**SQL KONU BAŞLIKLARI;**

**0. SQL (STRUCTURED QUERY LANGUAGE - YAPILANDIRILMIŞ SOLGU DİLİ)**

**1. VERİ TİPLERİ**

**A.TAM SAYILAR (BIT, TINYINT, SMALLINT. INT, BIGINT)**

**B.ONDALIKLI SAYILAR (DECIMAL-NUMERIC, FLOAT)**

**C.METİNSEL İFADELER (CHAR, VARCHAR, NCHAR, NVARCHAR,)**

**D. BİNARY SAYILAR (BINARY, VARBINARY)**

**E.PARASAL VERİ TİPLERİ (SMALLMONEY, MONEY)**

**F.TARİH - ZAMAN VERİ TİPLERİ (DATE, TIME, SMALLDATETIME, DATETIME, DATETIME2, DATETIMEOFFSET)**

**G. DİĞER VERİ TİPLERİ (SQL\_VARIANT)**

**2. SQL KOMUTLARI (DQL, DDL, DML, DCL, TCL)**

**A. DQL (SELECT) - (DATA QUERY LANGUAGE)**

**B. DDL (CREATE, DROP, ALTER, TRUNCATE, COMMENT, RENAME) - (DATA DEFINATION LANGUAGE)**

**C. DML (SELECT, INSERT, UPTADE, DELETE, MERGE) - (DATA MANUPLATION LANGUAGE)**

**D. DCL (GRANT, REVOKE, DENY) - (DATA CONTROL LANGUAGE)**

**E. TCL (TRANSACTION, COMMIT, ROLLBACK, SAVEPOİNT) - (TRANSACTION CONTROL LANGUAGE)**

**3. \*, FROM, WHERE, AS, TOP, LIKE, IS NULL, IS NOT NULL**

**4. MANTIKSAL OPERATÖRLER (AND, OR, IN, NOT IN, BETWEEN X AND Y, <, >, <=, >=, !=)**

**5. SIRALAMA (ORDER BY (ASC, DESC))**

**6. FİLTRELEME VE SAYMA (DISTINCT, COUNT)**

**7. REPLACE, LEFT(), RIGHT(), LEN(), UPPER(), LOWER(), SUBSTRINGS**

**8. MATEMATIKSEL FONKSİYONLAR (CEILING, FLOOR, ROUND, ABS, MAX, MIN, SUM, AVG, SQRT, POWER)**

**9. TARİH VE ZAMAN FONKSİYONLARI (DATEADD, DATEDIFF, YEAR(), MONTH(), DAY(), DATEPART(),GETDATE())**

**10. GRUPLAMA FONKSİYONU (GROUP BY (HAVING) (COUNT, MAX, MIN, SUM, AVG))**

**11. KÜMELEME FONKSİYONLARI FONKSİYONLARI (JOIN ( INNER JOIN (JOIN), LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN))**

**12. DİĞER KÜMELEME FONKSİYONLARI (UNION, UNION ALL, INTERSECT, EXPECT)**

**13. SUBQUERY (ALT SORGULAR)**

**14. TRANSACTION (BEGIN TRANSACTION, ROLLBACK, COMMIT) (TCL)**

**15. DECLARE (DECLARE @DEĞİŞKENADI VERİTÜRÜ)**

**16. OUTPUT (INSERTED, DELETED)**

**17. KONTROL YAPILARI (IF-ELSE YAPISI (EXİST, DEFAULT))  
18. CASE YAPISI (ELSE)**

**19. DÖNGÜLER WHILE YAPISI (CONTINUE, GOTO)**

**20. CAST CONVERT**

**21. WAITFOR (WAITFOR DELAY, WAITFOR TIME)  
22. CONSTRAINT (KISITLAYICILAR) (CHECK)**

**BUNUN DIŞINDA BİLİNMESİ GEREKENLER;**

**YORUM SATIRI -- /\* \*/**

**BİT, BYTE**

**ASCII KOD, UNICODE(N)**

**@@ERROR**

**SQL Veri Tipleri**

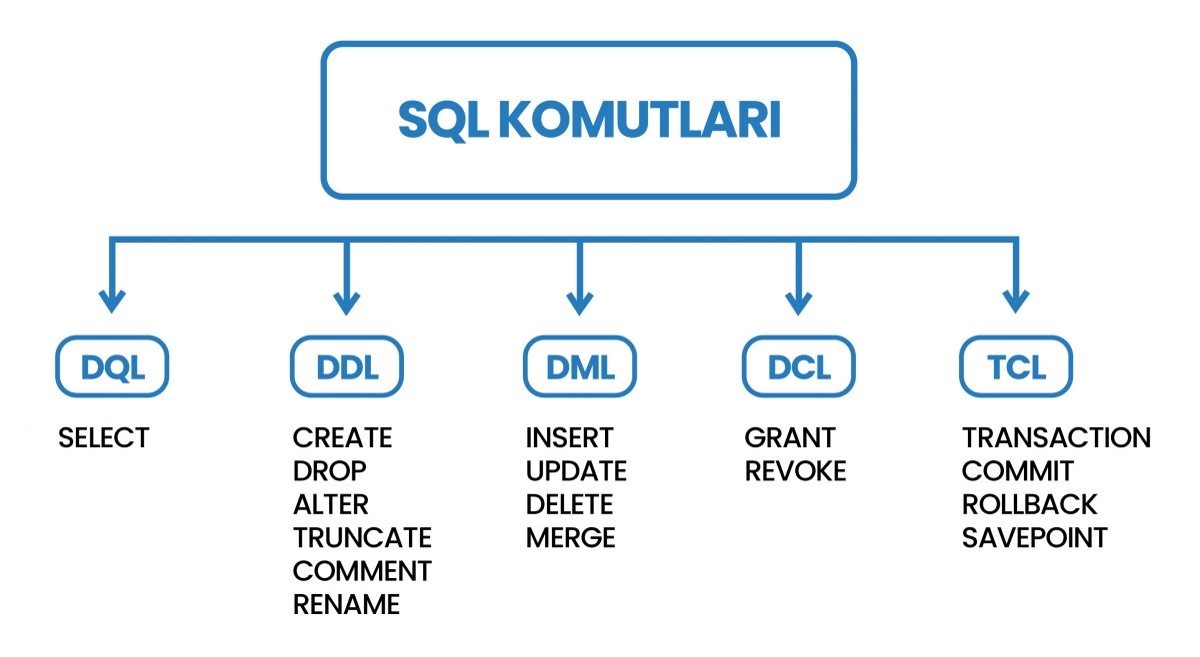
SQL (Structured Query Language), Türkçede yapılandırılmış sorgu dili anlamına gelir.

