**Transact SQL ile Veri Tabanı Oluşturmak**

Bir veri tabanı Management Studio ile oluşturulabildiği gibi diğer yöntem olan T-SQL

ile de oluşturulabilir. Ayrıca, Management Studio ile yapılan tüm işlemler T-SQL ile de

yapılabilir.

T-SQL ile bir veri tabanının oluşturulması en basit hâliyle şu şekildedir.

*CREATE DATABASE veri tabanı\_ismi*

Ancak, bu şekilde bir veri tabanı oluşturmak, model veri tabanının kopyalanması ve

yeni kopyanın adının belirtilen veri tabanı ismiyle değiştirilmesidir. Bir veri tabanı dosyasını

çeşitli özellikleriyle oluşturacak olursak genel kullanımı şöyle olacaktır.

*CREATE DATABASE veri tabanı\_ismi*

*ON*

*PRIMARY ( NAME=dosya\_takma\_ismi*

*, FILENAME=fiziki\_dosya\_ismi*

*, SIZE=dosya\_boyutu*

*, MAXSIZE=maksimum\_dosya\_boyutu*

*, FILEGROWTH=dosya\_artım\_miktarı )*

*LOG ON*

*( NAME=dosya\_takma\_ismi*

*, FILENAME=fiziki\_dosya\_ismi*

*, SIZE=dosya\_boyutu*

*, MAXSIZE=maksimum\_dosya\_boyutu*

*, FILEGROWTH=dosya\_artım\_miktarı )*

Birincil (Primary) dosya, veri tabanı için gerekli olan tüm nesneleri saklayan dosyadır.

Bir veri tabanı oluşturulurken birincil dosya belirtilmemişse ilk oluşturulan dosya birincil

dosya olarak belirlenir. Veri tabanında mutlaka birincil dosya bulunmalıdır.

Oluşturulan veri tabanında kullanılan ifadelerin anlamları şöyledir.

 **Veri tabanı\_ismi:** Veri tabanına verilecek isimdir.

 **Dosya\_takma\_ismi:** Veri tabanındaki dosyalar için belirlenen takma isimdir.

İlgili dosyaya erişimde pratiklik sağlar. İşletim sistemince bilinen isimdir.

 **Fiziki\_dosya\_ismi:** İşletim sisteminde saklanacak dosyanın adını ve yolunu

belirtir. Bu isimde işletim sistemince bilinen bir isimdir.

 **Dosya\_boyutu:** Veri tabanı tanımlandığındaki boyutunu belirtir. Başlangıçta

1MB’tır. Management Studio’da bu 3MB olarak tanımlanır. Oluşturulacak log

dosyası da bunun %10’u kadardır.

 **Maksimum\_dosya\_boyutu:** Dosyanın maksimum boyutunu belirtir.

Belirtilmezse disk dolana kadar dosya artım miktarı kadar artmaya devam eder.

 **Dosya\_artım\_miktarı:** Dosyanın belirtilen boyutu dolduğunda dosya boyutu

otomatik olarak artar. Artım miktarı bu parametreyle belirtilir.

**1.6. Veri Türleri**

Veri türleri, numara, resim, tarih gibi bir sütunda veya bir değişkende depolanabilen türdeki bilgiyi belirtir. SQL Server 28 "sistem tanımlı" veri türünü tanır. Bu veri türleri haricinde, belirli ihtiyaçları karşılayabilmeniz için kullanıcı tanımlı veri türlerini T-SQL ve .NET ile oluşturabilirsiniz.

 Sistem tanımlı veri türleri kategorileri şunlardır.

 Karakter dizilimi (Character strings)

 Unicode karakter dizilimi (Unicode character strings)

 Tarih ve saat (Date and time)

 Tahmini sayısal (Approximate numeric)

 Kesin sayısal (Exact numeric)

 Tam sayılar (Integer numbers)

 Parasal (Monetary)

 İkili (Binary)

 Özel (Special)

Bazı durumlarda, bir veri türünü T-SQL kodda ilişkilendirmek için farklı tanımlayıcılar/tanıtıcılar kullanabilirsiniz. Örneğin; Char veri türü Karakter olarak, varchar ise karakter çeşitliliği olarak ele alınabilir. Bu eş anlamlı sözcüklerin bazıları ANSI SQL-92 standart gereksinimlerine dayanmaktadır.

**1.6.1. Karakter Dizilimi**

Karakter veri türleri karakter dizilimlerini içerir. Dört farklı karakter türü, uzunlukları ve depolama özellikleri açısından çeşitlilik gösterir.

 char

 varchar

 varchar(max)

 text

char veri türü, sabit boyuttaki dizilimleri depolamak için kullanılır. Daha önce belirtildiği gibi, bu veri türünün maksimum uzunluğu/boyutu 8000 karakterdir ki bu, eski versiyonlardaki 255 karakter limiti bağlamında kayda değer bir artıştır.

**Örnek**

TCKimlikNo char(11)

varchar veri türü, 8000 karakter uzunluğa kadar olan çeşitli boyutlarda dizilimleri depolar. Uzunluğu nominal boyuttan az olan karakter değeri bir sütuna veya bir değişkene atandığında SQL Server peşinden boşluklar eklemek yerine onu olduğu gibi kaydeder.

Varchar veri türleri dizilimin uzunluğunu kaydetmek için 2 ilave byte kaplar.

**Örnek**

ad varchar(50)

varchar (max), varchar veri türü gibi görünür ve işler ancak dâhili yapı ve işlevselliği daha çok metin veri türü gibidir. Aynı zamanda 2 GB’a kadar olan geniş dizilimleri depolamak için tasarlanmıştır. Dizilim, kaydın geri kalanıyla veri sayfasında değil özel veri sayfalarında depolanır. Bunun en büyük avantajı varchar ile çalışan çoğu dizilim operasyon fonksiyonunun aynı zamanda varchar (max) ile de çalışmasıdır; bununla birlikte ilave işlevsellik için birkaç yeni fonksiyon bulunmaktadır.

**Örnek**

Yeni\_deger varchar(max))

text veri türü, büyük miktarlardaki veriyi depolamak için kullanılır. Tek bir alan 2GB

(231 - 1 byte) bilgiye kadar depolayabilir. Sadece 16 byte tabloda depolanır. Bu nedenle ilave işlem, metin sütunlarının kullanımıyla ilişkilendirilir. Metin değerleri işlemi için bazı özel fonksiyonlar bulunmaktadır.

**Örnek**

durum text

**1.6.2. Unicode Karakter Dizilimi**

Evrensel Kod standardının kullanılması ve onu destekleyen yazılımların bulunması son zamanlardaki küresel yazılım teknolojilerinin en popüler konusudur.

