**Temel veritabanı nesneleri hakkında küçük hatırlatmalar**

**ResultSet (Sonuç kümesi):**

Bir seçme işlemi gerçekte veri tabanın da olmayan bir tablo döndürür. Bu tablo bazen 1 satır ve 1 sütundan oluşur.

**Constraint (Kısıtlayıcı):**

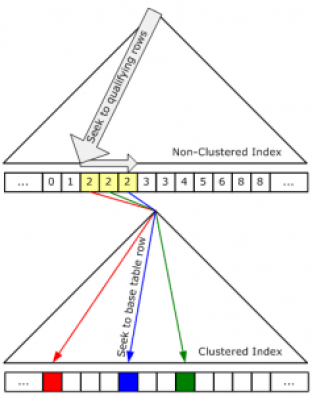
Herhangi bir alan için girilebilecek verileri zorlayıcı kurallara Kısıtlayıcı denir. Primary Key, Uniqe Key, Foreign Key gibi. Veritabanına kullanıcının keyfi değerler girmesi önlenmiş olur.

**Indeksler:**

Bir kaç yüz kayıttan oluşan bir tablo üstünde kayıt arama işlemi, VTYS (veritabanı yönetim sistemi) için oldukça basit bir işlemdir ancak bu sayı milyonlara çıktığında doğru verilere erişmek için fazla zamana ihtiyaç duyulur. Bu nedenle verileri çeşitli özelliklerine göre organize edip sıralayacak ve bu sayede daha hızlı erişmemize olanak tanıyacak yapılara ihtiyaç duyarız. Indexler, kayıtları fiziksel olarak sıraya sokuyorsa CLUSTERED INDEKS, fiziksel olarak sıraya koymuyorsa NONCLUSTERED INDEKS adını alır.

**Clustered INDEKS:**

Verilerin kendisinin index key (indexin verileceği kolon) değerine göre fiziksel olarak alfabetik sıralandığı index türüdür. Clustered Index bir tabloda maksimum bir tane olabilir. (Örnegin telefon defterinde yapılan bir arama)

**[](https://i0.wp.com/koddit.com/wp-content/uploads/2014/08/indeksler.png)**

**Non-Clustered Index:**

Non-Clustered Index mimarisinde fiziksel bir sıralama söz konusu değildir. Index değeri, Clustered Index’in aksine verinin kendisi değil pointer değeridir. Bu pointer değeri verinin nerede olduğunu gösterir. Non-clustered Index bir tabloda maksimum 999 tane olabilir.(Sql Server 2008 için) (Örneğin Kitap’ta sayfa numaralarına göre yapılan bir arama )

|  |  |
| --- | --- |
| **1** | **CREATE NONCLUSTERED INDEX IX\_LastName\_FName\_MName on Person.Person (LastName, FirstName, MiddleName)** |

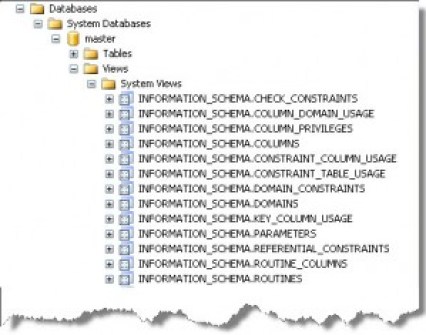
**Schema yaratma:**

Aynı isimde tablo, stored procedure gibi nesneleri farklı şemalar içerisinde yeniden tanımlayabilmek mümkündür.

|  |  |
| --- | --- |
| 1  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** | **Create schemaPinar**  **create table pinar.depertmant**  **(**  **ID int identity (1,1),**  **Ad varchar (50)**  **)**  **Select \* from pinar.depertmant** |

**View:**

Bazen tabloları olduklarından farklı gösterecek filtrelere ihtiyaç duyarız. View’lar birden fazla tablodan daha kolay veri almak için kullanılır. Program içinde yazılan kod daha basit olur. Yine tablolardaki bazı sütunlar gizlenebilir. Oluşturulan View’e bu sütun koyulmayabilir. View’ler bir çeşit istatistik olarak da kullanılabilir.

**[](https://i0.wp.com/koddit.com/wp-content/uploads/2014/08/vieww.jpg)**

View’ler sonuç itibari ile gerçek anlamdaki tablolar değildir. Bundan dolayı View’ler üstünde ekleme veya güncelleme yaparken çeşitli kısıtlamalar vardır. Aslında bize direk select sorgusunu getirir. Update, Insert, Truncate işlemleri yapamazsın. View oluşturulduğunda, yapılabilecek herhangi bir hata da geri alınabilir. Ancak her View den select çektiğinde db.de performans sorunu yaratabilir.

|  |  |
| --- | --- |
| **1** | **select \* from HumanResources.vEmployee** |

şeklinde gösterilir.

**Stored Procedure:**

Genellikle veritabanı uygulamalarında veri erişim katmanı olarak kodlanır. Dışardan parametre alabilirler ve dışarıya parametre resultset döndürebilirler.

1.Datalar üzerinde yapacağın işlemleri saklı yordamlarda saklayabilirsin . Her gece kendisinin çalışmasını istediğin kod parçalarını Strod procedure yazarak sağlayabilrsin.  
2.Jop’larla çalışma sürelerini ayarlayabilirsiniz.  
3.Stored’lerin execute edilmesi gerekir. Tekrardan select \* from sp yazmanıza gerek yoktur, zaten içinde select cümleleri saklıdır.

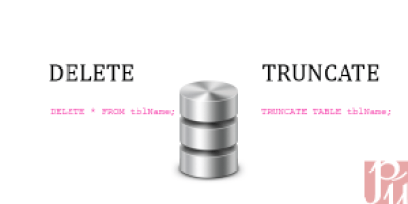
**Function:**

0-1 ya da yes-no şeklinde değerler döndürür .İstenildiği takdir de Table ‘da döndürebilir. Parametre alabilir (@PersonID int) gibi .

