=== Programlama ve Tasarım ===

--- The New Datasnap in Delphi 2009 ---

COM Arayüz'ü kullanarak Delphi'de Çok Katmanlı Uygulamalar Geliştirmek

--- Giriş ---

Delphi, uzun zamandır çok katmanlı veritabanı uygulamaları geliştirmek için bir teknoloji sunuyordu. DCOM teknolojisinin yerine, uzaktan bağlantıyı soketler veya HTTP ile sağlayan, COM'a dayanan bu yapı eskiden MİDAS, şu anda ise DATASNAP olarak bilinmektedir.Bir ara SOAP bağlantısını destekleyen çok az değiştirilmiş bir versiyonu olan CORBA teknolojisini destekledi̇ler.

Delphi 2009, klasik datasnap yapısını desteklediği gibi yeni bir uzaktan yönetim ve çok katman teknolojisini desteklemektedir. Bu teknoloji, dbExpress mimarisine dayanmaktadır.Bu teknolojinin adı da datasnap'tir ama kavram karmaşasına yol açmamak için "DataSnap 2009" olarak adlandırılmaktadır.

--- DataSnap 2009 Demosu Oluşturmak ---

Çok fazla detayın içine girmeden önce, basit 3 katmanlı veritabanı demosu ile başlayacağım. Bu, birkaç noktayı açığa kavuşturacak ve bir önceki versiyonla arasındaki farkları göstermemizi sağlayacaktır.

--- Sunucu Oluşturmak ---

İlk adım, DataSnap 2009 sunucu uygulaması oluşturmaktır. Bu, içine bir sunucu modülü eklemek için standart bir VCl uygulaması da olabilir (Sunucu Modülü, Yeni Eleman Ekleme (New İtem) diyalog kutusu içerisinde Delphi Dosyaları (Delphi Files) bölümünde bulunur, Çok Katmanlı (Multi-Tier) bölümünde değil)

Sunucu Modülün içerisine (standart bir veri modülü (data module) de eklenebilir aslında) veritabanı sunucusuna bağlanmak için genelde dbExpress bileşenleri (component) ve ek olarak belirlenen verisetlerini göstermek için bir veri seti sağlayıcısı bileşeni (dataset provider) eklenmelidir

```
object IBCONNECTION: TSQLConnection
        ConnectionName = 'IBCONNECTION'
        DriverName = 'Interbase'
        LoginPrompt = False
        Params.Strings = (
        'DriverName=Interbase'
        'Database=C:\Program Files\...\Data\Employee.GDB')
end
object EMPLOYEE: TSQLDataSet
        CommandText = 'EMPLOYEE'
        CommandType = ctTable
        SQLConnection = IBCONNECTION
```

```
end
object DataSetProviderEmployee: TDataSetProvider
        DataSet = EMPLOYEE
end
```

Sunucu modülü, daha önce yapılana çok benzer bir şekilde kuruldu. Peki COM desteği içerisindeki (tamamen gitmiş) konfigürasyon ve bağlantı desteği için ihtiyacımız olan 3 adet bileşen nerede? Bu 3 bileşen de şunlardır;

DSServer: Bütün DataSnap bileşenlerini bir arada tutarak iletişimini sağlayan, ana sunucu konfigürasyon bileşenidir.

DSServerClass: Her sınıf için o sınıfı sergilemeye yarayan bileşendir.Bu bileşen, bir sınıfı erişilebilir hale getirmez, ama bir sınıf fabrikası (class factory) gibi sınıfın nesnelerini, uzaktan bir istemcinin çağırmak isteyeceği nesneleri oluşturabilir.Bir diğer deyişle sınıfın public olarak tabir edilen parçalarını gösterir.

DSTCPServerTransport: Kullanılacak taşıma protokolünü ve konfigürasyonunu ayarlamaya yarayan bir bileşendir. (Delphi 2009 için direkt erişim sağlanabilecek tek bir protokol vardır.). Örnek olarak hangi TCP/IP portunun kullanılacağı verilebilir.

Demoda bu bileşenler sunucunun ana formundadır, şu şekilde konfigüre edilebilir.

```
object DSServer1: TDSServer
        AutoStart = True
        HideDSAdmin = False
        OnConnect = DSServer1Connect
        OnDisconnect = DSServer1Disconnect
end
object DSTCPServerTransport1: TDSTCPServerTransport
        PoolSize = 0
        Server = DSServer1
        BufferKBSize = 32
end
object DSServerClass1: TDSServerClass
        OnGetClass = DSServerClass1GetClass
        Server = DSServer1
        LifeCycle = 'Session'
end
```

Bu özelliklerin bazı detaylarına ilerde değineceğiz. Yukarıda TCP/IP portunun değerinin görememenizin sebebi, varsayılan (default) değer olan 211 üzerinde herhangi bir değişiklik yapmadığımdandır. Şu anda yazmamız gereken tek Delphi kodu, sağlayıcıları sergileyen sunucu modülüne bağlanacak DSServerClass1 bileşenine bağlanmak için sınıf fabrikası kodunu yazmaktır.

```
procedure TFormFirst3Tier2009Server.
        DSServerClass1GetClass(DSServerClass: TDSServerClass;
        var PersistentClass: TPersistentClass);
begin
        PersistentClass := TDSFirst3TierServerModule;
end;
```

Sunucu için gerekli olan kod kesimi sadece bu kadar. Üst taraftaki kod, DSServer bileşeninin olayları olan OnConnect ve OnDisconnect (bağlanma ve bağlantı kesilmesi) olay tutulması (event handle) için bir loglama deyimi yazılmıştır.

Tekrar belirtiyorum ki, her hangi bir yoldan üye olmaya (register) gerek yok. Basitçe Delphi IDE'sindeki Çalıştır/Debug Etmeden Çalıştır (Run/Run Without Debugging) özelliklerinden herhangi biri seçilerek basitçe çalıştırılabilir. Ayrıca istemci tarafı inşa edilebilir ve tasarım zamanlı (design time) olarak istemci , sunucuya bağlanabilir.