Evrensel Kod’un, istemci-sunucu ya da çok katmanlı uygulamalar ve web sitelerinde kullanımı önemli bir düzeyde tasarruf sağlamaktadır. Evrensel Kod tek bir yazılımın ya da bir web sitesinin yeniden tasarlanmasına ihtiyaç duyulmadan her türlü dil ve ülke için uyumlu hâle getirilmesini sağlar. Aynı zamanda verinin çeşitli işletim sistemleri içerisinden aktarımını da sorunsuz hâle getirmektedir.

Microsoft SQL Server 2005 non-ASCII (ASCII karakteri olmayan) karakter setleri kullanan Unicode veriyi depolamak için dört karakter veri türüne sahiptir. Bunlar char, varchar, varchar (max) ve metin veri türlerine eşdeğerdir ve aşağıdaki gibi adlandırılır:

 nchar

 nvarchar

 nvarchar(max)

 ntext

Bu yeni veri türleri ile eski veri türleri arasındaki fark; yeni veri türlerinin karakter başına 2 byte kaplayan Unicode karakterlerini tutabilmeleridir. Bu nedenle, depolayabildikleri maksimum dizilim uzunluğu, eski veri türlerindekinin yarısıdır (4000).

**Örnek**

TCKimlikNo nchar(11)

**Örnek**

ad nvarchar(50)

**Örnek**

Yeni\_deger nvarchar(max))

**Örnek**

durum ntext

**1.6.3. Tarih ve Saat Veri Türleri**

SQL Server tarih ve saati depolamak için iki T-SQL veri türünü destekler.

 Datetime

 Smalldatetime

Bu iki veri türü arasındaki fark kapladıkları alandır. datetime 8 byte, smalldatetime sadece 4 byte kullanır.

Aralarındaki diğer farklar ise depolanan tarihin doğruluğu/kesinliği ve kullanılabilecek tarihlerin genişliğidir. Smalldatetime’ın doğruluğu bir dakikadır ve 1 Ocak 1900’den 6 Haziran 2079’a kadar olan zamanı kapsar ki bu da yeterinden fazladır. Datetime’ın doğruluğu 3.33 milisaniyedir ve 1 Ocak 1753’ten 31 Aralık 9999’a kadar olan süreyi kapsar.

**1.6.4. Tahmini Sayısal Veri Türü**

Ondalık sayılar genellikle tek veya çift kesinlik (single and double precision) olarak da

bilinen real ve float veri türlerinde depolanır. Fazla yer kaplamamaları ve geniş aralıkta

sayıları tutabilmeleri avantajdır. Tek sorun şudur ki; kati/kesin değillerdir. Yaklaşık olarak

orijinal ondalık sayıya eş değer yaklaşık/tahmini ama kesin olamayan sayının çift değişkenli

temsilini depolar.

Kesinlik/doğruluk (Precision) numaradaki önemli/karakteristik basamak/hanelerin

sayısı ve Scale ondalık noktanın sağa doğru uzanan basamak/hanelerin sayısıdır.

Örneğin; 123456.789 sayısının 9 doğruluğu ve 3 ondalığı vardır. Gerçek sayıların

basamakları 7’ye kadardır ve float sayıların kesinliği 15 basamağa kadardır. Bu nedenle,

Dünya ve Ay arasındaki mesafeyi ölçerken birkaç metrenin önemsenmeyeceği bilim ve

mühendislik için idealdirler. Fakat şirket bütçesinin son kuruşuna kadar kesin olması gereken

finansal endüstri için yeterli değillerdir.

**1.6.5. Kesin Sayısal Veri Türü**

Ondalık veya sayısal veri türleri sayıları depolarken yaklaşıklık/tahmini kullanmazlar.

Ne yazık ki gerçek ve float veri türlerine göre daha fazla yer/alan gerektirirler. Ondalık bir

sütun veya değişken tanımlandığında/belirtildiğinde basamağını ve doğruluğunu (scale and

precision) belirtmelisiniz. SQL Server maximum 38 doğruluktaki sayıları depolayabilir.

Basamak (scale) doğruluktan daha az ya da onunla eşit olabilir.

**1.6.6. Tam Sayı Veri Türü**

SQL Server 1-, 2-, 4-, ve 8-byte tam sayıları destekler. Bit veri türü mantıksal doğru

ve yanlış değerleri temsil eden 1 ve 0 depolamak için kullanılır. Aşağıdaki tablo tam sayı

veri türlerini, depolama boyutlarını ve değerlerin aralığını listeler.

**Veri Türü Boyutu Minimum Maximum**

int 4 bytes -2, 147, 483, 648 2, 147, 483, 647

smallint 2 bytes -32768 32767

tinyint 1 bytes 0 255

bigint 8 bytes -9,223,372,036,854,775,808 9,223,372,036,854,775,807

bit 1 bit 0 1

int veri türleriyle ilgili en güzel şey küçük alanda oldukça fazla sayı depolayabilmesidir. Bu nedenle sık sık anahtar değerler (key values) için kullanılır. Ana anahtarın veri türü int ise tablo dört milyara kadar kayıt depolayabilir ki miktar herhangi bir amaç için oldukça yeterlidir. Bu nedenle tablonuzun boyutundan emin değilseniz ana anahtar olarak int kullanınız.

**1.6.7. Parasal Veri Türü**

Tam sayıya ondalık ifadesi eklenmiş veri türleridir. Duyarlılık ve ölçekleme değiştirilmez. SQL Server, verileri bir tam sayı gibi ele alır. Parasal sabitlerden önce $ veya diğer 18 para sembolünden biri gelebilir (SQL Server Books OnLine’da listelenmişlerdir).

**Veri Türü Boyutu Sınırı**

smallmoney 4 bytes -214 bin ile +214 bin arasında değerler için

money 8 bytes -922 milyar ile +922 milyar arasında değerler için kullanılır.

**1.6.8. Binary (İkili) Veri Türü**

Binary veri tipleri bitlerin diziliminin yerleştirilmesi için kullanılır. SQL Server dört temel çift değişkenli veri tipini, karakter veri türü ile benzer nitelikleri desteklemektedir.

 binary

 varbinary

 varbinary(max)

 image

binary and varbinary veri türleri 8000 byte bilgi ve resim, varbinary (max) 2 GB veri depolayabilir.

**1.6.9. Özel Veri Türleri**

Belirtilen veri türleri dışında bir gruba dâhil olmayan veri türleri vardır.

**Veri Türü Açıklama**

Cursor Tabloların yapısında yer almayan bir tür olan Cursor, kayıtlara bir sonuç kümesi (resultset) yerine, erişim için kullanılan bir değişken türüdür.