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **ALTER FUNCTION [dbo].[ufnGetContactInformation](@PersonID int)**  **RETURNS @retContactInformation TABLE**  **(**  **— Columns returned by the function**  **[PersonID] int NOT NULL,**  **[FirstName] [nvarchar](50) NULL,**  **[LastName] [nvarchar](50) NULL,**  **[JobTitle][nvarchar](50)NULL,**  **[BusinessEntityType] [nvarchar](50) NULL**  **)** |

**Truncate ile Delete işlemleri arasındaki farklar**

* TRUNCATE komutu ile parçalı silme yapılamazken DELETE komutu ile WHERE clause kullanılarak parçalı silme yapılabilmektedir.
* DELETE komutu ile yapılan kayıt silme işleminde her silinen kayıt Transaction Log da kayıt bazında loglanmaktadır. TRUNCATE de ise loglama işlemi kayıt bazında yapılmamaktadır. Bu yüzden TRUNCATE komutu DELETE komutuna oranla büyük tablolarda inanılmaz hızlıdır.

**[](https://i0.wp.com/koddit.com/wp-content/uploads/2014/08/ek.png)**

* Eğer silme yapılan tabloda identity column var ise,  TRUNCATE işleminden sonra bu kolon 1 den başlamakta, DELETE işleminden sonra ise kaldığı yerden devam etmektedir.
* Foreign key içeren tablolarda TRUNCATE işlemi yapılamaz. Çünkü Foreign key’e bağlı olan kayıtlarında silinebilmesi için table’ın trigerlanması gerekmekte ama TRUNCATE işleminde daha öncede söylediğim gibi loglama işlemi gerçekleşmemektedir. Genede TRUNCATE kullanmak istiyorsanız önce Foreign key in kaldırılması sonra TRUNCATE in yapılması ve daha sonrada Foreign Key in tekrar tanımlanması gerekmektedir.
* Foreign key içeren tablolar DELETE ile silinmek zorundadır ya da performanslı olmasını istiyorsanız işlemi DROP-CREATE şeklinde gerçekleştirebilirsiniz.

**Primary Key oluşturma:**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **Create table Musteri**  **(**  **MusteriID int identity(1,1) not null,**  **Ad nvarchar(50),**  **Soyad nvarchar(50)**  **Constraint PK\_Musteri Primary key clustered (MusteriID ASC)**  **)** |

**Foreign Key oluşturma:**

* Hareket tablolarından özet tablolara referans alır.
* Kendisine baglı özet tabloların primerykeylerini tutar.
* Foreign key hareket tablolarıdır,bunlar verilerin çok olduğu sürekli değişimin olduğu tablolardır. (Siparis Tablosu gibi)

Sonradan foreign key eklemek isterseniz yani tablo oluştuktan sonra,

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19** | **CREATE  table Musteri**  **(**  **MusteriID int identity(1,1) not null,**  **Ad nvarchar(50),**  **Soyad nvarchar(50)**  **Constraint PK\_Musteri Primary key clustered (MusteriID ASC)**  **)**  **Create table Siparis**  **(**  **SiparisID INT identity(1,1) not null,**  **MusteriIDINT not null,**  **Primary key (SiparisID),**  **Constraint PK\_Musteri FOREIGN KEY (MusteriID)**  **REFERENCES Musteri(MusteriID)**  **)**  **ALTER TABLE dbo.Siparis ADD CONSTRAINT FK\_Musteri FOREIGN KEY (MusteriID)**  **REFERENCES Musteri(MusteriID)** |

**Unique Key:**

Bir alanı Unique yaptığınızda onu eşsiz yapmış olursunuz, mesela email alanını unique yaptığınızda ikinci kez aynı emaili eklemek istediğinizde veritabanı hata verecektir. Her eklenen email bir kez eklenebilir ve veritabanında böylelikle tektir.

Örneğin müşteri kodu P423423 şeklinde string bir ifadeniz olsun, siz bunun tekil (primary key )olmasını istiyorsananız, bu bir string olduğu için sorun yaratabilir, ama Unique key derseniz sıkıntı çıkmaz .  
PR – Key’ide Unique gibidir bir alani eşsiz yapar ama farkı bir tabloda sadece bir tane primary key olmasıdır.  
Null değerlerde alabilir. Örneğin müşteri tablosundaki TCKN gibi uzun bir alana primery key de koymak istemiyorsanız, o zaman UK yapabilrsiniz.

**Sub Query:**

Sub Query yani sorgu içinde sorgu, içteki sorgunun dışta olan sorguya değer üretmesidir. BO tarafında kullanımı daha fazladır .

|  |  |
| --- | --- |
| **1**  **2**  **3** | **select \* from**  **(select TOP 10 \* FROM AdventureWorks2008R2.Production.Product) s**  **WHERE s.Color is null** |

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **select \* from Sales.Customer c inner join Person.Person p on c.PersonID = p.BusinessEntityID**  **inner join**  **(selectCustomerID,**  **TotalRevenue = SUM (LineTotal)**  **from  Sales.SalesOrderDetailSD**  **inner joinSales.SalesOrderHeaderSO**  **ON SD.SalesOrderID = SO.SalesOrderID**  **Group byCustomerID**  **having SUM (LineTotal) >5000)x**  **on x.CustomerID = c.CustomerID** |

Sub – Query’ler performans açısından gerekmediği durumlarda fazla tercih edilmezler, bunun yerine Temp Table kullanmak daha iyi olur .

**Temp Table:**

Geçici tablolar çalışma anında oluşturulur ayrıca normal bir tabloda yapabileceğimiz tüm işlemleri temp tablolarda da yapabiliriz. Temp Tablolar tempdb veritabanında saklanırlar.

**[](https://i0.wp.com/koddit.com/wp-content/uploads/2014/08/Types-of-Temporary-table.png)**

**1) Local Temp Table**

Local Temp tablolar sadece tabloyu oluşturan kullanıcının, oluşturmak için kullandığı connection üzerin de geçerlidir. Kullanıcı bağlantıyı kapattığı anda otomatik olarak silinir. Local Temp tablo tanımlamak için tablo isminin önüne # işareti koymamız yeterlidir.

