

Bölüm 3

**İleri Tasarım ve Uygulama**

1. Yapılandırılmış Sorgu Diline (SQL) Giriş
2. Gelişmiş SQL
3. Veritabanı Tasarımı

**Bölüm**

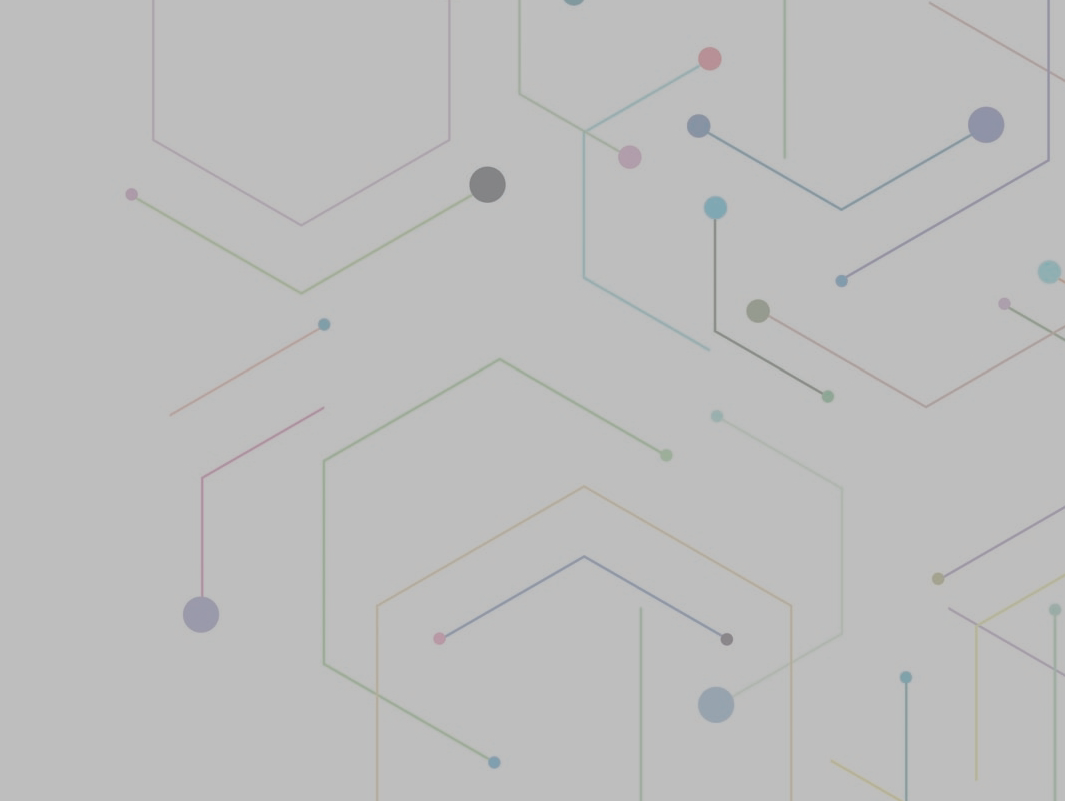
7

Yapılandırılmış Sorgu Diline (SQL) Giriş

**Öğrenme Hedefleri**

Bu bölümü tamamladıktan sonra şunları yapabileceksiniz:

* 1. **Bir veritabanından belirtilen veri sütunlarını alma**
  2. **Tek bir SQL sorgusunda birden fazla tabloya katılma**
  3. **Veri alımlarını karmaşık ölçütlerle eşleşen satırlarla kısıtlayın**
  4. **Satır grupları arasında veri toplama**
  5. **Diğer sorgulara dahil etmek üzere verileri önceden işlemek için alt sorgular oluşturma**
  6. **Dize, sayı ve tarih manipülasyonu için çeşitli SQL fonksiyonlarını tanımlama ve kullanma**
  7. **Bir SELECT sorgusu oluşturmanın temel ilkelerini açıklayın**



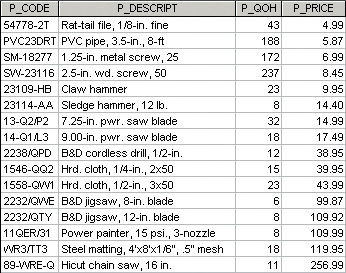
**Önizleme**

Bu bölümde, Yapısal Sorgu Dili'nin (SQL) temellerini öğreneceksiniz. *S-Q-L* veya *devamı* olarak telaffuz edilen SQL, kullanıcıların veritabanı ve tablo yapıları oluşturmasını, çeşitli veri işleme ve veri yönetimi türlerini gerçekleştirmesini ve yararlı bilgileri çıkarmak için veritabanını sorgulamasını sağlayan komutlardan oluşur. Tüm ilişkisel DBMS yazılımları SQL'i destekler ve birçok yazılım satıcısı temel SQL komut setine uzantılar .