Table table veri türü sonraki süreçlerde recordset yerleştirmek için kullanılır.

Bazı açılardan bu veri türü geçici tablo (temporary table) ile benzerdir.

Bu veri türünü sütun tanımlamak için kullanamazsınız. Sadece fonksiyonun değerini geri döndürmek için local değişken olarak kullanılabilir.

Timestamp Satır sürüm numarası olarak tanımlı bir türdür. Ortak zamanlı çalışma için, düzenleyici unsur olarak kullanılır.

Uniqueidentifier Her çalışmada benzeri olmayan bir 16 byte’lık 16’lı tabanda sayı üreten sayı türüdür. Çok geniş bir ölçümde tekil değerler elde etmek için kullanılır.

**2. PLANLAMA**

**2.1. Veri Tabanı Normalizasyonu**

**2.1. Veri Tabanı Normalizasyonu**

Bir tablo bir tek varlığa ait bilgileri tutmak için tasarlanır. Örneğin bir kitap tablosunda

sadece kitaplara ait bilgiler vardır. 20 adet kitap bilgisi varsa, bu tabloya 20 adet satır

eklenecek demektir. İlişkisel veri tabanı yaklaşımını ilişkisel yapan asıl unsur verilerin

tablolara parçalanarak saklanmasıdır.

Tabloların kaç adet olacağı ve birbiriyle nasıl ilişkilendirileceğine karar verirken var

olan kuralları kullanarak mantıksal bir plan elde etmelisiniz.

Eğer tablolar üzerinde verileri normalize etmezseniz birçok sıkıntı oluşabilir. Bazı

bilgiler birden fazla tekrarlanabilir. Bir bilgiyi güncellemek, veri eklemek ve silmek için

birden fazla yerde bulma, değiştirme ve silme yapmanız gerekebilir. Bu nedenle

normalizasyon kurallarını kullanmalısınız.

Genel kabul gören beş normalizasyon kuralı vardır. Bu kurallar, ilişkisel veri tabanının

tanımı ile birlikte ortaya konmuştur. Veri tabanı normalizasyonu Resim 2.1'e bağlı olarak

anlatılacaktır. Açıklamalar yapılırken bu resme dikkat etmelisiniz.



 **1. Normalizasyon Kuralı**

Bir satırdaki bir alan yalnızca bir tek bilgi içerebilir.

Örneğin kitap tablosunda, birden fazla yazarı olan kitap için yazar1, yazar2, yazar3

diye alanlar açsanız, bu kurala uymamış olursunuz. Böyle bir durumda, ayrıca yazarlar

tablosu da oluşturarak kuralı çiğnememeye dikkat etmelisiniz.

 **2. Normalizasyon Kuralı**

Bir tablo için, anahtar olmayan her alan, birincil anahtar olarak tanımlı tüm alanlara

bağlı olmak zorundadır.

Mesela, Ödünç tablosuna KitapAdi diye bir alan ekleseydik, bu sadece ödünç verilen

kitap ile ilgili bir bilgi olacaktı ve oduncNo'na bağlı bir nitelik olmayacaktı. Bunu çözmek

için, kitap adlarını ayrı bir tabloda tutarak sorun çözülür.

Ya da anahtar alanın birden fazla alandan oluştuğu tablolarda, anahtar alanlardan

sadece birine bağlı veriler, tabloda yer almamalı, ayrı bir tabloya taşınmalıdır.

 **3. Normalizasyon Kuralı**

Bir tablo için, anahtarı olmayan bir alan, anahtarı olmayan başka hiç bir alana bağlı

olamaz.

Örneğin, kitaplarımız için cilt tipi adında bir alan ekleyip burada da karton kapak için

K, deri cilt için D, spiral cilt için S yazsaydık, bu kodlama, kitap tablosunun birincil anahtarı

olan kitapNo alanına bağlı bir kodlama olmayacaktı. Çünkü bu kodlama bir başka anahtarı

olmayan alana bağlıdır. Bunun sonucunda da veri tabanımızda, karşılığı olmayan bir

kodlama yer almış olacaktır. Cilt tipi bilgisini kodlu olarak tutan alan aslında cilt tipi

açıklaması olan başka bir alana bağlıdır. Bu ilişki başka bir tabloda tutulmalıdır.

Bu durumda, ciltŞekli adında bir tablo açmamız gerekir. Bu tablonun alanları da

ciltTipKodu ve ciltSekli olmalıdır. Ancak bundan sonra, kitaplar tablosunda ciltTipi adında

bir sütun açıp buraya da D,S,K gibi kodları yazabilirsiniz..

 **4. Normalizasyon Kuralı**

Birincil anahtar alanlar ile anahtarı olmayan alanlar arasında, birden fazla bağımsız

bire-çok ilişkisine izin verilmez.

Örneğin, tablomuzda yer alan bir kitap hem hikaye kitabı hem de kişisel gelişim kitabı

olabilir.

4.normalizasyon kuralını sağlamak için, her bağımsız bire-çok ilişki için ayrı bir tablo

oluşturmak gerekir.

Bu örnekte, türler diye bir tablo açmamız ve daha sonra da kitapTurleri diye bir başka

tablo daha açmamız gerekiyor. “Kişisel Gelişim Hikâyeleri” adlı kitap için, öncelikle kitap

numarası, Hikâye bölümünün kodunun yer aldığı bir satır; ardından da yine kitap numarası,

ardından da kişisel gelişim türünün kodunun aldığı yeni bir satır daha eklemek gerekir.

 **5. Normalizasyon Kuralı**

Tekrarlamaları ortadan kaldırmak için her bir tabloyu mümkün olduğunca küçük

parçalara bölmek gerekir. Aslında ilk 4 kural sonuçta bu işe yarar ancak, bu kurallar

kapsamında olmayan tekrarlamalar da 5.normalizasyon kuralı ile giderilir.

Örneğin, kitaplarımız için bir edinme şekli bilgisi girilecek sütun eklemek isteyelim:

Bu bölüme girilebilecek bilgiler bellidir: Bağış veya satın alma gibi.

Bu bilgileri başka bir tabloda tutabiliriz. Böylece, kullanıcıların bu alana gelişigüzel

bilgiler girmesini engellemiş olursunuz. Bu da sorgulama esnasında verileriniz arasında bir

tutarlılık sağlar. Bu işlem sonucunda, tutarsızlıklara neden olabilecek ve sık tekrarlayan veriler başka bir tabloya taşınmış olur. Bu tablo için, veri tabanı programlamada “look-up

table” terimi kullanılır.

Ancak, veri tabanı normalizasyon kuralları, bir ilişkisel veri tabanının tasarlanma

aşamalarını değil de ilişkisel veri tabanında yer alacak kayıtların ilişkisel veri tabanı ile

uyumlu olup olmadığını denetlemeye yöneliktir.