**2) Global Temp Table**

Global olarak temp tablo tanımlamak için ## karakterlerini tablo isminden önce eklememiz yeterlidir. Global temp tablolar bir connection içinde tanımlandığı anda, normal tablolar gibi tüm kullanıcılar ve connectionlar içinde kullanılabilir hale gelirler. Global tabloyu oluşturduğunuz connection kapatılana kadar bu tablo var olmaya devam eder.

**Local Temp Table örneği:**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16** | **selectCustomerID,**  **TotalRevenue = SUM (LineTotal)**  **INTO #CustomerTotalRevenue**  **from  Sales.SalesOrderDetailSD**  **inner joinSales.SalesOrderHeaderSO**  **ON SD.SalesOrderID = SO.SalesOrderID**  **Group byCustomerID**  **having SUM (LineTotal) > 5000**  **SELECT p.\* fromSales.Customerc**  **inner joinPerson.Personp**  **onc.PersonID=p.BusinessEntityID**  **inner join**  **#CustomerTotalRevenue x**  **on x.CustomerID = c.CustomerID** |

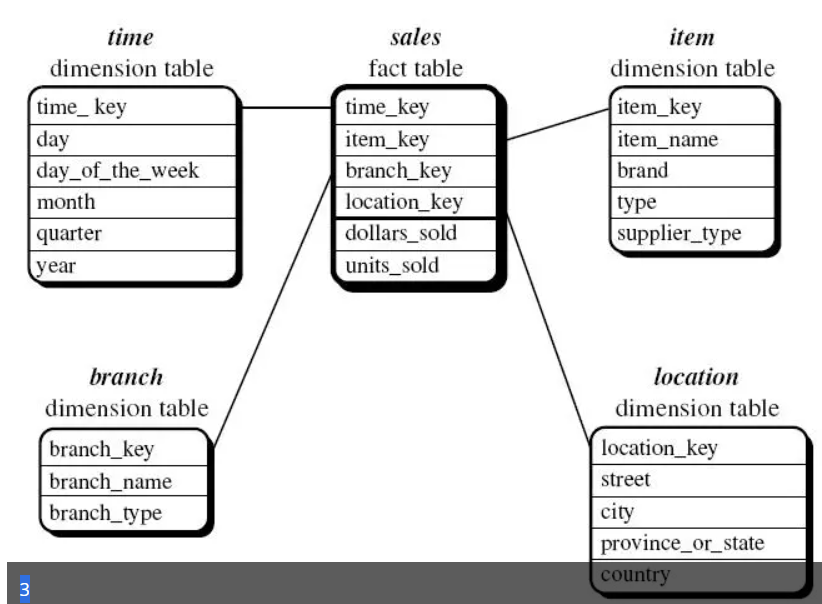
**Temp Table oluşturmanın bir diğer yöntemi ise** tek sefer kullanım hakkı olan Memory’de bir Table yaratmak.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34**  **35**  **36**  **37**  **38** | **;WITH CustomerTotalRevenue AS (**  **selectCustomerID,**  **TotalRevenue = SUM (LineTotal)**  **from  Sales.SalesOrderDetailSD**  **inner joinSales.SalesOrderHeaderSO**  **ON SD.SalesOrderID = SO.SalesOrderID**  **Group byCustomerID**  **having SUM (LineTotal) >5000**  **),**  **top5 AS**  **( SELECT TOP 5 \* from CustomerTotalRevenue**  **)**  **DECLARE @CustomerTotalRevenue TABLE**  **(**  **CustemerID int,**  **TotalRevenue decimal (18,5)**  **)**  **INSERT INTO @CustomerTotalRevenue**  **selectCustomerID,**  **TotalRevenue = SUM (LineTotal)**  **from  Sales.SalesOrderDetailSD**  **inner joinSales.SalesOrderHeaderSO**  **ON SD.SalesOrderID = SO.SalesOrderID**  **Group byCustomerID**  **having SUM (LineTotal) >5000**  **–DECLARE @CustomerTotalRevenue TABLE**  **–(**  **–CustemerID int,**  **–TotalRevenue decimal (18,5)**  **–)**  **Once tabloyu declare etmeliyiz daha sonra burayi kaldirip tum sorguyu calistirmaliyiz.**  **Select \* from @CustomerTotalRevenue** |