Unicode, tüm dillerin karakterlerini destekler. Unicode'u destekleyen veri tipleri "n" önekiyle başlar. Unicode karakterler 2 byte yer kaplar.

ANSI, standartlara uyan her karakter için 1 byte yer kaplarken, Türkçe’deki özel karakterler (ç,ğ,ş,ö) ANSI'de yer almaz.

**Tam Sayılar İçin Veri Tipleri**  
**bit:**  bir byte tam sayı veri tipidir. Mantıksal veri (0 veya 1, true/false için kullanılır).   
**tinyint:** 0-255 arası sayıları tutar. 1 byte.  
**smallint:** +-32000 arası sayıları tutar. 2 byte  
**int:**  +- 2 milyar arası sayıları tutar. 4 byte.  
**bigint:**  +-2^63 arası tam sayıları tutar.   
  
**Ondalıklı Sayılar İçin Veri Tipleri**  
**decimal/numeric(p,s):**  p: Sayının toplam basamak sayısı (tam sayı ve ondalık kısımlar dahil). s: Ondalıklı kısmın basamak sayısı. Belirtilen basamak sayısı kadar depolama kullanır. 38 basamağa kadar tutabilir. Boyutu 17 byte kadar çıkabiliyor.  
**float:** Çok büyük ve çok küçük kesirli sayılar için. Boyutu 4- 8 byte arası değişir.  
  
**Parasal Veri Tipleri**  
**smallmoney:** 4 byte uzunluğunda yaklaşık -214.000 ile 214.000 arası parasal değerleri tutmak için kullanılır. Ondalık tarafında 4 basamak saklar.  
**money:**  8 byte.922 trilyona kadar parasal değerleri tutar. Ondalık tarafında 4 hane saklar.  
 **Metin İçin Veri Tipleri**  
**char(n):** Unicode’u desteklemez. Belirtilenden(n) az karakter girilse dahi giriş yapılan boyut kadar yer kaplar. Veri giriş boyutları sabit olan veri kümelerinde kullanılması önerilir. Max 8000 karakter. 8000 byte. Örn char(5) 5 byte yer kaplar.  
  
**varchar:** Chardan farklı olarak verinin boyutu kadar yere kaplar. Birbirinden farklı uzunlukta veri girişi yapılacağı zaman kullanılması önerilir. varchar(MAX) kullanımı ile 2GB’a kadar depolama yapılabilir. Max 8000 karakter. 8000 byte.  
  
**nchar:** Unicode’u destekler. Char veri tipiyle özellikleri aynıdır. Max 4000 karakter. 8000 byte  
**nvarchar:** Unicode’u destekler. Verinin boyutu kadar yer kaplar. Max 4000 karakter. Max 8000 byte yer kaplar. nvarchar(MAX) kullanımı ile 2GB’a kadar depolama yapılabilr.  
  
 **Binary(İkilik) Veri Tipleri**  
**binary:** 1 ve 0 ları temsil eden ikilik taban veri tipidir.  
**varbinary:** 1 ve 0 ları temsil eden ikilik taban veri tipidir. Binaryden farkı girilen karakter kadar yer kaplar.

**Tarih-Zaman Veri Tipleri**  
**date:** Sadece tarihi saklar (YYYY-MM-DD).  
**time:** Sadece zamanı saklar (HH:MM:SS).  
**smalldatetime:** 1900-2079 arası.  
**datetime:** Tarih ve zaman (1753-9999 arası).  
**datetime2:** Tarih ve zaman (0001-9999 arası).   
**datetimeoffset:** Zaman dilimlerini içerir, ülkelere göre zaman farkını hesaplar.  
  
**Diğer Veri Tipleri**  
**sql\_variant:** sayı,metin, binary gibi farklı veri tiplerini tutar.



SQL temelde 3 alt ifade grubundan oluşur.

1- DDL

2 -DML

3- DCL

SQL’in veritabanı işlemleri için kullanılan komutları 5 kategoride toplanabilir.

Veri sorgulama komutları

Tabloya veri ekleme, değiştirme ve silme komutları

Veritabanı nesneleri oluşturma, değiştirme ve silme komutları

Veritabanına ve veritabanı nesnelerine erişimi kontrol etme komutları

Veritabanının tutarlığını ve bütünlüğünü koruma komutları

**DDL - Data Defination Language   
Veri Tanımlama Dili – (Tablo database için)**  
DDL, veri tabanındaki nesnelerin yapısını oluşturmak ve değiştirmek için kullanılır. Bu nedenle SQL komutları; **CREATE, DROP, ALTER**, TRUNCATE, COMMENT, RENAME şeklindedir.

•CREATE: Veri tabanı nesnelerini **oluşturmak** için kullanılır.  
•ALTER:  Veri tabanı nesnelerini **değiştirmek** için kullanılır.

•DROP:  Veri tabanı nesnelerini **silmek** için kullanılır.

•TRUNCATE:  Tablodaki tüm kayıtları siler. IDENTITY değerini de sıfırlar.

•COMMENT: Yorum eklemek için kullanılır.

•RENAME:  Veri tabanı nesnelerini yeniden adlandırmak için kullanılır.  
  