Özetle ilişkisel bir veri tabanı tasarımı şu dört ögeyi barındırmalıdır.

 Veri tekrarı yapılmamalıdır.

 Boş yer mümkün olduğunca az olmalıdır.

 Veri bütünlüğü sağlanmalıdır.

 Veriler, aralarında bir ilişki tanımlamaya müsait olmalıdır.

**1. T-SQL**

**1.1. Transact SQL Kavramı**

Microsoft'un veri tabanı sorgulama dilidir. Transact-SQL, SQL Server ve istemci (client) arasında iletişimi sağlayan SQL sorgulama dilinin gelişmiş bir versiyonudur. Transact Structured Query Language kelimelerinin kısaltmasıdır.

T-SQL kullanarak veri tabanına kayıt eklenebilir, silinebilir, güncellenebilir ya da sorgulama ve raporlama yapılabilir.

T-SQL ile döngü veya mantıksal işlemler yapmak için bir derleyiciye gerek yoktur. Herhangi bir programlama dili öğrenmeden de T-SQL ile tüm amaçlarınıza hitap edecek projeler gerçekleştirebilirsiniz.

T-SQL ifadelerini çalıştırabilmek için bir Management Studio ile SQL Server’a erişmeniz gerekir.

SQL deyimleri veri tabanları üzerinde çeşitli işlemleri yerine getirir. Veri tabanından sorgulama yapmak için SELECT, ekleme yapmak için INSERT güncelleme yapmak için UPDATE, silme yapmak için DELETE, yeni tablo oluşturmak için CREATE TABLE gibi komutlara sahiptir.

Bu komutlar, işlevlerine göre şu şekilde ayrılır:

 DDL (Data Definition Language): Veri tanımlama dili

 DML (Data Manipulation Language) : Veri işleme dili

 DCL (Data Control Language): Veri kontrol dili

**1.2. Veri Tanımlama Dili (DDL)**

SQL Server içinde veri tabanı, tablo ve kullanıcı tanımlı veri tipleri gibi nesneler oluşturmak ve bunları yapılandırmak için kullanılır. Temel komutları aşağıdaki şekildedir:

**Temel Komutlar Açıklama**

CREATE Nesne oluşturmak için kullanılır.

ALTER Nesneler üzerinde değişiklik yapmak için kullanılır.

DROP Nesneleri silmek için kullanılır.

**1.2.1. CREATE (Nesne Oluşturmak)**

Veri tabanındaki nesnelerin oluşturulabilmesi için CREATE komutu kullanılır. Oluşturulacak nesnenin özellikleri dikkate alınarak farklı parametreler kullanılmalıdır.

 **Genel Yazımı**

CREATE nesne\_adi

**Örnek:**

CREATE DATABASE Person

Person adında bir veri tabanı oluşturulur.

**Örnek:**

CREATE TABLE PERSONEL(

PERSONEL\_ID int,

AD varchar(10),

SOYAD varchar(10)

)

Bu şekilde bir yazımla PERSONEL adında bir tablo oluşturulur. Tablo sütunları da PERSONEL\_ID, AD, SOYAD’dır.

**Örnek:**

CREATE TABLE PERSONELYAKIN(

PERSONEL\_ID int,

YAKIN\_ID int,

YAKIN\_AD varchar(10),

YAKIN\_SOYAD varchar(10)

)

Bu örnekte de PERSONELYAKIN adında bir tablo oluşturulmuştur. Tablo sütunları da PERSONEL\_ID, YAKIN\_ID, YAKIN\_AD, YAKIN\_SOYAD’ dır.

**1.2.2. ALTER (Nesnelerde Değişiklik Yapmak)**

Daha önceden oluşturulmuş bir nesne özelliğinin değiştirilmesini sağlar.

 **Genel Yazımı**

ALTER nesne **nesne\_adi** değişim\_cümlesi

**Örnek:**

ALTER TABLE PERSONEL

ADD BABA\_AD varchar(20) NOT NULL

Bu şekildeki bir yazımla PERSONEL tablosuna BABA\_AD sütunu eklenmiştir. NOT NULL ile de bu sütuna veri girişi zorunlu hâle getirilmiştir.

**Örnek**

ALTER TABLE PERSONEL

ALTER COLUMN AD varchar(15) NOT NULL

Bu yazım ile de varolan AD sütununun alabileceği karakter sayısı 15 olarak değiştirilmiş ve veri girişi zorunlu hâle getirilmiştir.

**1.2.3. DROP (Nesne Silmek)**

Bir nesnenin silinmesini sağlayan komuttur. DROP komutu tüm nesneler için kullanılır.

 **Genel Yazımı**

DROP nesne nesne\_adi

**Örnek:**

DROP TABLE PERSONEL

Bu şekilde bir yazımla PERSONEL tablosu silinmiş olur.

**1.3. Veri İşleme Dili (DML)**

Veri tabanı içindeki veriler ile ilgili işlemler yapılmasını sağlar. Temel komutları aşağıdaki şekildedir.

**Temel Komutlar Açıklama**

SELECT Veri tabanındaki verileri seçmeyi sağlar.

INSERT Veri tabanına yeni veriler eklemek için kullanılır.

UPDATE Veriler üzerinde değişiklik (güncelleme) yapmak için kullanılır.

DELETE Veri tabanından veri silmek için kullanılır.

**1.3.1. SELECT Komutu**

Verilere erişmek için en sık kullanılan komuttur. Bir tablodaki bir veya daha çok alan için SELECT komutu yazılabilir.

 **Genel Yazımı**

SELECT sütun\_adi1, [sütun\_adi2],…………. [\*]

FROM tablo\_adi

**Örnek:**

SELECT \* FROM PERSONEL

Bu yazımla PERSONEL tablosundaki tüm alanlar seçilmiş olur.

**Örnek:**

SELECT PERSONEL\_ID, AD FROM PERSONEL

Bu yazım ile de PERSONEL tablosundaki PERSONEL\_ID ve AD alanları seçilmiş olmaktadır.

**Örnek:**

SELECT AD+ ‘ ‘ + SOYAD

FROM PERSONEL

PERSONEL tablosunda yer alan AD ve SOYAD alanlarını tek bir sütun gibi birleştirerek göstermeyi sağlayan SELECT ifadesidir.

**Örnek:**

SELECT ad FROM rehber

ifadesiyle rehber tablosundaki sadece “ad” alanı bilgilerinin elde edilmesini sağlar.

**Örnek:**

SELECT \* FROM rehber WHERE ad=’Ali’

ifadesiyle ad alanındaki Ali ismindeki tüm kayıtların elde edilmesini sağlar.

