001" In  
StokHareket
```

komutunda stok hareketlerinin tutulduğu StokHareket tablosunda mal kodu 153001 olan kayıtların birim fiyatlarını %10 artırılıyor. Eğer StokHareket tablosunda cKodMal sahası için bir indeks varsa indeksiz haline göre çok daha hızlı çalışacaktır.

Select (SQL) komutunda join ve where kısımları da otomatik ilgili indeksleri kullanır.

### 3.1.5. Tablo Tipleri

#### **Bağımsız Tablolar**

Bağımsız tablolar bir veritabanı çatısı altında olmayan tek başına yönetilebilen tablolar olarak açıklayabiliriz. Bu tip tablolara genel ve bir başka tablo ile ilişkisel yönetimler eklenemez.

#### **Veritabanı Tabloları**

Bu tablolar bir veritabanı çatısı altında olurlar. Genel ve başka tablolara ilişkisel yöntemler eklenebilir.