--- İlk İstemci ---

Şu anda erişilebilir bir sunucumuz olduğuna göre ilk istemcimizi oluşturabiliriz. DataSnap 2009 istemci uygulamasında yeni DataSnap dbExpress sürücüsü ile birleşebilecek bir SQLConnection (SQL Bağlantı) bileşeni kullanmalı ve bunu uygun TCP/IP portu ile konfigüre etmeliyiz.

Sonrasında ServerClassName özelliği ile sunucu sınıfa (server class) adres gösterecek bir DSComponentProvider bileşeni kullanmalıyız. Bu sunucudaki aracı bir sınıf fabrikası değildir(DSServerClass1), ama sınıf fabrikasının esas hedefidir. Benim örneğimde bu TDSFirst3TierServerModule sınıfıdır.

Geleneksel bir DataSnap uygulamasında olduğu gibi, ClientDataSet (istemci veri seti) bileşeni veri setine (dataset) gidip gelme, güncelleme ve uzaktan erişim için sağlayıcıyı kullanır. Öncelikle ClientDataSet'in RemoteServer özelliğine, sürükle-bırak listesindeki (drop-down) DSProviderConnection1 bileşenini sürekleyip bırakarak atama yaparız. Sonra ProviderName özelliğine, DataSetProviderEmployee sağlayıcısını gene sürükle bıraktan erişerek atarız ki uzaktaki veri modülünde bütün çıkarılmış DataSetProvider bileşenlerini dolduran bir sağlayıcıdır.

Aşağıdaki kod parçası yukarıda bahsettiğimiz bileşenlerin özelliklerini anlatan ayrıca bir veri kaynağı (DataSource) bileşeni, ki veritabanındaki bir tabloyu bir DBGrid içerisinde göstermek için kullanılır, içeren bir kod parçasıdır.

```
object SQLConnection1: TSQLConnection
        DriverName = 'Datasnap'
end
object DSProviderConnection1: TDSProviderConnection
        ServerClassName = 'TDSFirst3TierServerModule'
        SQLConnection = SQLConnection1
end
object ClientDataSet1: TClientDataSet
        ProviderName = 'DataSetProviderEmployee'
        RemoteServer = DSProviderConnection1
end
object DataSource1: TDataSource
        DataSet = ClientDataSet1
end
```

Tanıtıcı bir demoyu oluşturacak her şey buradadır. Eğer önce sunucuyu sonra istemciyi çalıştırarak açma ile ilgili butona tıkladığımız zaman veritabanı içerisindeki veriyi görebilirsiniz. Ayrıca sunucu tarafından üretilen log da görülebilmektedir.

--- DataSnap'tan DataSnap 2009'a ---

Geleneksel DataSnap uygulamaları ile karşılaştırıldığında, birkaç önemli fark bulunmaktadır. Mimari ve dağıtım ile daha çok ilişkili olarak esas kod üzerinde şunlar yazılmalıdır.

- Sunucu geliştirmek için COM'a ihtiyaç bulunmamaktadır. Eğer kullanıcı, geçmişteki soketleri kullanmak istiyorsa, bunun için sunucu tarafında soketten COM'a (socket-to-COM) haritalama servisi bulunmalıdır. Şu anda sunucu ve istemci uygulamalar direkt TCP/IP üzerinden iletişim kurabilmektedir.
- Yan etki olarak, sunucuya üye olmak zorunda değiliz, ayrıca yardımcı bir servis çalıştırmak zorunda da değiliz. Sunucunun sağlaması gereken tek şey, istemcinin erişebilmesi için açık bir TCP/IP portudur.
- Sunucu tarafındaki uygulamayı bir kere elle (manuel) çalıştırmak zorundayız, ya da bunu için bir servis oluşturulmalı. Eskiden COM desteği, ihtiyaç duyulduğunda sunucu uygulamasının çalışacağını işaret ediyordu.
- Bileşenler açısından sunucunun hayata geçirilmesi biraz daha karmaşıktır ama COM meslektaşına göre arka planda çok daha az kod yazarak işlemler halledilir.
- İstemcinin uygulamaya konması da aynı şekilde özdeştir. Sadce ihtiyacımız olan özel bir bağlantı bileşeni yerine standart bir SQLConnection bileşenidir.
- Sunucu tarafında, TDSServerModule, TDataModule bileşeninden IAppServer arayüzünü içerecek şekilde (daha önceden COM tabanlı TRemoteDataModule tarafından da kullanılan arayüz) ve $MethodInfo derleyici direktifine olanak sağlayacak şekilde kalıtım yapar
- İstemci tarafındaki dbExpress sürücüsü, %100 saf Delphi sürücüsü olduğundan, bağlantı için dbExpress kullanılsa bile istemci tarafında her hangi bir .dll dosyası oluşturmaya gerek yoktur.

Sunucu uygulamayı kapatırken son derece dikkatli olmak gerekir. COM mimarisine ters olarak, COM bekleyen bağlantı talepleri için uyarı verir, DataSnap 2009 sunucusu eğer üzerinde herhangi bir bağlantı kalmamışsa kapanır. Buna rağmeni ana form gitse bile, bütün bağlantılar kapanmış olsa bile bellekte yer kaplamaya devam eder. Sunucu uygulamasını kapatmak için Görev Yöneticisini çalıştırmak veya İşlem Gezgini çalıştırmak gerekir. Bütün istemci uygulamaların kapanması sizin için yeterli görünebilir ama gerçek bu şekilde değildir. Delphi IDE'si gerçekte sunucuya sergilenen metodları ve sınıflarını araştırmak için otomatik olarak bir bağlantı açar. Sunucuyu durdurmadan önce bütün SQLConnection nesnelerini kapatmayı unutmamak gerekmektedir.