**Örnek:**

SELECT \* FROM rehber WHERE ad=’Tuncay’ ORDER BY ad ASC

Verilen koşullara göre sütundaki bilgileri artan (ASC) ya da azalan (DESC) sırada ekrana getirir. WHERE ile oluşturulan koşul ifadelerinde mantıksal operatörler de kullanılabilir (and, or,not).

**Örnek:**

SELECT \* FROM rehber ORDER BY ad, soyad

ifadesiyle ad alanına göre kayıtları, adı aynı olanları da soyad alanına göre seçme işlemini gerçekleştirir.

**1.3.2. INSERT Komutu**

Veri tabanına yeni bir kayıt eklemek için kullanılır.

 **Genel Yazımı**

INSERT INTO tablo\_adi (sütunadi1 [,sütunadi2,……..])

VALUES (deger1 [,deger2, ……..])

**Örnek:**

INSERT INTO PERSONEL (AD,SOYAD)

VALUES (‘Ceylin’,’Yılmaz’)

şeklindeki ifadeyle Personel tablosunun ad ve soyad alanlarına yeni değerler ekler.

**1.3.3. UPDATE Komutu**

Kayıtları güncellemek için kullanılır. Hangi kayıtların güncelleneceği bir koşul veya koşullarla belirtilebilir.

 **Genel Yazımı**

UPDATE tablo\_adi SET alan\_adi=deger WHERE şart

**Örnek:**

UPDATE PERSONEL SET AD=’Ceylin’ WHERE SOYAD=’Yılmaz’

biçimindeki bir bildirim soyadı Yılmaz olan kayıtların ad bilgisini Ceylin olarak değiştirir.

Eğer birden fazla Yılmaz soyadı olsaydı hepsinin ad alanı Ceylin olarak değiştirilecekti.

SET sözcüğü değiştirilecek kolonları ve değerleri belirtir. WHERE sözcüğü ise değiştirilecek satırı belirtir.

**1.3.4. DELETE Komutu**

Tablodan kayıt silmek için kullanılır.

 **Genel Yazımı**

DELETE FROM tablo\_adi

WHERE şart

**Örnek:**

DELETE FROM PERSONEL

WHERE SOYAD=’Yılmaz’

SOYAD değeri Yılmaz olan tüm kayıtları siler.

**1.4. Veri Kontrol Dili (DCL)**

DCL, bir veri tabanı ile ilişkili kullanıcıları ve rollerin izinlerini değiştirmek için kullanılır. Diğer bir deyişle verilere erişim yetkilerini düzenlemede kullanılır.

Temel komutları aşağıdaki şekildedir.

**Temel Komutlar Açıklama**

GRANT Bir kullanıcının verileri kullanmasına ve T-SQL komutlarını çalıştırmasına izin verir.

DENY Bir kullanıcının verileri kullanmasını kısıtlar.

REVOKE Daha önce yapılan tüm kısıtlama ve izinleri iptal eder.

DCL komutlarını kullanabilmek için SQL Server'da varsayılan değer (default) olarak yetki sahibi olan gruplar: sysadmin , dbcreator , db\_owner , db\_securityadmin 'dir.

Sunucuya dışarıdan bir erişim sağlamak için bir giriş (login) oluşturulmalıdır.

Bunun için;

CREATE LOGIN Tuncay1 WITH PASSWORD =’123456’

Bu rol ile veri tabanına bir kullanıcı olarak erişim için aşağıdaki satırlar yazılmalıdır:

CREATE USER Tuncay

FOR LOGIN Tuncay1

Eğer User adı ile Login adı aynı ise FOR LOGIN satırına gerek kalmaz.

Veri tabanında uygulama rolü oluşturulması için de aşağıdaki satırlar kullanılabilir:

CREATE APPLICATION ROLE Lab1

WITH PASSWORD = ‘sifre’

, DEFAULT\_SCHEMA=Lab1;

**1.4.1. GRANT Komutu**

Veri tabanı kullanıcısına, veri tabanı rolüne veya uygulama rolüne izinler vermek için kullanılan komuttur.

 **Genel Yazımı**

GRANT {ALL veya izinler}

ON {izin\_verilenler}

TO {hesaplar}

ALL

ifadesi, tüm hakların verilebileceğini gösterir.

**Örnek:**

Tuncay1 adlı kullanıcıya tablo oluşturma izni şöyle verilebilir:

GRANT CREATE TABLE

TO Tuncay1

Aynı anda Lab1 rolüne de aynı izin verilebilir.

GRANT CREATE TABLE TO Lab1, Tuncay1

**Örnek:**

Bir kullanıcıya hem veri tabanı hem de tablo oluşturma izni şöyle verilir:

GRANT CREATE DATABASE, CREATE TABLE TO Ahmet

**1.4.2. DENY Komutu**

Kullanıcıların haklarını kısıtlayan komuttur.

 **Genel Yazımı**

DENY { ALL veya izinler} TO {kullanıcılar}

**Örnek:**

DENY CREATE TABLE

TO Tuncay1

Tuncay1 adlı kullanıcının tablo oluşturma yetkisi kısıtlanır.

**Örnek:**

DENY SELECT ON PERSONEL TO Tuncay1

PERSONEL tablosunda Tuncay1 adlı kullanıcı SELECT komutuyla ilgili işlem yapamaz.

**1.4.3. REVOKE Komutu**

Daha önce yapılan tüm kısıtlama ve izinleri iptal eden komuttur. Bir nesneyi oluşturan kullanıcının REVOKE ile nesne üzerindeki yetkilendirme ve kullanma hakkı yok edilemez.

REVOKE komutunu, sys\_admin rolüne veya db\_owner, db\_securityadmin sabit veri tabanı rollerine sahip kullanıcılar ve nesne için dbo olan kullanıcı çalıştırabilir.

 **Genel Yazımı**

REVOKE {ALL veya izinler} {TO veya FROM} {hesaplar}

**Örnek:**

REVOKE ALL TO PUBLIC

PUBLIC rolüne verilmiş olan tüm yetkiler kaldırılır.

**Örnek:**

REVOKE SELECT ON PERSONEL TO PUBLIC

PUBLIC rolüne PERSONEL tablosunda seçim için verilen izin kaldırılır.

**2. T-SQL İLE ÇALIŞMAK**

**2.1. Değişkenler**

Değişken, verilerin bellekte geçici olarak kaydedilmesini ve gerektiğinde kullanılmasını sağlayan değerdir. T-SQL kullanmanın en büyük kolaylıklarından biri de değişken kullanımına olanak tanımasıdır. Burada ifade edilen; değişken diğer tüm programlama dillerinde yer alan bir veri tipi ile sınırlandırılmış, oluşturulmasının ardından hafızada belli bir yer kaplayan, üzerine veri ataması yapılabilen ve daha sonra ismi kullanılarak program içerisinden çağrılıp kullanılabilecek yapıdır.