IDENTITY özelliği, SQL Server'da bir sütunun otomatik olarak artan bir sayı ile doldurulmasını sağlar.

**IDENTITY Kullanımı**

IDENTITY(başlangıç\_değeri, artış\_miktarı) şeklinde tanımlanır:

* **başlangıç\_değeri** → İlk eklenen kaydın alacağı değeri belirler.
* **artış\_miktarı** → Her yeni eklenen kaydın kaç artırılacağını belirler.  
    
  **Primary Key (Birincil Anahtar) ve Foreign Key (Yabancı Anahtar) Nedir?**
* **🔹 Primary Key (Birincil Anahtar):**
* Bir tablodaki her satırı benzersiz şekilde tanımlayan sütundur.
* Aynı değerden **birden fazla** olamaz (UNIQUE olmalıdır).
* **NULL değer içeremez**.

**Foreign Key (Yabancı Anahtar):**

* Bir tablonun başka bir tablodaki birincil anahtarına (Primary Key) bağlanmasını sağlar.
* Veri tutarlılığı sağlar (örneğin, olmayan bir müşteriye sipariş eklenemez).

Foreign Key, bir **tablo oluşturulurken (CREATE TABLE)** veya **sonradan (ALTER TABLE)** eklenebilir.  
  
CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT, -- Customers tablosuna bağlanacak sütun

OrderDate DATE,

CONSTRAINT FK\_Orders\_Customers FOREIGN KEY (CustomerID)

REFERENCES Customers(CustomerID)

);  
  
ALTER TABLE Orders

ADD CONSTRAINT FK\_Orders\_Customers FOREIGN KEY (CustomerID)

REFERENCES Customers(CustomerID);

**DML - Data Manuplation Language   
Veri İşleme Dili -** **(Verileri değiştirmek için)**Veri işleme dilidir. Veri tabanındaki verileri saklamak, değiştirmek, silmek ve güncellemek için kullanılır. DML komutlarından **bazıları SELECT, INSERT, UPDATE, DELETE** ve MERGE’dir.  
•\*SELECT; tablodaki **veri seçer** ve raporlar.

• INSERT; tabloya veri **ekler.**

• UPDATE; tablodaki verileri **günceller.**

• DELETE; tablodaki verileri **siler.**   
•MERGE: İki tablo arasında veri senkrizasyonu sağlar

**DCL – Data Control Language  
Veri Kontrol Dili – (Yetki İşlemleri için)**

Veri kontrol dilidir. Veri tabanı üzerinde yetki işlemleri için kullanırız.

•VTYS’de tanımlı roller ve kullanıcılar için ifade ve nesne kullanma izni tanımlar.

•Temel 3 ifadeden oluşur: **GRANT, REVOKE, DENY**

• Grant; Veri tabanında **yetki vermek** için kullanırız. erişim izni verir.

• Revoke; Veri tabanında yetki **kısıtlamak** için kullanırız. erişim iznini kaldırır.

•Deny; **erişim engeller**.  
  
SELECT INTO komutu, bir sorgu sonucunda elde edilen verileri yeni bir tabloya kopyalamak için kullanılır. Bu komut, mevcut veritabanında yeni bir tablo oluşturur ve ardından bu tabloya sorgudan elde edilen verileri ekler.

**DQL - Data Query Language**

**Veri Sorgulama Dili**  
DQL (Data Retrieval) veri tabanından veri almak için kullanılır. SQL komutu **SELECT**’dir.

SELECT : Veri tabanından veri almak için kullanılır.

**TCL - Transaction Control Language**

**İşlem Kontrol Dili**    
TCL, Veri tabanı İşlem Yönetimi olarak da bilinir. İşlem Yönetimi, veri tabanı ile ilgili işlemi sürdürmek, yani veri tabanının ACID özellikleri için temel kuralları takip etmek anlamına gelir. İşlemin yalnızca iki sonucu vardır; başarı veya başarısızlık. SQL komutları **TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT** şeklindedir.   
DML komutları tarafından yapılan değişiklikleri yönetmek için TCL kullanılır. Dolayısıyla, DML komutları tarafından yapılan değişiklikler DCL komutları tarafından kontrol edilebilir.

•COMMIT: İşlemi gerçekleştirir.

•ROLLBACK: Hata durumunda işlemi geri alır.

•SAVEPOINT: İşlem içinde bir kayıt noktası oluşturur.

•SET TRANSACTION: İşlemin özelliklerini belirtir.

---------------------------------------------------------------------------------------------------------------  
**distinct:** Farklı olan kayıtları getirir.Direkt kayıtların adlarını getirir.  
**count komutu:**  Tabloda kaç satır olduğunu veya belirli bir sütunda kaç değer olduğunu sayar. Sonuç sayı **count\*)**: Tüm satırları sayar (NULL dahil).

**count(sütun\_adi)**: Belirli bir sütundaki NULL olmayan değerleri sayar.

**order by** küçükten büyüğe doğru (asc)(a..z, 123..)veya büyükten küçüğe doğru (desc)(z..a, 321..)şekilde sıralamak için.

asc yazmasak da order by artan şekilde sıralamaya programlanmış  
**asc - ascending (Artan)**  
**desc - Descending (Azalan)**  
**between** 10 and 20 deyince 10 ve 20yi de dahil eder.  
  