--- Sunucu Metodları Ekleme ---

Geçmişte olduğu gibi, şu anda da sunucu içerisine istemci taraftan çağrılması için metodlar yazabiliriz. Bu COM teknolojisine dayalı olup, bunlar için öncelikle tür kütüphanesi (type library) ve arayüzleri (interface) eklemek, bunları sunucu nesnelerinde tamamlamak, ve istemci tarafta COM yapısını kullanarak bu metodları çağırmak gerekmektedir. DataSnap 2009'da uzaktan yönetim metodları çağırma, sunucu metodları çağırma Delphi'nin Çalışma Zamanı Tür Tanımlamasına (RTTI = Run Time Type İdentification) aittir. Önemli bir nokta olarak, bununla beraber geçiş yapan parametreler Delphi'nin dil türü olmayıp, dbExpress parametre türlerine dayanır.

Biz metodları gösteren çoklu sunucu sınıflarına sahip olabiliriz, ama daha önce oluşturduğumuz basit proje ile devam etmek için sunucu uygulamasındaki sunucu modül sınıfına ekstra bir metod yazmak gerekir. Bununla ilgili kod aşağıdadır.

```
type
    TDSFirst3TierServerModule = class(TDSServerModule)
        IBCONNECTION: TSQLConnection;
        EMPLOYEE: TSQLDataSet;
        DataSetProviderEmployee: TDataSetProvider;
    private
        { Private declarations }
    public
        function GetHello: string;
    end;

function TDSFirst3TierServerModule.GetHello: string;
begin
    Result := 'Hello from TDSFirst3TierServerModule at '
    + TimeToStr (Now);
end;
```

Uzaktan yürütmeyi yapılabilir hale getirmek için metodlarını DSServerClass fabrikasına sergileyebilmek için gerekli olan sınıfa bağlanmamız gerekir. (Bu durumu biz zaten veritabanı demo kısmında gerçekleştirdik.) İkinci yapmamız gereken şey, $MethodInfo direktifi etkinleştirilmiş olarak derlenmiş bir sınıfı kullanmaktır ama bu zaten ana sınıf olan TDSServerModule sınıfında bir tanım olarak yer almıştır. Bu da demek oluyor ki bizim daha önce yazılan koda yapacağımız şey, sadece sunucu modüle public diye tabir edilen bir metod eklemektir. Bunun dışında gereken her şey yapılmıştır.

İstemci uygulamadan sunucu metodu nasıl çağırabiliriz? Bunun için basitçe 2 alternatif vardır. İlki, yeni bir SqlServerMethod bileşeni kullanmak ve sunucu metodu sanki bir saklanan prosedür (stored procedure) gibi çağırmaktır. İkincisi ise, istemci uygulamada bir proxy sınıfı oluşturmak ve metodu çağırmada bu sınıfı kullanmaktır.

Aşağıdaki istemci demosunda 2 metodu da uyguladım. İlki için istemcinin formuna bir SqlServerMethod bileşeni ekledim, bağlantıya bağladım, ServerMethodName özelliği için Object İnspector tarafında bir değer ekledim (pek çok uygun değer arasında standart IAppServer arayüz metodları da listelenmiştir.), ve Params özelliğinin değer alanını kontrol ettim. Aşağıdaki kod bileşen ayarlarının bir kopyasıdır. (Yapılan örnek bir çağrıyı parametreleri kontrol ederek uygularken ortaya çıkan sonucu da içerir.)

```
object SqlServerMethod1: TSqlServerMethod
    GetMetadata = False
    Params = <
        item
            DataType = ftWideString
            Precision = 2000
            Name = 'ReturnParameter'
            ParamType = ptResult
            Size = 2000
            Value = 'Hello from TDSFirst3TierServerModule...'
        end>
    SQLConnection = SQLConnection1
    ServerMethodName = 'TDSFirst3TierServerModule.GetHello'
end
```

Doğal string türü, 2000 karakterlik bir string parametresi ile haritalandırılmıştır. SqlServerMethod bileşeninin konfigürasyonu bittikten sonra, program bu metou giriş

parametrelerini kullanarak (burada değil şu anda) ve çıkış parametreleri (sonuç olarak) ile bir stored procedure veya sorgu cümleciği (query) gibi çağırabilir.

```
procedure TFormFirst3Tier2009Client.btnHelloClick(
Sender: TObject);
begin
        SqlServerMethod1.ExecuteMethod;
        ShowMessage (SqlServerMethod1.Params[0].Value);
end;
```

Çağırma kodunu daha kolay bir şekilde yazabilmek için daha önce de bahsettiğim istemci uygulama içerisinde bir proxy sınıfı oluşturma daha yararlı olur bizim açımızdan. Bunu başarabilmek için Delphi IDE'sini sunucu sınıfın arayüzünü parse edecek ve lokal bir proxy sınıfı oluşturacak bir şekilde kullanabiliriz. (SQLConnection bileşenine tıklayarak DataSnap Oluştur (Generate DataSnap) komutunu çalıştırarak otomatik olarak halledebiliriz.) Örneğin bu kesiminde Delphi bize otomatik olarak bir unit ve bununla ilgili sınıfı oluşturacaktır. (Aşağıdaki kodda yapıcı ve yıkıcı kesimler ihmal edilmiştir. (Constructor ve destructor))

```
type
TDSFirst3TierServerModuleClient = class
private
        FDBXConnection: TDBXConnection;
        FInstanceOwner: Boolean;
        FGetHelloCommand: TDBXCommand;
public
        constructor Create(
        ADBXConnection: TDBXConnection); overload;
        constructor Create(
        ADBXConnection: TDBXConnection;
        AInstanceOwner: Boolean); overload;
        destructor Destroy; override;
        function GetHello: string;
end;

function TDSFirst3TierServerModuleClient.GetHello: string;
begin
        if FGetHelloCommand = nil then
        begin
                FGetHelloCommand := FDBXConnection.CreateCommand;
                FGetHelloCommand.CommandType :=
                TDBXCommandTypes.DSServerMethod;
                FGetHelloCommand.Text :=
                'TDSFirst3TierServerModule.GetHello';
                FGetHelloCommand.Prepare;
        end;

        FGetHelloCommand.ExecuteUpdate;
        Result := FGetHelloCommand.Parameters[0].
        Value.GetWideString;
end;
```

Oluşturulmuş kod kesimi, yüksek seviye olan SqlServerMethod bileşenini kullanmaz ama TDBXCommand sınıfı gibi düşük seviye dbExpress uygulama nesnelerini direk olarak çağırabilir.