SQL Server'da da değişkenler yerel ve genel olmak üzere ikiye ayrılır.

Yerel değişkenler, "@" ön eki ile tanımlanır (@değişken). Genel değişkenler ise SQL Server tarafından tanımlanmıştır ve kullanıcı tarafından oluşturulamaz. "@@" ön eki ile tanımlanırlar (@@SERVERNAME). Genel değişkenler genellikle SQL Server hakkındaki bilgileri verir. SQL Server'da tanımlanmış birçok genel değişken vardır.

**2.1.1. Nesne ve Değişken İsimlendirme Kuralları**

Nesne veya değişkene bir isimlendirme yaparken aşağıdaki kurallara dikkat etmelisiniz:

 Harf veya alt çizgi (\_) ile başlamalıdır.

 Türkçe karakterler ve boşluk isimlendirmede kullanılmamalıdır.

 Değişken ismi SQL’de özel anlamı olan sembollerle (@, @@, #, ##, $) başlamamalıdır.

 T-SQL komutları değişken ismi olarak verilmemelidir (SELECT,UPDATE vb).

 SQL ifadeleri prensip olarak büyük harfle yazılır.

 Nesne isimleri kısa ve anlamlı olmalıdır.

 Nesne isimlendirilirken işlerin kolaylaştırılması açısından tekil isim tercih edilmelidir (TabloOgrenciler yerine tblOgrenci gibi).

 NULL terimi, daha önce hiçbir şey girilmemiş (değersiz) anlamındadır.

Klavyedeki SPACE (ASCII 32) tuşu ile NULL aynı değerleri içermez. NULL boş veya bilinmeyen değerler için kullanılır.

**2.1.2. Değişken Tanımlama**

SQL Server'da değişkenler DECLARE ifadesi kullanılarak oluşturulur.

 **Yazım Şekli**

DECLARE @degisken\_adi <veri\_tipi> [(boyut)]

**Örnek:**

DECLARE @ogr\_no VarChar(10)

DECLARE @tckimlik\_no int

Aralara virgül koyarak da birden fazla değişkeni tek bir DECLARE ifadesi ile oluşturabilirsiniz.

**Örnek:**

DECLARE @ogr\_no varchar(10),@tckimlik\_no int

Varchar, int türlerinde değişken tanımlayabildiğiniz gibi tablo türünde değişken de tanımlayabiliriz.

**Örnek:**

DECLARE @değişken\_adı TABLE (tablo tanımı)

şeklindedir.

Bir değişken oluşturulduğunda NULL değere sahiptir. Değişkenlere değer atamanın SET, SELECT ve tablolar için INSERT INTO gibi birkaç farklı şekli vardır.

 **SET ifadesi kullanılarak değişkene değer atama**

SET @değişken\_adi=değer

şeklinde yapılır.

 **SELECT ifadesiyle değer atama**

SELECT @değişken\_adı=değer

şeklinde yapılır.

 **Tablo değişkenlere INSERT INTO ifadesi ile değer atama**

INSERT INTO @tablo\_degisken SELECT adi, soyadi FROM person

ifadesi ile person tablosunun adı ve soyadı sütunlarının içerdiği değerlerden oluşan bir tabloyu @tablo\_değişken adlı değişkene atamış olursunuz.

**2.1.3. Açıklama Satırları**

T-SQL'de bir satırın dikkate alınmamasını istiyorsanız "--" kullanabilir ya da /\*........\*/ kullanabilirsiniz.

**Örnek:**

-- CREATE TABLE ogrenci

veya

/\* CREATE TABLE ogrenci \*/

şeklindeki kullanımlardan birini seçebilirsiniz.

**2.2. Yığın Kavramı**

SQL Server’da yığın, sorguların sırayla işleme alınması demektir. Çalışma esnasında SQL Server’a gönderilen birden fazla sorgu yığınlar hâlinde ele alınır.

**2.2.1. GO Komutu**

Bir yığının sonunu belli etmek için **GO** komutu kullanılır. Bir yığın SQL Server’da işlenmeye başladığı anda önce Parse(ayrıştırma) edilir. Daha sonra derlenerek (Compile) çalıştırılır(Execute).

 **Genel Yazımı**

Komutlar

Komutlar

GO

şeklindedir.

**2.2.2. USE Komutu**

T-SQL’de çalışacağınız veri tabanını belirme işlemi için bu komut kullanılmalıdır USE komutuyla hangi veri tabanı üzerinde işlem yapılacağı belirtilir.

 **Genel Yazımı**

USE Veritabani\_Adi

şeklindedir.

**Örnek:**

USE master

**2.2.3. PRINT Komutu**

Değişkenlerin değerlerini, hataları vb. diğer ihtiyaç duyulan olaylarda PRINT komutu kullanılır. DEBUG işlemi için ihtiyaç duyulan bir komuttur.

 **Genel Yazımı**

PRINT @degisken\_adi

şeklindedir.

**Örnek:**

USE Deneme

GO

DECLARE @ad VARCHAR(10)

SELECT @ad=’Mustafa’

GO

PRINT @ad

GO

ad değişkeninin içeriği PRINT ile gösterilmiş olacaktır.

**2.3. İşlem Operatör Türleri**

T-SQL’de işlem yapabilmek için bazı operatörlere ihtiyaç duyulur.

**2.3.1. Karşılaştırma Operatörleri**

Karşılaştırma ifadesinde karşılaştırılan verilerin türü aynı olmalıdır. Yani karakter veri türü ile ancak karakter türünde başka bir veri; bir sayısal veri ile ancak sayısal olan başka bir veri karşılaştırılabilir.

**OPERATÖR ANLAMI**

< Küçük

> Büyük

= Eşit

<= Küçük veya eşit

>= Büyük veya eşit

<> Eşit değil

!= Eşit değil

LIKE Metin Karşılaştırma Operatörü

**Örnek:**

Öğrenci veri tabanında Tablo 2’deki ikinci notu 56’dan küçük olan öğrencileri listeleyen T-SQL kod satırlarıdır.

SELECT \*

FROM tablo2

WHERE nt2<=56

**Örnek:**

Adı Ali olmayan kayıtları listelemek için kullanılan T-SQL kod satırlarıdır.

SELECT \*

FROM Personel

WHERE ad<>’Ali’

 **Joker Karakterler**

Sadece LIKE operatörüyle kullanılan joker karakterler, bir veya daha fazla harfin yerine geçer. Belirli aralıklardaki belli harfle başlayan ve biten sorgularda joker karakterler kullanılır.