**as** takma isimler oluştururken kullanılır.  
  
**substring** fonksiyonu, bir metin (string) ifadesinden belirli bir alt dizeyi (substring) almak için kullanılır. substring(metin veya tablo adı, başlangıç karakter sayısı(1), uzunluk)  
ilk karakter 1dir.  
select substring ('Merhaba Dünya', 1, 3) as ‘AltDize’  
select substring (Ad\_Soyad, 1, 5) as ‘İlkBeşHarf’ from ogrenciTablosu  
  
**LEN(LENGHT)** kaç karakter sayısı olduğunu söylüyor.  
  
**replace** fonksiyonu metni yeni bir metinle değiştirir. Sorgu sonucu değişir. Tabloda değişklil yapmak. Tabloda değişiklik olması için update fonk kullanılmalı.  
(metin, eski\_dize, yeni\_dize)  
  
**abs** mutlak değer alma fonk. Negatif değerleri pozitif yapar. Açılım Absolute – Mutlak demek.  
select abs(-20) as ‘mutlak’  
select abs(SinavNetleri) as ‘Mutlak Net Sayıları’ from SinavBilgileri  
  
**ceiling** ondalık sayıları üste yuvarlar. Virgülden sonrası kaç old önemli değil  
**floor** ondalık sayıları alta yuvarlar.  
**round** ondalık sayıyı en yakın sayıya yuvarlar  
  
**AVG** ortalama hesaplar. MAX MIN de var.  
  
**Datediff** fonksiyonu, iki tarih arasındaki farkı bulur  
DATEDIFF(gün/ay/yıl, başlangıç\_tarihi, bitiş\_tarihi)  
**Dateadd**  
sorunun cevabı doğru mu? Person.Person tablosundaki kişilerin adlarını ve soyadlarını yeni bir sütun olarak ekleyin. select FirstName + '-' + MiddleName + '-' + LastName as 'İsim ve soyisim' from Person.Person order by FirstName --  
MiddleName sütunu NULL olabilir: Eğer bir kişinin MiddleName değeri yoksa (NULL ise), sonuçta birleşim (+) işleci NULL değer üretecektir. Bu durumda, bazı sonuçlar tamamen NULL dönebilir.

Bu sorunu çözmek için ISNULL veya COALESCE fonksiyonlarını kullanabilirsiniz. İşte düzeltilmiş hali:  
SELECT FirstName + '-' + ISNULL(MiddleName, '') + '-' + LastName AS 'İsim ve soyisim'

FROM Person.Person

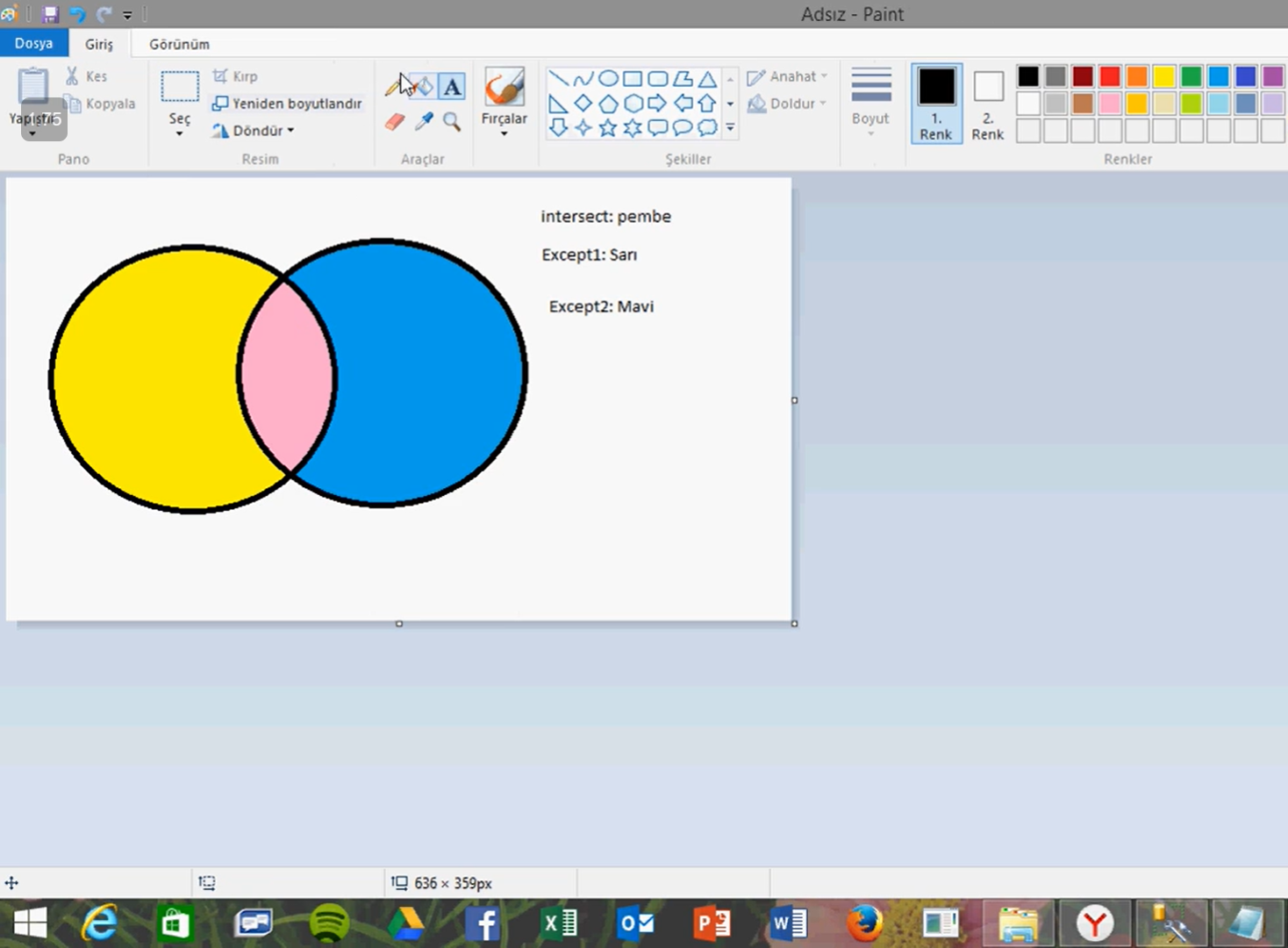
ORDER BY FirstName;  
**ISNULL(MiddleName, '')**: Eğer **MiddleName** sütunu NULL ise, boş bir string ('') olarak değiştirilir. Böylece sonuçta NULL yerine doğru bir birleşim sağlanır.  
  