Bu proxy sınıfını kullanılabilir hale getirmek için, şu anda istemci program üzerinden sunucu taraftaki bu metodu daha dil-dostu bir şekilde çağırabiliriz, yalnız proxy sınıftan bir varlığı oluşturmamız gerekir (ya da bir kere oluşturup onu saklamak). Aşağıdaki kod SqlServerMethod bileşeni ile ilgili olarak yazılan kod ile aynı işlemi yapar.

```
procedure TFormFirst3Tier2009Client.btnHelloClick(
Sender: TObject);
begin
        with TDSFirst3TierServerModuleClient.Create(
        SQLConnection1.DBXConnection) do
        try
                ShowMessage (GetHello);
        finally
                Free;
        end;
end;
```

Bu kod parçası daha öncekine göre biraz daha uzun olabilir, bunun sebebi de herhangi bir parametre istememesindendir, ki bu da dile bağlı kodlamayı daha aza indirgemeyi sağlar. Hala, kullanmaya hazır bir proxy nesnemiz olduğundan, şunu yazabiliriz.

```
ShowMessage (ServerProxyObject.GetHello);
```

--- Veritabansız Bir DataSnap Sunucusunda Oturum ve Thread İşlemleri ---

Eğer DataSnap 2009 yapısını kullanmak için en sık kullanılan metod olan IAppServer arayüzünü direkt olarak kullanırsak, veritabanı içeriği dışında uzaktan metod yürütme işlemlerini çok katmanlı bir şekilde gerçekleştirmek de mümkün olabilir. Aynı teknolojiyi kullanarak IAppServer olmadan da veritabanına erişim ve veritabanı üzerinde işlemler yapmak da mümkün olabilir, eğer yapmak istediğimiz şey sadece sunucudan veri okumak ise. Eğer istemci makinaya sunucudaki veriyi değiştirerek sunucuya tekrar yazmasını istiyorsak, geleneksel metodları kullanmak, ClientDataSet ve DataSetProvider bileşenleri tarafından sağlanan kullanmaya hazır IAppServer'a göre son derece bıktırıcı olabilir.

Her halükarda, ikinci örnekte ben bir çok sınıfı sergileyecek bir minimum bir sunucu oluşturmak istiyorum. İlerleyen bölümlerde bir kaç ilişkili olayı (sunucu bellek yönetimi, sunucu ve istemci bazında thread işlemleri gibi) çözmek için bu oluşturacağım örnek sınıfı kullanacağım.

DSnapMethodServer projesinde yayınlamak istediğim ilk sunucu aşağıdakidir. (2 metodu bulunmakta)

```
{$MethodInfo ON}
type
        TSimpleServerClass = class(TPersistent)
```

```
	public
		function Echo (const Text: string): string;
		function SlowPrime (MaxValue: Integer): Integer;
	end;
{$MethodInfo OFF}
```

İlk metodun codu basitçe inputu yansıtır, son kısmını tekrar eder, ikinci metod da klasik bir şekilde yavaşça hesaplama yapar. Aşağıda 2 metodun da kodu bulunmaktadır.

```
function TSimpleServerClass.Echo(
const Text: string): string;
begin
	Result := Text + '...' +
	Copy (Text, 2, maxint) + '...' +
	Copy (Text, Length (Text) - 1, 2);
end;

function TSimpleServerClass.SlowPrime(
MaxValue: Integer): Integer;
var
	I: Integer;
begin
	// counts the prime numbers below the given value
	Result := 0;
	for I := 1 to MaxValue do
	begin
		if IsPrime (I) then
			Inc (Result);
	end;
end;
```

Yukarıdaki kod kesiminde sunucunun loglaması ile ilgili kodları yazmadım.

Buradaki Sunucu uygulaması, ana form ve 2 adet sunucu taraflı sınıfı içeren tek bir unit'ten oluşmaktadır. Form, geleneksel olan DataSnap sunucu bileşenlerini (1 adet DSServer ve 1 adet DSTCPServerTransport) ve 2 tane de sergilemek istediğimiz sınıflar için kullandığımız DSServerClass bileşeni içerir. Sunucuyu derleyip başlattıktan sonra Delphi'ye yeni istemci uygulamanın SQLConnection bileşenini kullanacak bir proxy sınıfı oluşturmasına izin veririm. Aşağıdaki kod, istemci proxy sınıfına aittir.

```
type
TSimpleServerClassClient = class
	private
		FDBXConnection: TDBXConnection;
		FInstanceOwner: Boolean;
		FEchoCommand: TDBXCommand;
		FSlowPrimeCommand: TDBXCommand;
	public
		constructor Create(
		ADBXConnection: TDBXConnection); overload;
		constructor Create(
		ADBXConnection: TDBXConnection;
		AInstanceOwner: Boolean); overload;
```

```
        destructor Destroy; override;
        function Echo(Text: string): string;
        function SlowPrime(MaxValue: Integer): Integer;
    end;
```

İstemci programda, butonun OnClick olayı, sunucudaki Echo (Yansıma) metodunu çağırır, eğer bir proxy sınıfı nesnesi oluşturulmamışsa oluşturulduktan sonra çağrılır.

```
procedure TFormDsnapMethodsClient.btnEchoClick(
Sender: TObject);
begin
    if not Assigned (SimpleServer) then
        SimpleServer := TSimpleServerClassClient.Create (
        SQLConnection1.DBXConnection);
    Edit1.Text := SimpleServer.Echo(Edit1.Text);
end;
```

Örnekte, örnek metin olarak "Marco" verildiğinde ve butona tıklandığında "Marco" server tarafından çağrıldığında "Marco..arco...co" şekline çevrilir. Bu, IAppServer kullanılmadan, veritabanı erişimi bulunmadan bütünüyle bir sunucu oluşturmaya iyi bir örnektir. Bu metod Delphi'de method yönetme için kullanılan tek teknik değildir, soket bazlı uygulamalarda SOAP da kullanılabilir, ya da 3. parti araçlar da kullanılabilir. Ama bu yöntemin uzaktan veritabanına erişim yeteneği ekstra bir özellik olarak büyük bir artıdır bizim açımızdan.