**Veri Ambarı Şemaları**

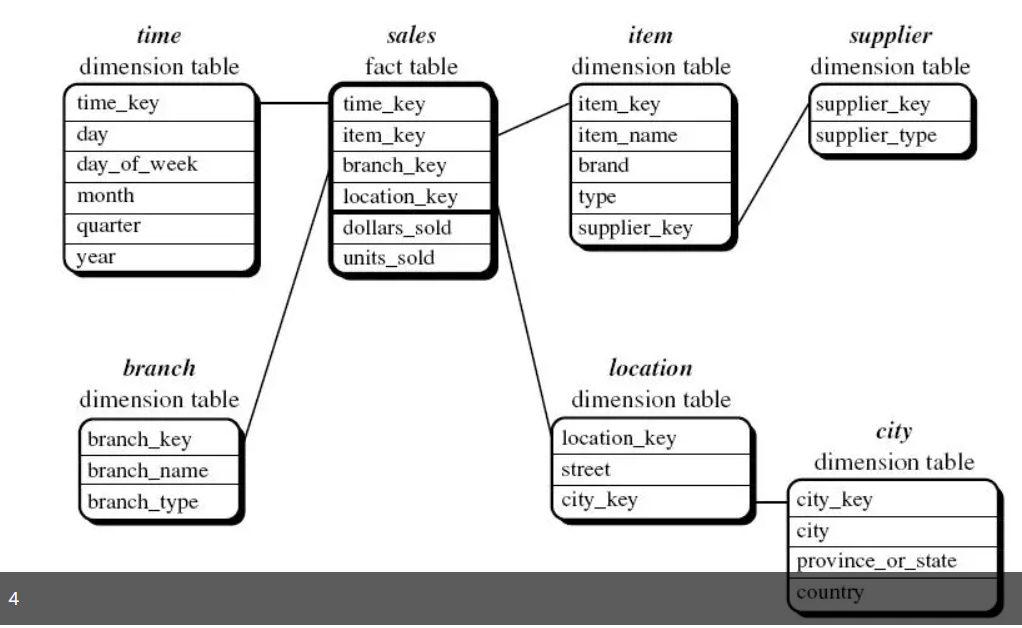
**1) Star SCHEMA**

* Bu şema yapısı en sık tercih edilen şema modelidir. Merkezde konumlandırılmış bir fact table ve onun etrafını sarmış dimension table’ların bir yıldız şeklini anımsatmasından dolayı da Star Scheme (Yıldız Şema) adını almıştır.
* Bu şema yapısında fact table ‘a keylerle bağlanmış olan dimension tablelar demormalize edilmiştir.
* Fact tablolarda, dimension tabloların primery keyleri ve sayısal değerler tutulur.
* Her dimension tablodan fact tabloya bir bağlantı gelir.



**2) Snowflake SCHEMA**

* Bu şema yapısındaysa; merkezde konumlandırılmış bir fact table ve onun etrafını sarmış dimension table’lara bağlanmış başka dimension table’ların kar tanelerine benzer bir şekil oluşturmasından dolayı Snowflake Schema denmiştir.
* Bu şema yapısında fact table ‘a keylerle bağlanmış olan dimension tablelar normalizasyon kurallarına göre dizilmişlerdir, zaten dimensionlara bağlı başka dimensionlar olması da normalizasyon kurallarına uymak içindir.
* Özet tablonun da bir özetini çıkarıyoruz. Buna da bir foreıgn key tanımlıyoruz .
* Sorgu performansını artırır.



Normalizasyon Nedir?

Normalizasyonun iki temel amacı vardır. Veri tabanında veri tekrarlarını ortadan kaldırmak ve veri tutarlılığını (doğruluğunu) artırmak.

Normalizasyon, veri tabanlarına seviyelerle (normal formlar) uygulanır. Bir veri tabanının bu normal formlardan herhangi birine uygun olduğunu söyleyebilmek için, söz konusu normal formun tüm kriterlerini eksiksiz yerine getiriyor olması şarttır.

Başarılı bir şekilde uygulandığında normalizasyon işlemi veri tabanının süratini büyük oranda artırır. Veri tabanının sabit diskteki boyutunu azaltır. Ayrıca veri tutarlılığını artırarak veri tekrarlarını engeller. Bilmem, özellikle güncelleme ve silme işlemlerinde ortaya çıkabilecek aksaklıkları minimize ettiğini söylemeye gerek var mı?

Normal Formlar

Basitçe tanımlamak gerekirse, normal formlar normalizasyon seviyeleridir. Bu seviyeler gereksiz veri tekrarlarını ne derecede engellediği ve tutarlılığı ne kadar sağladığına bağlı olarak derecelendirilir. Seviye yükseldikçe veri tutarlılığı artar, veri tekrarı düşer.

Normalizasyon seviyeleri 1NF (Birinci Normal Form), 2NF, 3NF, BCNF(Boyce-Codd Normal Form, 3.5NF'de denir), 4NF şeklinde adlandırılır ve yukarı doğru devam eder. Ancak daha yukarı normalizasyon seviyeleri çok nadiren kullanılır çünkü çoğu zaman uygulanması mümkün olmayabilir.

Konuyu detaylandırabilmek için bir veri tabanı oluşturalım ve normalizasyonunu yapalım. Tabloda bir teknik destek firmasının çalışanları, servis araçları, servis soförleri ve servis verilen semtler bulunsun. Her bir şoför tek araç ile semt bazında servis yapmaktadır. Örneğin şoför "Ahmet", teknik elemanları (çalışanları) "Toyota" araçla, "Levent", "Etiler" ve "Ulus" semtlerindeki destek çağrılarına götürmektedir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Calisan** | **Soyad** | **Sofor** | **Arac** | **Semt** |
| Orçun | Yılmaz | Ahmet | Toyota | Levent, Etiler, Ulus |
| Metin | Seyyar | Mehmet | Honda | Bakırköy, Ataköy, Yeşilköy |
| Metin | Seyyar | Tolga | Ford | Kandilli, Beylerbeyi, Kuzguncuk |

Çoğu programcı için veri tabanı tasarımı burada biter. Cılkını çıkartmaya gerek yok, değil mi? Bu arkadaşlara tavsiyem: Wordpress'ten şaşmayın, sizi o paklar...

İşin doğrusu, veri tabanımızı oluşturmaya henüz başladık.

1NF (1. Normal Form)

Bir veri tabanının 1NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

* Aynı tablo içinde tekrarlayan kolonlar bulunamaz,
* Her kolonda yalnızca bir değer bulunabilir (bkz. "Semt" kolonu)
* Her satır bir eşsiz anahtarla tanımlanmalıdır (Unique Key - Primary Key)

Veri tabanımızda ikinci kurala açıkça uymayan bir kolon var: Semt. Bu durumu düzeltmek için tekrar düzenleyelim:

**Ana Tablo**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Calisan** | **Soyad** | **Sofor** | **Arac** | **Semt 1** | **Semt 2** | **Semt 3** |
| Orçun | Yılmaz | Ahmet | Toyota | Levent | Etiler | Ulus |
| Metin | Seyyar | Mehmet | Honda | Bakırköy | Ataköy | Yeşilköy |
| Metin | Seyyar | Tolga | Ford | Kandilli | Beylerbeyi | Kuzguncuk |