GROUP BY komutu aynı değerlere sahip verileri gruplamamızı sağlar. Group by en çok (COUNT, MAX, MIN, SUM, AVG) fonksiyonlarıyla beraber kullanılır.  
  
  
ilk 5 kaydı getir dediğinde top ifadesini kullanıyotuz. O da select ifadesinden sonra oluyor.  
  
Join işlemleri  
SQL JOIN ifadesi, tabloları birleştirmek için kullanılır.  
SELECT *tablo\_adi.sutun\_adi*, ...

FROM *tablo\_A*

*{INNER JOIN | LEFT JOIN | RIGHT JOIN}* *tablo\_B* ON *tablo\_A.sutun\_adi* = *tablo\_B.sutun\_adi*;

* **(INNER) JOIN:** Inner join en çok kullanılan join türüdür ve her iki tablodaki ortak kayıtları döndürür. Bir başka ifade ile iki tablonun kesişimini döndürür.  
  **LEFT (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sol kayıtlar için kullanılır.
* **RIGHT (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sağ kayıtlar için kullanılır.
* **FULL (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sol ve sağ kayıtlar için kullanılır. LEFT ve RIGHT JOIN birleşimidir.  
    
  **Inner Join** yerine sadece **Join** yazmamız yeterli
* **Left Outer Join** yerine sadece **Left Join** yazmamız yeterli
* **Right Outer Join** yerine sadece **Right Join** yazmamız yeterli
* CROSS JOIN tablolar arasında yapılan birleştirme işleminde seçilen alanlar arasındaki tüm kombinasyonları sonuç tablosu olarak vermeyi sağlar. Veritabanlarında fazla kullanılmayan bir yöntem olan CROSS JOIN'in oluşturduğu sonuç tablosunda satır sayısı, alanların kartezyen çarpım sayısı kadardır. Örneğin; ilk alanda 4, ikinci alanda 3 satır varsa 3\*4=12 tane satır oluşacaktır.

aynı sütun adı bulunan iki tabloyu birleştirirken tablo.sütun yazmanız gerekir. Eğer farklıysa sadece sütun ismi yazabilirsin.  
   
 **RIGHT JOIN** ile çözebileceğiniz bir soruyu **LEFT JOIN** ile de çözebilirsiniz, ancak tablonun sıralanışını değiştirmeniz gerekir.

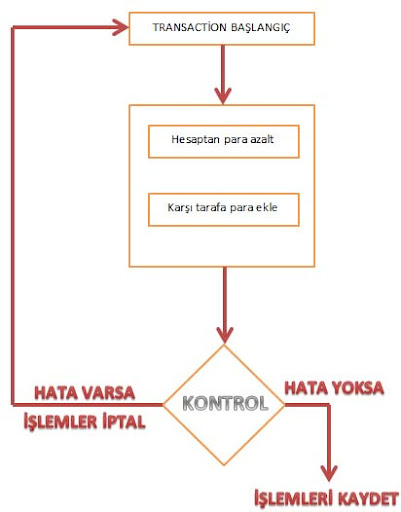
 **LEFT JOIN** ve **RIGHT JOIN** aynı mantıkla çalışır; fark, hangi tablodan tüm verilerin alındığıdır.  
**RIGHT JOIN** kullandığınızda, sağdaki tablo (yani **RIGHT JOIN**'in sağındaki tablo) tüm kayıtları alırken, soldaki tablodan (yani **RIGHT JOIN**'in solundaki tablo) sadece eşleşen kayıtları alır.  
  
eğer sorguda sadece join yazıyorsa inner joini kastediyordur.  
  
intersect   
except   
  
subquery   
in ve = dikkat   
select ProductName from Products where SupplierID in (select SupplierID from Suppliers where Country = 'UK')

Transaction   
**Transaction:** Birden çok işlemin bir arada yapıldığı durumlarda eğer parçayı oluşturan işlemlerden herhangi birinde sorun olursa tüm işlemleri iptal etmeyi sağlar.

**Örneğin** kredi kartı ile alışveriş olayında **transaction** olayı vardır, siz marketten ürün alırken sizin hesabınızdan para düşülecek marketin hesabına para aktarılacaktır bu işlemde hata olmaması gerekir ve bu işlem transaction blogu sayesinde yapılır. Yani kodlarımız sırayla işler bu esnada bir sorun çıkarsa bütün işlem **RollBack** sayesinde iptal edilir.

Transaction", daha küçük parçalara ayrılamayan en küçük işlem yığınına denir. Belirli bir grup işlemin arka arkaya gerçekleşmesine rağmen, işlemlerin seri ya da toplu halde değerlendirilip hepsinin düzgün bir şekilde ele alınması gerektiğinde kullanılır. "Transaction", prensip olarak ya bütün işlemleri gerçekleştirir ya da hiçbirini gerçekleştirmez.  İşlemlerden biri dahi başarısız olursa, bu prensip nedeniyle hiçbir işlem olmamış kabul edilir; ancak tüm işlemler başarılı olduğunda" transaction", içinde gerçekleşen tüm veri değişikliklerini onaylamış demektir.  
MSSQL'de **transaction (işlem)**, bir grup SQL işlemini bir arada çalıştıran bir mekanizmadır.  
Transaction’un amacı, **veri tutarlılığını ve güvenliğini sağlamak** ve **birden fazla adım içeren işlemleri bir bütün olarak ele almak**tır. Bu şekilde, sistemdeki herhangi bir aksaklık veya hata durumunda veritabanı tutarlı bir halde kalır.  
 BEGIN TRANSACTION: Bir transaction başlatır.

 COMMIT: Transaction'ı başarıyla tamamlar ve değişiklikleri kalıcı hale getirir.

 ROLLBACK: Transaction'ı iptal eder ve yapılan değişiklikleri geri alır.  
  