Bu örneğe odaklanmamın sebeplerinden biri de DataSnap 2009'un bazı ilişkili özelliklerine de açıklık getiriyor olmasıdır. Bunlardan biri, sunucu taraflı nesnelerin istemci tarafındaki proxy'lere veya sunucu metod yürütme işlemi ile ilişkili olmasıdır. Bu, kendi durumunun izini tutan br sınıf nesnesi ile daha iyi bir şekilde gösterilebilir. Tıpkı demo proje içerisindeki 2. sunucu sınıfı gibi.

```
{$MethodInfo ON}
type
    TStorageServerClass = class(TPersistent)
    private
        FValue: Integer;
    public
        procedure SetValue(const Value: Integer);
        function GetValue: Integer;
        function ToString: string; override;
    published
        property Value: Integer read GetValue write SetValue;
    end;
{$MethodInfo OFF}
```

Getter ve Setter metodlar lokal alanı basitçe okur ve yazarlar, ToString fonksiyonu ise hem değeri hem de nesne belirleyicisini tanımlandığı hash kodu ile döndürür.

```
function TStorageServerClass.ToString: string;
begin
    Result := 'Value: ' + IntToStr (Value) +
    ' - Object: ' + IntToHex (GetHashCode, 4);
```

```
end;
```

Bu metodu bir sunucu nesnesinin yaşam döngüsünün nasıl çalıştığını göstermek için yazdım. Sınıfta özellik tanımı, bu nesnenin istemci tarafa sergilenmeyeceğini, sadece sunucuya has bir şey olduğunu belirtiyor. İlişkili proxy'nin arayüzü, özel alanları, yapıcı ve yıkıcı metodları çıkardıktan sonra şu hali alıyor.

```
type
TStorageServerClassClient = class
        public
                procedure SetValue(Value: Integer);
                function GetValue: Integer;
                function ToString: string;
```

Şunu vurgulamak gerekir ki kullandığımız fonksiyonu "Override" olarak işaretlemezsek, sınıf bize aşağıdaki uyarıyı getirecektir.

```
Method 'ToString' hides virtual method of base type 'TObject'
```

Bu örneğin amacı, birden fazla istemci uygulamanın aynı sunucuyu kullanmak istediği zaman neler olacağını göstermek idi. DataSnap 2009 Sunucusunun davranışı, kullanılan DSServerClass bileşeni içerisindeki LifeCycle (Yaşam Döngüsü) string değerine bağlı olarak değişir.

--- Sunucu Nesneleri Yaşam Döngüsü ---

DataSnap 2009 sunucu nesnesinin yaşam döngüsü, ilişkili olduğu DSServerClass bileşeninin ayarlarına bağlıdır. Bu bileşenin LifeCycle adındaki özelliği, 3 adet string türünde değeri barındırır (DSServerClass bileşeni tarafından DSServer nesnesi açıksa, çalışma zamanındaki (runtime) değişiklikleri olmamış sayarak okur)

Session (Oturum) : Sunucu, her istemci soket bağlantısı için yeni bir nesne oluşturur ki bu nesne her istemci için bir sunucu nesnedir. Sunucu nesneleri bağlantı kapandığı zaman serbest bırakılır. Çoklu kullanıcılar, eğer sunucu nesnesi bir data module ise birbirinden bağımsız durumlara ve ayrı veritabanı erişimlerine, belki kendi veritabanı erişim bileşenleri ile sahip olurlar. Bu varsayılan ayarlamadır.

İnvocation (Yürütme) : Sunucu method her yürütüldüğünde yeni bir sunucu nesnesinin oluşturulup sonrasında serbest bırakılacağını belirtir. Bu klasik durumsuz bir davranıştır, sunucuyu ölçülebilir bir halde tutar ve aynı zamanda aynı verinin gidip gelmesini tekrar ve tekrar konu eder.

Server (Sunucu) : Paylaşılan bir sunucu nesnesini, bir singleton'u belirtir. Her kullanıcı, aynı sunucu nesnesini, aynı veriyi kullanacaktır ve bundan dolayı senkronizasyon problemleri olacaktır. (Farklı istemci yürütmeleri, farklı sunucu threadleri ile tetiklenerek çözümlenebilir.) Paylaşılan sunucu nesnesine erişim senkronizasyon teknikleri ile konuma altına alınabilir. (Mesela yeni bir TMonitor kaydı tutularak)

Bu varsayılan ayarları kullanmak dışında, sunucu tarafındaki nesnelerin oluşturulması ve yok edilmesi DSServerClass bileşeninin OnCreateInstance ve OnDestroyInstance olayları tetiklenerek de yapılabilir. Bu sunucu nesneleri havuzlamasında da kullanılabilir.

--- Sunucuyu Başlatan ve Çoklu Oturum Açan İstemci ---

Uygulamalı bir örnek olarak, DSnapMethods projesi tek bir istemci uygulamadan birden çok istemci bağlantının oluşturulmasına müsade etmketedir. (Birden çok uygulama kullanmak da aynı sonucu sağlar). Form üzerinde SQLConnection bileşenii bulunduran birden çok varlık oluşturulabilir, bunu en başta oluşturulan bir istemci proxy'nin lokal nesnesi içerisinde saklayabilirsiniz. İstemci sadece çoklu istemci bağlantı oluşturmakla kalmaz, aynı zamanda sunucu programı belirlenen bir yaşam döngüsü ayarlaması ile başlatmak da mümkündür. Bunu başarmak, istemci ve sunucu uygulamalar aynı bilgisayarda olduğundan oldukça kolaydır.