Bir sorun var. Tablo bu şekliyle birinci kuralla çelişti. Semt 1, Semt 2, Semt 3 tekrarlayan kolonlar. Bir daha deneyelim:

**Ana Tablo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Calisan** | **Soyad** | **Sofor** | **Arac** | **Semt** |
| Orçun | Yılmaz | Ahmet | Toyota | Levent |
| Orçun | Yılmaz | Ahmet | Toyota | Etiler |
| Orçun | Yılmaz | Ahmet | Toyota | Ulus |
| Metin | Seyyar | Mehmet | Honda | Bakırköy |
| Metin | Seyyar | Mehmet | Honda | Ataköy |
| Metin | Seyyar | Mehmet | Honda | Yeşilköy |
| Metin | Seyyar | Tolga | Ford | Kandilli |
| Metin | Seyyar | Tolga | Ford | Beylerbeyi |
| Metin | Seyyar | Tolga | Ford | Kuzguncuk |

Evet, şimdi oldu. Tekrar eden kolonlar yok. Bir kolonda birden çok veri yok. Süper. Pardon? Eşsiz anahtar mı dediniz? Doğru... E, nasıl yaparız?

Önce  "aday anahtar" (candidate key) ve "eşsiz anahtar" (primary key) kavramlarına bir göz atalım o zaman:

**Aday Anahtar (Candidate Key):** Bir ya da daha fazla kolondan meydana gelir. Tablonun her bir veri satırını eşsiz olarak tanımlar, başka bir deyişle tabloda kaç satır olursa olsun bu kombinasyonu bulunduran birden fazla satır asla olamaz. Örneğin, "Çalışan - Soyad" kombinasyonu bir aday anahtar değildir çünkü 1, 2 ve 3üncü satırlar ve 4, 5, 6, 7, 8 ve 9uncu satırlarda değerler tekrar etmektedir. Öte yandan "Çalışan - Semt" kombinasyonu hiç bir şekilde tekrar etmiyor. Öyleyse "Çalışan - Semt" kombinasyonu bir **aday anahtar**dır.

**Eşsiz Anahtar (Primary Key):** Tablodaki aday anahtarlardan herhangi birini eşsiz anahtar olarak atayabiliriz. Bu anahtar tablodaki satırları tanımlamak için kullanılır ve bir tabloda yalnızca 1 tane eşsiz anahtar bulunabilir.

Tablomuza dönecek olursak, "Çalışan - Semt" kombinasyonunu eşsiz anahtar olarak atayabiliriz. Elimizdeki örnek son derece basit bir tablo olduğundan sorun yok ama daha karmaşık işlerde eşsiz anahtar için kolon kombinasyonlarını pek tercih etmiyoruz. Eh, sonuçta veri tabanlarına **Auto Increment (Otomatik Sayı - Otomatik Artış)**fonksiyonlarını spor olsun diye koymuyorlar, değil mi?

**Ana Tablo**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Calisan** | **Soyad** | **Sofor** | **Arac** | **Semt** |
| 1 | Orçun | Yılmaz | Ahmet | Toyota | Levent |
| 2 | Orçun | Yılmaz | Ahmet | Toyota | Etiler |
| 3 | Orçun | Yılmaz | Ahmet | Toyota | Ulus |
| 4 | Metin | Seyyar | Mehmet | Honda | Bakırköy |
| 5 | Metin | Seyyar | Mehmet | Honda | Ataköy |
| 6 | Metin | Seyyar | Mehmet | Honda | Yeşilköy |
| 7 | Metin | Seyyar | Tolga | Ford | Kandilli |
| 8 | Metin | Seyyar | Tolga | Ford | Beylerbeyi |
| 9 | Metin | Seyyar | Tolga | Ford | Kuzguncuk |

İşte şimdi oldu. "Id" sütununu eşsiz anahtar olarak atadık. 1NF kurallarına harfiyen uyduk. Süperiz...

**1NF Zararları**

**Satır Ekleme**: Yeni bir kullanıcı tanımlandığında ders kodu girilmesi zorunludur.

**Satır Silme**: Kullanıcı silersek tablodan bir bölümde silmiş oluruz, örnek vermek gerekirse eğer 26365 numaralı öğrenciyi silersek ve sadece bir öğrenci kayıtlıysa o bölüme doğal olarak o bölümde silinecektir.

**Satır Güncelleme:**Bir öğrencinin bölümü değiştiğinde birden fazla alanın güncelleştirilmesi gerekecektir

2NF (2. Normal Form)

Bir veri tabanının 2NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

* Tablo 1NF olmalıdır,
* Anahtar olmayan değerler ile kompozit (bileşik) anahtarlar arasında kısmi (partial) bağımlılık durumu oluşmamalıdır. Kısmi bağımlılık durumu, anahtar olmayan herhangi bir değer kompozit bir anahtarın yalnızca bir kısmına bağıl ise oluşur. (Evet farkındayım çok karmaşık görünüyor, örnekte net bir şekilde anlayacaksınız. Söz...)
* Herhangi bir veri alt kümesi birden çok satırda tekrarlanmamalıdır. Bu tür veri alt kümeleri için yeni tablolar oluşturulmalıdır.
* Ana tablo ile yeni tablolar arasında, dış anahtarlar (foreign key) kullanılarak ilişkiler tanımlanmalıdır.

Aslında üç ve dördüncü maddeler ikinci maddenin sonuçlarıdır. Eğer anahtar olmayan bir kolonla herhangi bir komposit anahtar arasında kısmi bağımlılık varsa her zaman tekrarlayan veri alt kümeleri oluşur. Bu durumu düzeltmek için bahis konusu alt kümeleri farklı bir tablo haline getirmeli ve elde ettiğimiz tablolar ile ana tablomuz arasındaki ilişkiyi tanımlamalıyız.