**Joker Karakterler Anlamı**

% Birden fazla harf ya da rakamın yerini tutar.

\_ Bir tek harf veya rakamın yerini tutar.

[HARF] Herhangi bir harf yerine gelebilecek harfleri belirtir.

[^HARF] Herhangi bir harf yerine gelemeyecek harfleri belirtir.

[A-Z] A ile Z arasındaki harfleri belirtir.

**Örnek:**

tablo1 tablosundaki ad alanı içinde baş tarafı “Er” ile başlayan isimleri listeleyen TSQL kodlarıdır.

SELECT \*

FROM tablo1 WHERE ad LIKE '%Er%'

**Örnek:**

tablo1 tablosundaki soyad alanı içinde baş tarafı “DE” ile başlayan soyadları listeleyen T-SQL kodlarıdır.

SELECT \*

FROM tablo1 WHERE soyad LIKE '%DE%'

**2.3.2. Mantıksal Operatörler**

NOT, OR ve AND mantıksal operatörleri yardımı ile birden çok koşulun gerçekleşmesine bağlı olarak ifade edilebilecek karmaşık ya da birleşik koşullu listelemeleri gerçekleştirmek mümkün olmaktadır. BETWEEN ifadesi de AND operatörü ile aynı işlemi gerçekleştirebilmektedir.

**Örnek:**

Doğum tarihi 1970’ten önce olan, maaşı 700 – 1200 arasında olan personeli listeleyen kodlardır.

SELECT \*

FROM Person

WHERE dogum\_tarih<{01/01/70} AND

Maas>=700 AND maas<=1200

**Örnek:**

BETWEEN..AND yapısıyla, maaşı 700 ile 1200 YTL arasında olanları gösteren T\_SQL kodlarıdır.

SELECT \*

FROM Person

WHERE maas BETWEEN 700 AND 1200

**Örnek:**

Doğum tarihi 1970’ten büyük ve cinsiyeti erkek olan veya doğum tarihi 1975’ten büyük ve cinsiyeti kadın olan personeli listeleyen T-SQL kodlarıdır.

SELECT \*

FROM Person

WHERE dogum\_tarih>={01/01/70} AND cinsiyet=Erkek

OR dogum\_tarih>={01/01/75} AND cinsiyet=Kadın

**NOT:** AND operatörü OR operatörüne göre daha önceliklidir.

**Örnek:**

Sinif tablosunda adı Ali olmayanları listeleyen T-SQL kodlarıdır.

SELECT \* FROM Sinif

WHERE NOT ad=’Ali’

**Örnek:**

SELECT \*

FROM Ogrenci

WHERE bolum=’Bilgisayar’ OR bolum=’Elektronik’ OR bolum=’Elektrik’

OR operatörü yerine In operatörü de kullanarak kod satırlarını yazabiliriz.

SELECT \*

FROM tablo1

WHERE bolum In ('Bilgisayar','Elektronik','Elektrik')

**2.3.3. Aritmetiksel Operatörler**

T-SQL’de kullanılan aritmetiksel operatörler şunlardır.

**OPERATÖR İŞLEVİ**

% Mod alma

\* Çarpma

/ Bölme

+ Toplama

- Çıkarma

SELECT komutu ile veri tabanında mevcut tablolardan listeleme yaparken tabloda ayrı bir sütun (alan) olarak yer almamış ve ancak bir hesaplama sonucunda üretilebilecek bilgileri de listeleme içine almak mümkündür.

**Örnek:**

Ogr\_Notlar tablosunda notların not ortalamasını hesaplayan T-SQL kodlarıdır.

SELECT (nt1+nt2+nt3) /3

FROM Ogr\_Notlar

Öncelik sırası, matematikte ve diğer bilgisayar dillerinde olduğu gibidir. Üs alma, hepsinden öncedir. Sonra çarpma (\*) ve bölme (/) gelir. Toplama (+) ve çıkarma (-) en son önceliklidir. Parantez kullanılarak öncelik sırası değiştirilebilir.

**2.4. Fonksiyonlar**

**2.4.1. Kümeleme Fonksiyonları**

SQL, tablo içinden çeşitli matematiksel işlemlerin sonucunu otomatik olarak üretmeyi sağlayan fonksiyonlara sahiptir.

 **SUM Fonksiyonu**

Fonksiyonla belirtilen sütun ile ilişkili olarak toplama işlemini yapar.

**Örnek:**

Ogr\_Notlar tablosunda nt1 sütununun not toplamını hesaplayan T-SQL kodlarıdır.

SELECT SUM(nt1)

FROM Ogr\_Notlar

**Örnek:**

Maaşları 500 YTL’nin altında olan personelin maaşları toplamını hesaplayan T-SQL kodlarıdır.

SELECT SUM(maas)

FROM Person

WHERE maas<500

 **AVG Fonksiyonu**

Aritmetiksel ortalama (avarage) hesaplamak için kullanılır.

**Örnek:**

İşçilerin maaşları ortalamasını veren T-SQL kodlarıdır.

SELECT AVG(maas)

FROM Person

Bu fonksiyon ile de koşula bağlı olarak hesaplatma yaptırılabilir.

 **MAX Fonksiyonu**

Tablo içinde, belirtilen sütun (alan) içindeki en büyük değeri bulur.

**Örnek:**

Ogr\_Notlar tablosu içindeki en yüksek öğrenci not ortalamasını veren T-SQL kodlarıdır.

SELECT MAX(not\_ortalama)

FORM Ogr\_Notlar

 **MIN Fonksiyonu**

Tablo içinde belirtilen sütun (alan) içindeki en küçük değeri bulur.

**Örnek:**

Ogr\_Notlar tablosu içindeki en düşük öğrenci not ortalamasını veren T-SQL kodlarıdır.

SELECT MIN(not\_ortalama)

FORM Ogr\_Notlar

 **COUNT Fonksiyonu**

Tablo içerisinde herhangi bir sayma işlemi gerçekleştirmek için kullanılır.

**Örnek:**

Ogr\_Notlar tablosu içindeki öğrenci not ortalamalarının sayısını veren T-SQL kodlarıdır.

SELECT COUNT (not\_ortalama)

FROM Ogr\_Notlar

COUNT fonksiyonu, DISTINCT sözcüğü ile de kullanılabilir. DISTINCT, her bir

kaydın tekil olarak yer alması istendiğinde bu deyim kullanılır. Normalde SELECT ile aynı

özelliğe sahip birden fazla kayıt listelenebilir.

**Örnek:**

Bir okulda birden fazla bölüme kayıtlı öğrenci olabilir. Sadece okuldaki bölüm adlarını listeleyen T-SQL kodlarıdır.