Bunu başarmak için sunucunun ana formunu belirten unit içerisine DSServerClass bileşeninin LifeCycle özelliğini belirlemesi için global bir değişken ekleriz.

```
var
ParamLifeCycle: string;

procedure TFormDsnapMethodsServer.DSServerClass2GetClass(
DSServerClass: TDSServerClass;
var PersistentClass: TPersistentClass);
begin
        DSServerClass2.LifeCycle := ParamLifeCycle;
        Log ('LifeCycle: ' + DSServerClass2.LifeCycle);
        PersistentClass := TStorageServerClass;
end;
```

ParamLifeCycle adlı değişkenin ilk değerini proje dosyası kaynak kodunun başında sunucu uygulamasının komut satırında aldığı parametrelerle aşağıdaki gibi belirleyebiliriz.

```
begin
        if ParamCount > 0 then
                ParamLifeCycle := ParamStr(1);
        Application.Initialize;
```

Sunucuda bulunan bu kodla, istemci uygulamanın ana formunda (burada bağlantı yok, bağlantı ikincil formlarla ayarlanıyor) bir RadioGroup oluşturularak aşağıdaki kod yazılır.

```
object rgLifeCycle: TRadioGroup
        ItemIndex = 0
        Items.Strings = (
        'Session'
        'Invocation'
        'Server')
end
```

Butona tıklandığı zaman, istemci program güncel değeri alır ve bu değerleri sunucuya parametre olarak gönderir (Sunucuyu iki kez çalıştıramayacağımızı, aynı makinadan aynı anda 2 uygulamalayla aynı portu kullanarak aynı soketi dinleyemeyeceğimizi unutmamak gerekir.). İlgili kod aşağıdadır.

```
procedure TFormDsmcMain.btnStartServerClick(
Sender: TObject);
```

```
var
	aStr: AnsiString;
begin
	Log (rgLifeCycle.Items[rgLifeCycle.ItemIndex]);
	aStr := 'DsnapMethodsServer.exe ' +
	rgLifeCycle.Items[rgLifeCycle.ItemIndex];
	WinExec (PAnsiChar (aStr), CmdShow);
end;
```

Şu anda istemci uygulamanın ana formu ikincil formları oluşturacak, OnClose olayının tetiklenmesine bağlı olarak sunucuya yapılacak özelleşmiş bağlantıları kapatacak ve bu şekilde oluşturulan formları yok edecek bir butona sahip olmuştur. O an kullanılan istemci formların durumlarının loglanması için ayrı bir buton kullanılmalıdır. Bununla ilgili kod aşağıdadır.

```
procedure TFormDsmcMain.btnUpdateStatusClick(
Sender: TObject);
var
	I: Integer;
begin
	for I := 0 to Screen.FormCount - 1 do
		if Screen.Forms[I].ClassType = TFormDsmcClient then
			Log (IntToStr (I) + ': ' +
				Screen.Forms[I].ToString);
end;
```

İkincil formların birinden ToString fonksiyonu çağrıldığı zaman, bu public türdeki ToString metodunu çağırarak bağlandığı sunucu nesnesinin durumunu döndürür.

```
function TFormDsmcClient.ToString: string;
begin
	InitStorageServer;
	Result := StorageServer.ToString;
end;
```

İlk uygulama örneğinde, varsayılan Oturum yaşam döngüsü ile bir sunucu oluşturdum, 2 istemci form açtım, değerlerini 3 ve 4 olarak atadım, ve bütün durumunu sorduğumda şöyle bir sonuç çıktı

```
Session
1: Value: 3 - Object: 1C38400
2: Value: 4 - Object: 1C384E0
```

İkinci uygulamada İnvocation yaşam döngüsü ile yaptım ve bütün durumu iki kere istediğim zaman şöyle bir sonuç çıktı

```
Invocation
1: Value: 0 - Object: 1D185B0
2: Value: 0 - Object: 1D18490
1: Value: 0 - Object: 1D185C0
2: Value: 0 - Object: 1D185D0
```

Burada her uygulama için yeni bir nesne elde ettiğimizi ve bu nesnenin durumunun her zaman 0 olduğunu unutmamak gerekir. (Her yürütmeden sonra nesne yok olduğunda bütün ayarlar da otomatik olarak yok olur). Bu olay, sadece durumsuz işlemlerde mantık kazanır.

En sonunda ilk uygulamadaki gibi Server yaşam döngüsü ayarlarıyla 3 ve 4 değerlerini atadım, ve bu sefer her istemci formu, aynı sunucu nesnesini kullandı ve son değer ataması ile kullanılmış oldu. Şöyle bir sonuç çıktı ortaya

```
Server
1: Value: 4 - Object: 1E08490
2: Value: 4 - Object: 1E08490
```

Diğer bir deyişle, alıştırma gösterdi ki teori doğruymuş. Demoda yaşam döngüsü konfigürasyonunu incelerken gördük ki, istemci lokal sunucuyu ihtiyaç duyulduğunda başlatabiliyor ve sunucuya eş zamanlı olarak çoklu bağlantı kurabiliyor.

--- Eski DataSnap Yapısını Yeni Sisteme Geçirmek ---

DataSnap 2009 ile bazı alternatiflere yöneldikten sonra, müsaade edin biraz eskiye gideyim, en çok kullanılan klasik senaryoya, çok katmanlı veritabanı uygulamasına. Hali hazırda yeni bir DataSnap veritabanı uygulamasının nasıl oluşturulacağına ilişkin adımları gördük. Şimdi göreceğimiz şey ise eski bir DataSnap (ya da MIDAS) uygulamasını bu mimariye göre şekillendirmek.