Tablomuzu bir gözden geçirelim: "Çalışan - Soyad" kombinasyonuna bakın. Çok tekrarlanıyor çünkü eşsiz anahtara verimli bir şekilde bağlayamamışız. Bunu düzeltmek için tablomuzu aşağıdaki gibi ikiye bölelim ve aralarında bir ilişki oluşturalım:

**Ana Tablo**

|  |  |  |
| --- | --- | --- |
| **Id** | **Calisan** | **Soyad** |
| 1 | Orçun | Yılmaz |
| 2 | Metin | Seyyar |

**Servis Tablosu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Cid** | **Sofor** | **Arac** | **Semt** |
| 1 | Ahmet | Toyota | Levent |
| 1 | Ahmet | Toyota | Etiler |
| 1 | Ahmet | Toyota | Ulus |
| 2 | Mehmet | Honda | Bakırköy |
| 2 | Mehmet | Honda | Ataköy |
| 2 | Mehmet | Honda | Yeşilköy |
| 2 | Tolga | Ford | Kandilli |
| 2 | Tolga | Ford | Beylerbeyi |
| 2 | Tolga | Ford | Kuzguncuk |

Yeni tablomuz ile ana tablomuzu ilişkilendirmek için "Cid" (Çalışan ID) isimli bir kolon yarattık. Dikkat ederseniz bu kolonun aldığı değer ana tablomuzdaki eşsiz anahtarı işaret ediyor. Bu ilişkilendirmeye **Foreign Key** diyoruz.

Ayrıca bilmem söylemeye gerek var mı, Şoför - Araç - Semt kombinasyonu bu yeni tablomuzun eşsiz anahtarı olarak gayet iyi iş görüyor.

Evet, artık bu noktada 2NF işini hallettik diyebiliriz.

**2NF Zararları**

**Satır Silme**: Bazı durumlarda bir bölümde bir kayıt bulunur. O sütundaki başka bir kayıtta sadece bir veri içeriyorsa silinen bölümle beraber diğer veri de silinecektir. Örneğin Öğrenci tablosunda ki öğrencinin bölümünü ele alalım. Öğrenci silinirse ve sadece bir öğrenci o bölümdeyse öğrenci ile beraber bölümde silinecektir.

3NF (3. Normal Form)

Bir veri tabanının 3NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

* Veri tabanı 2NF olmalıdır,
* Anahtar olmayan hiç bir kolon bir diğerine (anahtar olmayan başka bir kolona) bağıl olmamalı ya da geçişken fonksiyonel bir bağımlılığı (transitional functional dependency) olmamalıdır. Başka bir deyişle her kolon eşsiz anahtara tam bağımlı olmak zorundadır.

Veri tabanımızı 3NF şartlarına uydurabilmek için anahtar olmayan ve eşsiz anahtara tam bağımlı olmayan tüm kolonları kaldırmalıyız. Dikkat ederseniz bizim tablomuzda "Araç" kolonu eşsiz anahtarımıza değil "Şoför" kolonuna bağımlı. Birbirine bağlı olan bu iki kolonu (Şoför - Araç) ayrı bir tabloya ayırmamız ve tablomuzla aralarında bir ilişki yaratmamız gerekiyor.

**Ana Tablo**

|  |  |  |
| --- | --- | --- |
| **Id** | **Calisan** | **Soyad** |
| 1 | Orçun | Yılmaz |
| 2 | Metin | Seyyar |

**Servis Tablosu**

|  |  |  |
| --- | --- | --- |
| **Cid** | **Sid** | **Semt** |
| 1 | 1 | Levent |
| 1 | 1 | Etiler |
| 1 | 1 | Ulus |
| 2 | 2 | Bakırköy |
| 2 | 2 | Ataköy |
| 2 | 2 | Yeşilköy |
| 2 | 3 | Kandilli |
| 2 | 3 | Beylerbeyi |
| 2 | 3 | Kuzguncuk |

**Şoför Tablosu**

|  |  |  |
| --- | --- | --- |
| **Sid** | **Sofor** | **Arac** |
| 1 | Ahmet | Toyota |
| 2 | Mehmet | Honda |
| 3 | Tolga | Ford |

Öncelikle şoför tablosu adında yeni bir tablo oluşturduk. Bu tabloda Sid (Şoför ID) adıyla bir eşsiz anahtar yarattık ve Servis tablomuzdaki Sid kolonundan bu eşsiz anahtara referans vererek foreign key oluşturduk.

3NF'i de gördük ya... Artık karada ölüm yok.

BCNF / 3.5NF (Boyce-Codd Normal Form)

Bir veri tabanının 3.5NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

* Veri Tabanı 3NF olmalıdır,
* Her determinant (belirleyici kolon) aynı zamanda bir aday anahtar olmalıdır.

**Determinant:** Aynı satırdaki diğer kolon değerlerini belirlemek için kullanılan kolon kümesi determinant olarak adlandırılır.

Servis tablomuza dikkatle baktığımızda iki tane determinant olduğunu görebiliriz. **Semt** kolonu, **Cid - Sid** kombinasyonunun; **Sid** ise **Cid** kolonunun determinantıdır.

Bu noktada Semt kolonunun hali hazırda bir aday anahtar olduğunu görebiliyoruz çünkü her bir değer tekrar oluşturmaksızın tüm kayıt satırını tanımlayabilmekte. Öte yandan Sid için aynı şeyi söylemek mümkün değil çünkü tekrarlanıyor.

Elbette bu durumu düzeltmek için tabloyu ikiye ayıracak semt tablosunun değerini tabloları ilişkilendirmek için foreign key olarak kullanacağız.