  
  
metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu  
  
  
  
**Ne zaman Kullanılır**

* Kritik işlelerde verinin güvenli bir şekilde veritabanına kaydedilmesi veya hata ile karşılaşıldığında sürecin geriye dönmesini istediğimiz durumlarda kullanılır. Örneğin ATM’den para çekerken arıza oluşması durumunda paramızın hesabımızdan çekilmemesini sağlamak.
* Hem ilgili birden çok ilgili işlem için hem de veritabanını aynı anda güncelleyen birden çok kullanıcı için veri bütünlüğünü korumak için kullanılır. Veritabanı işlemlerinin atomic olmasını sağlar.
* Bir tablo üzerinde değişiklik yapmak istediğimizde ve diğer tablolarda da tutarlılığı olmasını istediğimizde transaction yapısı kullanılır. Customer (Müşteri) ve Order (Sipariş) tabloları üzerinden örnek verecek olursak, Order tablosu Customer tablosuna bağlıdır. Bu durumda, müşteri silindiğinde onunla igili siparişlerinde silinmesi gerekmektedir. Eğer silinmezse müşterisiz sipariş şeklinde tutarsız veri kaydı oluşur. Bu gibi durumda transaction yapısı ile veri bütünlüğü sağlanır.

**Transaction’un Amacı Nedir?**

1. **Atomiklik Sağlamak (Tümü ya da Hiçbiri):**
   * Birden fazla işlem bir arada çalışırken, işlemlerin tamamı başarıyla gerçekleşir ya da hiçbir işlem yapılmamış gibi geri alınır.
   * Örneğin, bir para transferi sırasında:
     + Bir hesaptan para çekilir.
     + Diğer hesaba para yatırılır.
     + Eğer bir adım başarısız olursa, işlem geri alınır ve tutarsızlık olmaz.
2. **Hataları Yönetmek:**
   * Eğer bir hata oluşursa (örneğin bağlantı kopması veya kod hatası), yapılan değişiklikler **ROLLBACK** ile geri alınabilir.
   * Veriler, işlem öncesindeki durumuna döner.
3. **Veri Tutarlılığı:**
   * Veritabanında, işlemler arasında çakışma olmadan güvenli bir şekilde çalışmayı sağlar.
4. **Çok Kullanıcılı Ortamlarda Güvenlik:**
   * Aynı anda çalışan kullanıcılar veya işlemler nedeniyle veri tutarsızlıkları yaşanabilir. Transaction, bir işlemi diğerlerinden izole ederek bu durumu engeller.

Transaction’un amacı, **veri tutarlılığını ve güvenliğini sağlamak** ve **birden fazla adım içeren işlemleri bir bütün olarak ele almak**tır. Bu şekilde, sistemdeki herhangi bir aksaklık veya hata durumunda veritabanı tutarlı bir halde kalır. Daha iyi anlaman için aşağıdaki başlıklar faydalı olabilir:

**Transaction’un Amacı Nedir?**

1. **Atomiklik Sağlamak (Tümü ya da Hiçbiri):**
   * Birden fazla işlem bir arada çalışırken, işlemlerin tamamı başarıyla gerçekleşir ya da hiçbir işlem yapılmamış gibi geri alınır.
   * Örneğin, bir para transferi sırasında:
     + Bir hesaptan para çekilir.
     + Diğer hesaba para yatırılır.
     + Eğer bir adım başarısız olursa, işlem geri alınır ve tutarsızlık olmaz.
2. **Hataları Yönetmek:**
   * Eğer bir hata oluşursa (örneğin bağlantı kopması veya kod hatası), yapılan değişiklikler **ROLLBACK** ile geri alınabilir.
   * Veriler, işlem öncesindeki durumuna döner.
3. **Veri Tutarlılığı:**
   * Veritabanında, işlemler arasında çakışma olmadan güvenli bir şekilde çalışmayı sağlar.
4. **Çok Kullanıcılı Ortamlarda Güvenlik:**
   * Aynı anda çalışan kullanıcılar veya işlemler nedeniyle veri tutarsızlıkları yaşanabilir. Transaction, bir işlemi diğerlerinden izole ederek bu durumu engeller.

**Transaction Ne Zaman Kullanılır?**

Transaction, genellikle şu durumlarda kullanılır:

1. **Birden Fazla İşlem Birbiriyle Bağlantılıysa:**
   * Örneğin, bir e-ticaret uygulamasında:
     + Sipariş kaydı oluşturulur.
     + Stok bilgisi güncellenir.
     + Ödeme işlemi yapılır.
   * Eğer ödeme başarısız olursa, diğer işlemler de iptal edilmelidir.
2. **Veritabanı Tutarlılığı Gerektiren Kritik Durumlar:**
   * Banka işlemleri (para transferi, ödeme, borç tahsili gibi).
   * Stok yönetimi (ürün ekleme/çıkarma işlemleri).
3. **Hataların Geri Alınması Gerekiyorsa:**
   * Yanlış bir işlem veya beklenmeyen bir hata oluşursa, yapılan işlemleri geri almak için kullanılır.
4. **Çok Kullanıcılı Uygulamalar:**
   * Aynı tabloya farklı işlemler yapılırken veri çakışmasını önlemek için.

@@ERROR, **Microsoft SQL Server**'da kullanılan bir sistem değişkenidir. Bu değişken, bir SQL komutunun çalıştırılmasından hemen sonra oluşan **hata kodunu** döndürür.

* Eğer bir hata oluşmazsa, @@ERROR değeri **0** olur.
* Eğer bir hata oluşursa, bu hata için bir kod döndürülür.

T-SQL KODLARI – TRANSACTİON   
DECLARE ifadesi, bir değişken tanımlamak için kullanılır. Bu ifade, geçici olarak veri saklamak ve sorgular sırasında bu veriyi işlemek için kullanılır. Tanımlanan değişken yalnızca o **batch**, **stored procedure**, veya **trigger** içinde geçerlidir.  
  