Uygulamalı bir örnek olarak, Mastering Delphi 2005'te bahsettiğim ThinPlus uygulamasını yeni yapıya geçireceğim. Çünkü hem DataSnap'in bir kaç yeteneğini göstermiş olacağım, hem de üzerinde tamamen çalışabileceğim bir proje. Bu uygulama ile aynı zamanda COM sunucusuna soket kullanarak yürütme gönderen bir istemciyi nasıl saf soket tabanlı mimariye çevireceğime de dikkat etmek gerekecek. Yeni örnek, sunucu ve istemci dosyalarıyla birlikte ThinPlus2009 klasörü içerisinde olacak

Unutmamak gerekir ki, eski modeli yeni DataSnap modeline çevirmek ilginç bir öneridir ama zorunlu değildir. Eski mimarinin istemci ve sunucuları da Delphi 2009 içerisinde derlenip çalıştırılabilir.(Program Mastering Delphi 2005'te olduğu gibi Mastering Delphi 7 içinde de bulunmaktadır.Ben burada bazı özeliiklerini kullanacağım, bu kitaplar DataSnap ve MIDAS hakkında projenin sahip olduğu pek çok özellikten bahsetmektedir.)

--- Sunucuyu Yeni Sisteme Geçirmek ---

Sunucu projeyi yeni sisteme çevirmek için şu adımları takip ettim.

- AppsRDM adı verilen uzaktan yönetim data module unit dosyası içerisindeki initialization bölümünü kaldırdım. Kaldırılan kod, TComponentFactory sınıfının yapıcısını (constructor) çağırmak içindi.
- Aynı data module unit dosyası içerisindeki TAppServerPlus sınıfının UpdateRegistry sınıf metodunu da kaldırdım.
- Bu noktada uzaktan yönetim data module içerisinde COM ve ActiveX ile ilişkili uses kesimlerini de kaldırdım (ComServ, ComObj, VCLCom ve StdVcl)

Sonra IAppServerPlus arayüzünün (interface) referansını, ki proje tarafından sunucu metodlarını desteklemek için kullanılıyor, kaldırdım. (Bu arayüz, proje tür kütüphanesinde tanımlıdır.)

Tür kütüphanesini (type library) ve RTDL dosyalarını (projeyi Delphi 2009 içinde açınca otomatik oluşuyor) hem projeden hem de disk üzerinden sildim. Ayrıca tür kütüphanesi unit'ini kullanan kodlardaki uses kısmından da sildim tür kütüphanesini

Uzaktan yönetim data modül sınıfında tek sunucu metodu olan Login metodunu protected bölümünden alarak, public tanımlamalara **safecall** modifikasyonnunu kaldırarak taşıdım. Zaten TRemoteDataModule sınıfı $MethodInfo açık bir halde derlendiği için bu tanımı projenin unit dosyasında bulundurmaya gerek yok.

En sonunda programın ana formuna geleneksel bileşen üçlüsünü ekledim (sunucu, sunucu sınıfı ve sunucu taşıma), bunları bağladım ve sunucu sınıfı bileşeninin OnGetClass olayı tetiklemesinde TAppServerPlus döndürdüm.

Sadece bunları yaparak eski modellemeden Delphi 2009'daki yeni DataSnap modeline güncelledim. Çok gibi görünse de aslında çok çabuk yapılmaktadır. Şimdi de istemci uygulamaya bakacağız ve orada da birtakım işlemleri gerçekleştireceğiz.

--- İstemciyi Güncellemek ---

İstemci bölümü yeni sistme geçirmek, Delphi 2009'da sunucudan daha kolaydır. Ana adım bağlantı bileşenlerini kaldırmak ve bunları SQLConnection ve DSProviderConnection ile yer değiştirmek, sonrasında da yeni uzaktan yönetim bileşenini ClientDataSet bileşenine bağlamaktır. (Bu uygulama içerisinde 3 tane var kullanıcıların bağlantı çeşitliliğine bağlı olarak)

Değiştirmek durumunda kalacağım tek kod parçası Login sunucu metodunun çağrıldığı koddur. Bu, bağlantı bileşenindeki OnAfterConnection'da olup, bunu SQLConnection bileşeninin ilgili olayına taşıyacağım, Aşağıdaki kod bu işe yaramaktadır.

```
procedure TClientForm.SQLConnection1AfterConnect(
Sender: TObject);
begin
        // was: ConnectionBroker1.AppServer.
        //          Login (Edit2.Text, Edit3.Text);
        SqlServerMethod1.ParamByName('Name').AsString :=
        Edit2.Text;
        SqlServerMethod1.ParamByName('Password').AsString :=
        Edit3.Text;
        SqlServerMethod1.ExecuteMethod;
end;
```

Bu çağrının yaptığı iş, istemcinin bağlantı bilgilerini sunucuya göndermektir. Sunucu bilgiyi doğrular ve başarılı ise sağlayıcıya (provider) verisini gösterir. Şifre kontrolü önemsiz, ama yaklaşım ilginçtir. Bu da sunucudaki Login metodudur.

```
procedure TAppServerPlus.Login(
const Name, Password: WideString);
begin
        if Password <> Name then
```

```
                raise Exception.Create (
                'Wrong name/password combination received');
        ProviderDepartments.Exported := True;
        ServerForm.Add ('Login:' + Name + '/' + Password);
end;
```

Burada sunucu tarafında oluşturulan istisnai durumun (exception) sunucu tarafında görüntülendiğini Figür 3'te görebilirsiniz.

--- ThinPlus2009'un Gelişmiş Özellikleri ---

Daha önce bahsettiğim adımları gerçekleştirerek, ThinPlus istemci ve sunucu uygulamalarını DataSnap 2009 yapısına güncelledim. Çeşitli özelleştirmelerle daha kompleks bir yapıya dönüştürdüm. Bunlara örnek olarak, veri paketlerinin elle gidip gelmesi, master/detay yapısı kullanımı, parametrik sorguları çalıştırma, veri paketlerinin yanında ekstra veri transferi, özel uzaktan bağlantı ile login işlemleri benim gözlemlediklerimdir.