SELECT DISTINCT bolum

FROM Ogrenci

**Örnek:**

Bir okulda birden fazla bölüme kayıtlı öğrenci olabilir. Sadece okuldaki bölümlerin sayısını listeleyen T-SQL kodlarıdır.

SELECT COUNT (DISTINCT bolum)

FROM Ogrenci

COUNT komutunda, \* karakterinin kullanılması, bütün sütunların işleme sokulmasını sağlar.

**2.4.2. T-SQL’de Gruplandırma**

“GROUP BY” yardımcı sözcüğü bir alana göre kayıtları gruplamak için kullanılır.

**Genel kullanımı** aşağıdaki gibidir:

SELECT [ DISTINCT | ALL ] <sütun(lar)> FROM <tablo adı (lar)>

[ WHERE <şart (lar)> ]

[ GROUP BY <sütunlar>]

**Örnek:**

Öğrenci notları tablosunda öğrenci numarasına göre her bir öğrencinin almış olduğu not ortalamalarını gösteren T-SQL kodlarıdır.

SELECT ogrno, AVG(ort)

FROM tablo2

GROUP BY ogrno



**Örnek:**

Öğrenci notları tablosunda öğrenci numarasına ve bölüm adına göre her bir öğrencinin almış olduğu not ortalamalarını gösteren T-SQL kodlarıdır.

SELECT ogrno,bolum,MAX(ort)

FROM Tablo2

GROUP BY ogrno,bolum



Gruplandırarak kümeleme fonksiyonlarını uygularken koşul da verilebilir. Bu durumda, grup üzerindeki hesaplamalarla ilişkili koşul belirtirken **HAVING** sözcüğünü kullanmak gerekir.

**Örnek:**

Öğrenci notları tablosunda öğrenci numarasına göre her bir öğrencinin almış olduğu not ortalamasını 70’den büyük gösteren T-SQL kodlarıdır.

SELECT ogrno,AVG(ort)

FROM Ogr\_Notlar

GROUP BY ogrno

HAVING AVG(ort)>70



HAVING sözcüğü, SELECT komutunda GROUP BY sözcükleri bulunmadığı zaman

geçersizdir. HAVING sözcüğünü izleyen ifade içinde SUM, COUNT (\*), AVG, MAX ya da

MIN gibi kümeleme fonksiyonlarından en az biri bulunmalıdır.

WHERE sözcüğü bir tablonun tek tek satırları üzerinde işlem yapan koşullar için

geçerli iken HAVING sözcüğü ,sadece gruplanmış veriler üzerindeki işlemlerde geçerlidir.

**Örnek:**

Öğrenci notları tablosunda öğrenci numarasına göre bölümü “Bilgisayar” ve

öğrencinin almış olduğu not ortalaması 65’ten büyük olanları gösteren T-SQL kodlarıdır.

SELECT ogrno,bolum,AVG(ort)

FROM Ogr\_Notlar

WHERE bolum='Bilgisayar'

GROUP BY ogrno,bolum

HAVING AVG(ort)>65



 **COMPUTE ve COMPUTE BY Deyimleri**

Toplama fonksiyonunu kullanarak sonuç olarak bir toplam değeri üretir. COMPUTE deyimi her SELECT ifadesiyle kullanılır. COMPUTE BY ise ORDER BY deyimine gerek duyar.

**Örnek:**

Öğrenci notları tablosundaki öğrenci numarasına göre ortalama sütunundaki değerleri toplayarak bir sonuç üreten T-SQL kodlarıdır.

SELECT ogrno, ort

FROM Ogr\_Notlar

ORDER BY ogrno

COMPUTE SUM(ort)



**2.4.3. Tarih ve Zaman Fonksiyonları**

Tarih ve zaman üzerinde işlem yapmayı sağlayan fonksiyonlardır.

 **GETDATE () Fonksiyonu**

Şimdiki tarih ve saat değerini gösterir.

**Örnek:**

SELECT GETDATE()

 **DATEADD () Fonksiyonu**

Verilen bir tarihe istenilen sayıda bir tarih bilgisi eklemek için kullanılır.

DATEADD fonksiyonunda

dd gün

mm ay

yy yıl anlamındadır.

**Örnek:**

**Tarihe gün eklenmesi**:Belirtilen tarih bilgisine 90 gün ekleyen T-SQL kodudur.

SELECT DATEADD(dd, 90, '02.10.2007')

**Tarihe 2 ay eklenmesi:**

SELECT DATEADD(mm, 2, '02.10.2007')

**Tarihe 2 yıl eklenmesi:**

SELECT DATEADD(yy, 2, '02.10.2007')

 **DATEDIFF () Fonksiyonu**

Belirtilen iki tarih arasındaki gün sayısını göstermektedir.

**Örnek:**

SELECT DATEDIFF(dd, '04.04.1974', '02.10.2007')

Ayrıca hafta için wk, saat için hh, dakika için mi, saniye için ss kullanılabilir.

 **DATEPART () Fonksiyonu**

Tarihle ilgili sayısal bilgilerin alınmasını sağlar.

**Örnek:**

SELECT DATEPART(dd, '01.04.1974')

SELECT DATEPART(mm, '01.04.1974')

SELECT DATEPART(yy, '01.04.1974')

**2.4.4. Karakter Fonksiyonları**

Karakter alanlarla ilgili işlem yapmak için bu fonksiyonlar kullanılır.

 **CHAR ()**

ASCII kodu verilen karakteri görüntüler.

SELECT CHAR(65) A harfini verir.

SELECT ASCII(A) 65 rakamını verir. Değer int tipindedir.

 **CHARINDEX()**

Bir metin içerisindeki metin parçasını istenilen konumdan itibaren arar.

SELECT CHARINDEX ('bu',@soyad,1)



 **LEFT ()**

Metnin baş taraftan itibaren istenilen sayıdaki harflerini alır.

SELECT LEFT (@ad,2)



 **LEN ()**

Metnin uzunluğunu veren fonksiyondur.

SELECT LEN(@soyad)



 **LOWER ()**

Metni küçük harfe çevirir.

****

 **LTRIM ()**

Metnin başında bulunan boşlukları siler.



 **RIGHT ()**

Metnin sonundan istenilen kadar karakteri almak için kullanılır.

****

 **RTRIM ()**

Metnin sonundaki boşlukları siler.

****

 **REVERSE ()**

Metni ters çevirir.

****

 **SUBSTRING ()**

Bir metinde belirtilen karakterden itibaren belirli sayıda karakter almak için kullanılır.

****

 **UPPER ()**

Metnin tümünü büyük harfe çevirir.