  
**Temel Kullanım**

DECLARE @değişkenAdı veriTürü;

* @değişkenAdı: Değişkenin adı, her zaman @ ile başlar.
* veriTürü: Değişkenin alabileceği veri türüdür (ör. INT, VARCHAR, DATETIME).

declare ile geçici tablo oluşturma ile ilgili

* **Tablo değişkenleri** yalnızca geçerli oturum boyunca geçerlidir ve oturum sona erdiğinde otomatik olarak silinir.
* **tablo değişkenleri** otomatik olarak **DROP** edilmez. Oturum bitiminde sistem tarafından silinir.  
  declare ile oluşturulan tabloyu çalıştırmak istedğimizde sadece select \* from personel çalışırsa tablo çalışmaz tüm tablo ile ilgili yazdığın koddlar görünür olmalı.

CAST veya CONVERT fonksiyonları, veri türü dönüşümüne ihtiyaç duyduğunda kullanılır.  
@toplam değişkeni bir **int** türünde olduğu için, PRINT komutunda metinle birleştirirken veri türü uyumsuzluğu hatası alıyorsunuz. Çünkü **int** türü ile doğrudan metin birleştirme yapılamaz.

**CAST** veya **CONVERT** fonksiyonları kullanarak @toplam değişkenini metin (NVARCHAR) türüne dönüştürmelisiniz.

Output kavramı   
output kısmında hangi ifadeler kullanılabilir? örneğin inserted uptaded deleted gibi

SQL Server'da **OUTPUT** komutuyla birlikte kullanılan bazı temel **"fancy aliases"** (takma adlar) vardır. Bu ifadeler, işlem sonucu üzerinde işlem yaparken hangi veriye erişebileceğimizi belirler. Bu ifadeler şunlardır:

**1. INSERTED:**

* **INSERTED** alias'ı, bir **INSERT** veya **UPDATE** işleminde, **eklenen yeni veriyi** temsil eder.
* **INSERTED**, işlem sırasında **yeni satırlara** veya **güncellenen satırlara** ait bilgileri döndüren bir alias'tır.

**DELETED:**

* **DELETED** alias'ı, bir **DELETE** veya **UPDATE** işleminde, **silinen eski veriyi** temsil eder.
* **DELETED**, **DELETE** işleminde silinen satırları veya **UPDATE** işleminde eski değerleri döndürür.

Yani output işleminde güncelleme veri ekleme gibi işlemlerde inserted kullanırsın  
güncellenen veri için inserted kullanılır. Updated diye bir şey yok. Sadece inserted ve deleted var.  
  
IF ELSE YAPILARI  
bir IF...ELSE yapısında koşulu yazarken, **değişken tanımlamak zorunda değilsiniz**. SQL sorgularınızda, tablodaki bir sütun adı ile koşul oluşturabilirsiniz. Bunun için sütun adı doğrudan koşulda kullanılabilir.  
IF (koşul)

BEGIN

-- Koşul doğruysa yapılacak işlemler

END

ELSE

BEGIN

-- Koşul yanlışsa yapılacak işlemler

END  
  
EXISTS genellikle IF...ELSE, WHERE gibi yapılarla birlikte, belirli bir koşulun sağlanıp sağlanmadığını kontrol etmek amacıyla kullanılır  
temel kullanım  
IF EXISTS (SELECT 1 FROM tabloadi WHERE koşul)

BEGIN

-- Koşul sağlanıyorsa yapılacak işlemler

END

ELSE

BEGIN

-- Koşul sağlanmıyorsa yapılacak işlemler

END  
sqlde sütunları doğrudan bir IF yapısında kullanamazsınız. Bunun için declare kullanılır.   
ya declare kullanılır ya da sorgu yazılır

   
CASE YAPISI  
WHİLE DÖNGÜSÜ (FOR DÖNGÜSÜ YOK SQLDE)  
while döngüsü continue ve goto yapısıyla kullanımı  
fsrk ettim ki primary key foreign keyden bahsetmiyor onunla ilgili bilgi öğren ve alıştırma için tablolar yap büt sınavı gibi.  
  
SQL'de **CONTINUE** ve **GOTO** genellikle kontrol akışını değiştirmek için kullanılır. İşte basit bir açıklama:

**CONTINUE NEDİR?**

* **WHILE döngüsünde kullanılır.**
* **Bir koşul gerçekleştiğinde o döngünün geri kalan kısmını atlar ve bir sonraki iterasyona geçer.**
* **Döngüyü tamamen sonlandırmaz, sadece mevcut adımı atlar.**

**CONTINUE KULLANIM YAPISI**

1. Bir **WHILE döngüsü** yazarsın.
2. İçinde bir **IF koşulu** kullanırsın.
3. Eğer bu **IF koşulu sağlanırsa**, **CONTINUE** çalışır ve döngünün kalan kodlarını atlar.
4. Döngü bir sonraki adımdan devam eder.

Waitfor kavramı   
waitfor delay, waitfor time  
UNION UNION ALL  
merge  
ACID kavramı  
CONSTRACTIONS’LARA ÇALIŞ  
CONSTRAINT (kısıtlama), bir tabloya eklenen kurallar veya kısıtlamalardır.  
RIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT.