**Ana Tablo**

|  |  |  |
| --- | --- | --- |
| **Id** | **Calisan** | **Soyad** |
| 1 | Orçun | Yılmaz |
| 2 | Metin | Seyyar |

**Servis Tablosu**

|  |  |
| --- | --- |
| **Cid** | **Semt** |
| 1 | Levent |
| 1 | Etiler |
| 1 | Ulus |
| 2 | Bakırköy |
| 2 | Ataköy |
| 2 | Yeşilköy |
| 2 | Kandilli |
| 2 | Beylerbeyi |
| 2 | Kuzguncuk |

**Semt Tablosu**

|  |  |
| --- | --- |
| **Semt** | **Sid** |
| Levent | 1 |
| Etiler | 1 |
| Ulus | 1 |
| Bakırköy | 2 |
| Ataköy | 2 |
| Yeşilköy | 2 |
| Kandilli | 3 |
| Beylerbeyi | 3 |
| Kuzguncuk | 3 |

**Şoför Tablosu**

|  |  |  |
| --- | --- | --- |
| **Sid** | **Sofor** | **Arac** |
| 1 | Ahmet | Toyota |
| 2 | Mehmet | Honda |
| 3 | Tolga | Ford |

Görülebileceği gibi artık tablolarımızın hiçbirinde aday anahtar olmayan determinant yok. Bu nedenle veri tabanımız BCNF'tir diyebiliriz.

**Not:** Hazır Boyce - Codd demişken: [Raymond F. Boyce](http://en.wikipedia.org/wiki/Raymond_F._Boyce" \t "_blank), [Edgar F. Codd](http://en.wikipedia.org/wiki/Edgar_F._Codd).

4NF (4. Normal Form)

Bir veri tabanının 4NF olabilmesi için aşağıdaki özellikleri karşılayabilmesi gerekir:

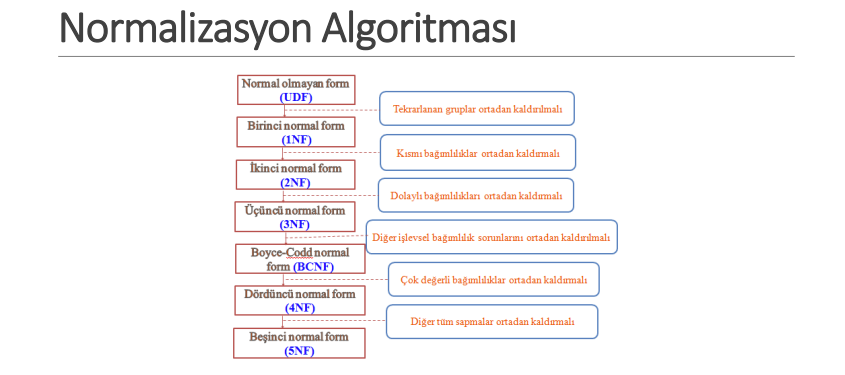
* Veri Tabanı 3NF olmalıdır,
* Çok-değerli bağımlılıkları (Multli-Valued dependency) olmamalıdır.

**Multi-Valued Dependency:** Bu durum bir ya da daha çok veri satırının var olması, aynı tabloda başka bir (ya da daha çok) veri satırının bulunmasını gerektirdiğinde ortaya çıkar. Örneğin, bir yazılım firması düşünelim. Geliştirdikleri yazılımların masaüstü bilgisayarlar için olanlarını tek-kullanıcılı ve çok-kullanıcılı olarak iki versiyonla piyasaya sunuyor olsunlar. Diyelim ki bu firmanın geliştirdiği tüm yazılımları barındıran bir veri tabanı oluşturuyoruz. Bu veri tabanında bir masaüstü yazılımın tek-kullanıcılı versiyonunu eklediysek mutlaka bir başka satırda aynı yazılımın çok-kullanıcılı versiyonu için de bir kayıt açılmış olmak durumundadır...

Hali hazır örneğimizde böyle bir durum oluşmadığından 4NF uyarlaması yapılmasına gerek (ve imkan) yoktur.

Sonuç

**En azından ilk üç seviye normalizasyonu her zaman ve mutlaka yapmak gerektiğine inanıyorum**. Ancak bazı özel durumlarda (Örneğin nadiren kayıt girişi yapılan fakat sürekli yeni yeni sorgular yazılan bir veri tabanı düşünün) sorgularınızı kodlamanın biraz daha kolaylaşması için bazı kolonların birden çok tabloda tekrarlanmasını isteyebilirsiniz. Bu türden durumlarda normalizasyonu bir seviyeden sonra yapmamayı tercih edebilirsiniz. O seviyenin 3NF'den aşağı olmamasını şiddetle tavsiye ederim. Yine de hiç üşengeçlik etmesek daha iyi tabii..



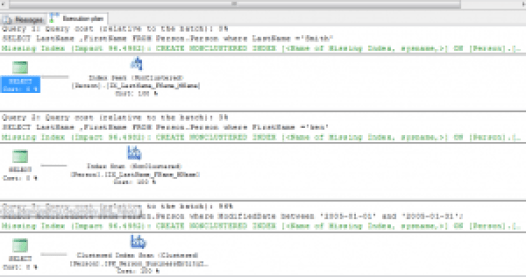
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Performans karşılaştırmasını nasıl görebiliriz?**

Performans karşılaştırması yapmak için **Display Estimated Execution Plan** butonuna tıkladıgımızda, cost’tan nerede ne kadar zaman harcadıgını görebiliriz.

Örneğin,

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **SELECT LastName,FirstName FROM Person.Person**  **where LastName =‘Smith’**  **SELECTLastName,FirstName FROM Person.Person**  **where FirstName =‘ken’**  **SELECT ModifiedDate FROM Person.Person**  **where ModifiedDate between ‘2005-01-01’ and ‘2005-01-31’;** |

**[](https://i0.wp.com/koddit.com/wp-content/uploads/2014/08/6.png)**

**Merge komutunun kullanımı**

Update, insert gibi komutları bir arada conditiona bağlı olarak yapıldığı, eşleşen kayıt varsa update et,  kayıt yoksa insert et işlemlerini yapmak için kullanırız.