Bu özelliklere kısaca göz attığımızda daha önce DataSnap kullanmamış insanlara, DataSnap'in gücünü göstermek açısından son derece yararlı olabilir. Daha önce yazmış olduğumuz kodları ne kadar kolay bir şekilde yeni mimariye çevirebilmemiz gibi. Sunucu uygulama sağlayıcıya bağlı olan master data set, bunu refere edecek data source ve data source'a refere edecek detay data source gibi ayarlamaları barındıran bir master/detay yapıyı kullanıyor.

```
object ProviderDepartments: TDataSetProvider
        DataSet = SQLDepartments
end
object SQLDepartments: TSQLDataSet
        CommandText = 'select * from DEPARTMENT'
        SQLConnection = SQLConnection1
end
object DataSourceDept: TDataSource
        DataSet = SQLDepartments
end
object SQLEmployees: TSQLDataSet
        CommandText =
        'select * from EMPLOYEE where dept_no = :dept_no'
        DataSource = DataSourceDept
        Params = <
                item
                Name = 'dept_no'
                ParamType = ptInput
        end>
        SQLConnection = SQLConnection1
end
```

İstemci tarafında program sağlayıcı ile bağlanan ilk ClientDataSet 'i ve burada ilkindeki özel bir data set alanına refere eden ikinci bir ClientDataSet kullanıyor.

```
object cds: TClientDataSet
```

```
        FetchOnDemand = False
        PacketRecords = 5
        ProviderName = 'ProviderDepartments'
        RemoteServer = DSProviderConnection1
        object cdsDEPT_NO: TStringField...
        object cdsDEPARTMENT: TStringField...
            ...
        object cdsSQLEmployees: TDataSetField
            FieldName = 'SQLEmployees'
        end
    end
    object cdsDet: TClientDataSet
        DataSetField = cdsSQLEmployees
    end
```

2 ClientDataSet bileşeni içerisindeki veri, 2 ayrı DBGrid içerisinde gösterilmektedir. Programın her veri paketinde sadece 5 kaydı nasıl alıp verdiğini (PacketRecords özelliğinde belirtildi), ilk paketten sonra alıp verme işlemini kestiğini (FetchOnDemand özelliği False olarak tanımlandı), ve gridin tam dolmadığını dikkatle incelemek gerekir. Bununla ilgili ekran görüntüsünü Figür 4'te görebilirsiniz.

Takip eden veri paketleri, form üzerinde ilgili buton tıklanarak manuel olarak alınıp verilebilir.

```
procedure TClientForm.btnFetchClick(Sender: TObject);
begin
    btnFetch.Caption := IntToStr (cds.GetNextPacket);
end;
```

Butonun Caption özelliğinde program her paketle kaç kaydın alınıp veridiğini gösterir. Bu yeteri kadar kayıt varsa 5'tir, yoksa olduğu kadardır, bütün kayıtlar alındığında da 0'dır. Her alıp verme işleminde grid daha fazla veri gösterir ve buna bağlı olarak ScrollBar'ı da otomatik olarak düzenlenir. Bu vakte kadar kaç kayıt alındığını ayrıca btnRecCount butonundan da öğrenme imkanımız vardır.

İstemci program Query adındaki buton tıklanarak gösterilen ve farklı bir dataset'e sahip olan 2. bir forma sahiptir. Bu ClientDataSet bileşeni serverda tanımlanan parametrik bir sorgu ile bağlanır, şöyle ki;

```
object SQLWithParams: TSQLDataSet
    CommandText =
    'select * from employee where job_code = :job_code'
    Params = <
    item
        DataType = ftString
        Name = 'job_code'
        ParamType = ptInput
        Value = 'Eng'
    end>
    SQLConnection = SQLConnection1
end
```

İstemci programda bir adet listbox vardır, bu listbox sunucuya parametre olarak gönderilecek ve tasarım zamanında doldurulacak Departman İsimleri ile doludur. Şunu

unutmamak gerekir ki, en başta parametrelerin tanımlamalarını güncelleyecek bir kod yazmak gerekir, bu işlem ClientDataSet bileşeni için ilgili bileşen editörü komutu kullanarak tasarım zamanında yapılır. İstemcinin uzaktan parametrik sorgu çalıştırması için gerekl olan çağrı şu şekildedir.

```
procedure TFormQuery.btnParamClick(Sender: TObject);
begin
        cdsQuery.Close;
        cdsQuery.Params[0].AsString := ComboBox1.Text;
        cdsQuery.Open;
...
```

Sunucu tarafında, sorgu OnGetDataSetProperties olayında çalıştırıldığı zaman sağlayıcı geri dönen paket içerisine ekstra veri ekler.

```
procedure TAppServerPlus.
ProviderQueryGetDataSetProperties(Sender: TObject;
DataSet: TDataSet; out Properties: OleVariant);
begin
        Properties := VarArrayCreate([0,1], varVariant);
        Properties[0] := VarArrayOf(['Time', Now, True]);
        Properties[1] := VarArrayOf([
        'Param', SQLWithParams.Params[0].AsString, False]);
end;
```

Unutmamak gerekir ki, variant türünde dizi parametrelerinin kullanımı hala çalışmakla birlikte DataSnap 2009 teknolojisi ile ulaşımı artık farklıdır. İstemci tarafında, btnParamClick olay tetiklemesine fazladan 2 satır daha kod yazarak ekstra bilginin dönen paketle birlikte gelmesini sağlayabiliriz.

```
 Caption := 'Data sent at ' + TimeToStr (
TDateTime (cdsQuery.GetOptionalParam('Time')));
Label1.Caption := 'Param ' +
cdsQuery.GetOptionalParam('Param');
```

Eski versiyondan DataSnap 2009 yapısını geçirme yapılırken bir kaç özellik daha anlatılabilir, ama ThinPlus2009 programı (Delphi 6'da yazılan haline göre baya değişti) benim amaçlarımı gerçekleştirmem için şu anda yeterlidir. Bu amaç ise, DataSnap'in gücü ve COM tabanlı bir uygulamanın bu teknolojiye ne kadar çabuk güncellenebildiğidir.