MAX, MIN, COUNT, AVG, SUM

MAX: en çok/büyük veriyi listeler

MIN: en az/küçük veriyi listeler

SUM: toplam

AVG: ortalama

COUNT: veri sayısı

LEFT, RIGHT, UPPER, LOWER, LEN

LEFT(sutun1, n) : parantez içinde belirtilen sütunun solundan belirtilen n karakter kadar listeler

RIGHT(sutun1, n) : parantez içinde belirtilen sütunun sağından belirtilen n karakter kadar listeler

LEN ( sutun1/ifade) : parantez içinde belirtilen sütun veya ifadenin karakter sayısını listeler

UPPER (sutun1/ifade): Belirtilen sütundaki ifadeyi büyük harf olarak listeler

LOWER( sutun1/ifade) : Belirtilen sütundaki ifadeyi küçük harf olarak listeler

TRIM(sutun1/ifade): başında veya sonunda ki fazla boşlukları kaldırarak listeler

RTRIM(sutun1/ifade) : ifadenin sonundaki/sağındaki fazla boşlukları kaldırarak listeler

LTRIM( sutun1/ifade): ifadenin başındaki/solundaki fazla boşlukları kaldırarak listeler

REPLACE(sutun1/ifade, değişecek\_bilgi, yeni\_bilgi): belirtilen sütundaki ifade değişikliğini sağl  
  
**NOTLAR  
Örnek Databaseler**   
https://red9.com/blog/sample-sql-databases/  
  
**YORUM SATIRI**

Bir satırı **yorum satırı** haline getirmek için ‘-- ’  
Toplu yorum satırı için /\* \*/  
Toplu bir şekilde - - (yorum satırı) eklemek için CTRL K + CTRL C tuş kombinasyonu   
Yorum satırlarını açmak için için CTRL K + CTRL U tuş kombinasyonu  
  
bir satıra **bookmark** eklemek için CTRL K + CTRL K   
  
insert hiç olmayan bir satıra veri girişi yapar  
update olan satırda güncelleme yapıyor.   
  
Eğer bir tabloda identity özelliği olan bir sütun varsa sütun adı belirtmeden İNSERT ile değer eklemeye çalıştığında hata verir.

ya identity özelliğini açıp manuel değerini gireceksin ya da sütun adı yazarken onu yazmayıp sistemin otomatik artan sıraya göre değer girmesini isteyeceksin.  
  
SQL sorgularında ondalık . ile girilirken tabloda , ile girilir.  
SQL sorgularında tarih değerleri tek tırnak içinde yazılır.  
  
**EXEC Nedir?**

EXEC (veya EXECUTE), **SQL Server'da** bir prosedür veya sorguyu çalıştırmak için kullanılan bir komuttur. Bir SQL komutunun veya saklı prosedürün (stored procedure) çalıştırılmasını sağlar.

**Örnek Kullanım:**

sql

KopyalaDüzenle

EXEC sp\_rename 'AdventureWorks2017.dbo.Customers', 'Clients';

Bu örnekte, sp\_rename adlı saklı prosedür çalıştırılmaktadır.

**sp\_rename Nedir?**

sp\_rename, **SQL Server'da** kullanılan bir saklı prosedürdür (stored procedure). Bu prosedür, bir veritabanı nesnesinin (tablo, sütun, index, vs.) adını değiştirmek için kullanılır.

**Söz Dizimi:**

sql

KopyalaDüzenle

EXEC sp\_rename 'eski\_ad', 'yeni\_ad', 'objenin\_türü';

* **eski\_ad** → Değiştirilecek nesnenin mevcut adı.
* **yeni\_ad** → Verilecek yeni ad.
* **objenin\_türü** → Değiştirilecek nesnenin türü (tablo, sütun, index, vs.). Eğer tür belirtilmezse, varsayılan olarak **tablo** kabul edilir.

**TRUNCATE TABLE** komutu:

* Tablodaki tüm verileri **silmekle** birlikte, **identity değerini sıfırlar** (eğer tablo IDENTITY özelliğine sahipse).
* **TRUNCATE** komutu daha hızlıdır çünkü veriyi tek tek satır bazında silmez, tüm tabloyu hızlıca temizler.

UNIQUE ifadesi, bir sütundaki her değerin **benzersiz** olmasını sağlamak için kullanılır. Yani, **Email** sütunundaki her e-posta adresi farklı olmalıdır ve tekrarlayan e-posta adreslerinin tablodaki birden fazla satırda yer almasına izin verilmez.

**Açıklama:**

* **UNIQUE** kısıtlaması, aynı değerin bir sütun içinde birden fazla kez bulunmasını engeller.  
    
  İNDEX İndeksler, veritabanlarında veri erişimini hızlandırmak için kullanılır. Bunu, bir kitabın arka kısmındaki **konu dizini** gibi düşünebilirsin: Kitabın içeriğine doğrudan gitmeden önce, dizin sayesinde hızlıca istediğin bilgiye ulaşabilirsin. Veritabanındaki indeksler de benzer şekilde, büyük veri setlerinde hızlı veri erişimi sağlar.  
  **CREATE INDEX** komutu, belirttiğin sütunlar üzerinde bir indeks oluşturur.

 **Sorgu Performansını Artırmak:**

* İndeksler, özellikle **SELECT** sorgularında, **WHERE**, **JOIN**, **ORDER BY** ve **GROUP BY** gibi işlemlerde hızlı veri aramayı sağlar. Örneğin, WHERE City = 'London' gibi bir sorgu indeksli sütunlarda daha hızlı çalışır.

 **Veri Filtreleme:**

* Veritabanındaki veriler çok büyük olduğunda, indeksler arama koşullarına göre veri filtreleme işlemini hızlandırır. Yani, aradığın belirli veriyi (örneğin bir şehir adı veya müşteri kimliği) hızlıca bulabilirsin.

 **Sıralama (Sorting) ve Gruplama (Grouping) İşlemleri:**

* İndeksler, veri sıralama ve gruplayarak raporlama gibi işlemleri hızlandırır. Örneğin, ORDER BY veya GROUP BY komutlarında indeksler, verilerin hızlıca sıralanmasını sağlar.

 **Eşsiz (Unique) Verilerin Kontrolü:**

* İndeksler, belirli sütunlar üzerinde **eşsiz (unique)** değerlerin korunmasına yardımcı olur. Bu, veritabanının tekrar eden verileri engellemesine olanak tanır. Örneğin, bir müşteri ID'si gibi benzersiz bir değeri kontrol etmek için indeksler kullanılır.