Olan kaydı update edersin, source içinde yoksa insert edersin.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **MERGE INTOtablename**  **USINGtable\_reference**  **ON(condition)**  **WHEN MATCHED THEN UPDATE**  **SETcolumn1=value1[,column2=value2…]**  **WHEN NOTMATCHED**  **THEN INSERT (column1 [, column2 …]) VALUES (value1 [, value2 …** |

Örneğin,

|  |  |
| --- | --- |
| 1  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **MERGE#producttgt**  **USINGProductWellDonesrc**  **ONsrc.ProductID=tgt.ProductID**  **WHENMATCHED**  **THEN**  **UPDATE**  **SET tgt.HasAferin = ‘Yes’**  **WHEN NOT MATCHED BYSOURCE**  **THEN**  **DELETE ;** |

**Hadi bir de örneklere göz atalım!**

**1. Cursor kullanımı:**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32** | **USE dukkan**  **GO**  **— oncelikle scroll cursor’umuzun sececigi ifadeyi görelim:**  **SELECT markaKod, marka FROM tblMarka**  **ORDER BY markaKod**  **— Scroll cursor tanımlayalım.**  **DECLARE crScrMarka SCROLL CURSOR FOR**  **SELECT markaKod, marka FROM tblMarka**  **ORDER BY markaKod**  **OPEN crScrMarka**  **— cusror’un son satirina gidelim:**  **FETCH LAST FROM crScrMarka**  **— anlik olarak ilk kayda gecelim:**  **FETCH PRIOR FROM crScrMarka**  **— bastan 2.kayda konumlanalim:**  **FETCH ABSOLUTE 2 FROM crScrMarka**  **— bulundugumuz yerden 3 kayit ileri konumlanalim:**  **FETCH RELATIVE 3 FROM crScrMarka**  **—  bulundugumuz kayittan 2 kayit geriye konumlanalim:**  **FETCH RELATIVE -2 FROM crScrMarka**  **CLOSE crScrMarka**  **DEALLOCATE crScrMarka**  **GO** |

**2. Tablo tipi parametre kullanım örneği:**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29** | **— 1. Daha onceden tanimlanmamissa once tablo tipini tanımlayalim.**  **CREATE TYPE dbo.MarkaTip AS TABLE(**  **markaKod INT,**  **marka VARCHAR(50)**  **);**  **GO**  **—–2.. Tablo alan Stored Procedure kodlanmasi**  **ALTER PROC \_SP\_BelliMarkaUrunlerSec(@markaTablo AS dbo.MarkaTip READONLY )**  **AS**  **BEGIN**  **SELECTU.urunKod,U.urunAd,M.marka**  **FROM tblUrun U JOIN @markaTablo M ON U.markaKod=M.MarkaKod**  **END**  **GO**  **— 3.simdi de kullanalim….**  **–Once girdi parametreyi olusturup dolduralim**  **DECLARE @marka AS dbo.MarkaTip;**  **INSERT INTO @marka(markaKod,marka)**  **SELECTTOP(9)markaKod,Marka**  **FROM tblMarka;**  **–sonra stored Procedure’u secelim**  **EXEC \_SP\_BelliMarkaUrunlerSec @marka**  **GO** |

**3. Transaction örneği:**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31** | **CREATE TABLEtblHesap(**  **HesapNoCHAR(10) PRIMARY KEY NOT NULL,**  **isimVARCHAR(55),**  **soyadVARCHAR(55),**  **subeINTEGER,**  **bakiyeMONEY**  **) ;**  **INSERT tblHesap**  **VALUES(‘000000023’,‘Ali’,‘Eryatan’,749,30000000);**  **INSERT tblHesap**  **VALUES(‘000000042’,‘Ahmet’,‘Eryatan’,749,30000000);**  **— Ali Ahmet’e 1000 lira havale yaparsa…**  **BEGIN TRAN**  **UPDATE tblHesap**  **SET bakiye=bakiye – 1000**  **WHERE hesapNo=‘000000023’**  **UPDATE tblHesap**  **SET bakiye=bakiye + 1000**  **WHERE hesapNo=‘000000042’**  **COMMIT**  **— TRY-CATCH ile kullanilmali.** |

**4. Veritabanı yetkinliklerini artırma – yedekleme ve yedekten döndürmek:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | USE master;  ALTER DATABASE dukkan SET RECOVERY FULL ;  BACKUP DATABASE dukkan  TO DISK=‘D:\yedekler\dukkan.bak’;  ha  BACKUP DATABASE dukkan  TO DISK=‘D:\yedekler\dukkan.bak’  WITH INIT;  SELECT b.backup\_set\_id yedekno, b.media\_set\_id medya\_no, b.database\_name veritabani, f.logical\_name ad, f.file\_type yedekTip  FROM backupset b LEFT JOIN backupfile f ON b.backup\_set\_id = f.backup\_set\_id  RESTORE DATABASE [dukkan3] FROM  DISK = N‘D:\Yedek\dukkan.bak’  WITH  MOVE ‘dukkan\_Data’ TO ‘D:\SQLData\2008\_dukkan\dukkan3.MDF’,  MOVE ‘dukkan\_Log’ TO ‘D:\SQLData\2008\_dukkan\dukkan3\_1.LDF’  USE msdb ;  SELECT backup\_set\_id, media\_set\_id, position, name, type  FROMbackupset  WHERE name LIKE ‘dukkan-%’ ;  RESTORE DATABASE [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 1, NORECOVERY ;  RESTORE LOG [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 2,NORECOVERY ;  USE msdb ;  SELECT backup\_set\_id, media\_set\_id, position, name, type,backup\_start\_date,backup\_finish\_date  FROMbackupset  WHERE name LIKE ‘dukkan-%’  — Belli bir zamana donmek  RESTORE DATABASE [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 1, STOPAT=‘2010-06-28 00:00’, NORECOVERY ;  RESTORE LOG [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 2, STOPAT=‘2010-06-28 00:00’, NORECOVERY ;  RESTORE LOG [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 3, STOPAT=‘2010-06-28 00:00’, NORECOVERY ;  RESTORE LOG [dukkan]  FROMDISK= ‘D:\Yedek\dukkan.bak’  WITH FILE = 4, STOPAT=‘2010-06-28 00:00’, RECOVERY ; |