Oldukça kullanışlı ve güçlü olmasına rağmen SQL, uygulamalar arenasında tek başına durmak için tasarlanmamıştır. SQL ile veri girişi mümkündür ancak veri düzeltmeleri ve eklemelerinde olduğu gibi zordur. SQL'in kendisi menüler, özel rapor formları, kaplamalar, açılır pencereler veya son kullanıcıların genellikle beklediği diğer özellikleri oluşturmaz. Bunun yerine, bu özellikler satıcı tarafından sağlanan geliştirmeler olarak mevcuttur. SQL, veri tanımlama (tablo ve dizin oluşturma) ve veri manipülasyonuna (ekleme, değiştirme) odaklanır,

silme ve veri alma). SQL programcıları için en yaygın görev veri erişimidir. İş gereksinimlerini karşılamak için bir veritabanından veri alma yeteneği, veritabanı uzmanları için en kritik becerilerden biridir. Bu bölümde veri alma konusu ayrıntılı olarak ele alınmaktadır.

|  |  |  |  |
| --- | --- | --- | --- |
| Birçok iş sorusu, tek bir tablo kullanılarak alınan veriler kullanılarak yanıtlanabilir  FROM cümlesinde. Ancak, diğer sorular birden fazla tablo içeren daha karmaşık bir FROM cümlesi gerektirir. FROM cümlesine birden fazla tablo dahil etmek, FROM içinde tabloların bir listesini sağlamak kadar basit değildir. Tabloların bir listesini eklemek, Bölüm 3'te tartışıldığı gibi bir Kartezyen çarpım oluşturacaktır. Bölüm 3'te de belirtildiği gibi, bir Kartezyen çarpım bir iş sorusu için gereken doğru sonuçları neredeyse hiçbir zaman üretmeyecektir. Bunun yerine, tablolar Bölüm 3'te de ele alınan bir tür JOIN işlemi kullanılarak birleştirilmelidir. Hatırlayacağınız gibi, veritabanlarında birçok birleştirme türü kullanılır ve doğru birleştirme türünü seçmek ve bunun için sözdizimini yazmak biraz göz korkutucu olabilir. JOIN işlemlerinin kullanımıyla ilgili daha ileri düzeydeki konuların tam bir açıklaması bu bölümün ilerleyen kısımlarında yer almaktadır.  **7-5 ORDER BY Cümle Seçenekleri**  **ORDER BY** cümlesi özellikle listeleme sırası sizin için önemli olduğunda kullanışlıdır. Sözdizimi  şöyledir: SEÇİNİZ *sütun listesi*  FROM *tablelist*  [ORDER BY *sütun listesi* [ASC | DESC]];  Sıra türünü (artan veya azalan) bildirme seçeneğiniz olmasına rağmen, varsayılan sıra artan sıradır. Örneğin, PRODUCT tablosunun içeriğinin P\_PRICE'a göre artan sırada listelenmesini istiyorsanız, aşağıdaki komutu kullanın:  SEÇİNİZ P\_CODE, P\_DESCRIPT, P\_QOH, P\_PRICE FROM ÜRÜN  ORDER BY P\_PRICE;  Çıktı Şekil 7.9'da gösterilmektedir. ORDER BY'nin artan bir fiyat listesi verdiğine dikkat edin. Şekil 7.9'daki listeleme ile daha önce Şekil 7.9'da gösterilen gerçek tablo içeriği karşılaştırıldığında  7.2, Şekil 7.9'da ilk olarak en düşük fiyatlı ürünün listelendiğini, ardından bir sonraki en düşük fiyatlı ürünün geldiğini ve bu şekilde ettiğini görebilirsiniz. Ancak, ORDER BY sıralanmış bir çıktı üretse de, gerçek tablo içeriği ORDER BY işleminden etkilenmez. | | | |
|  | **Şekil 7.9** | **Fiyata Göre Artan Sıralı Ürünler** |  |
|  | | | |



### ORDER BY

Çıktıyı sıralamak için yararlı olan bir SQL cümlesi bir SELECT sorgusunun (örneğin, artan veya azalan s rada).

ı

Azalan sırayı belirtmek için öznitelikten sonra DESC ekleyebilirsiniz. Ürünlerin fiyata göre azalan sırada sıralandığı bir liste oluşturmak için şunu girersiniz:

SEÇİNİZ P\_CODE, P\_DESCRIPT, P\_QOH, P\_PRICE FROM ÜRÜN

ORDER BY P\_PRICE DESC;

Sıralı listeler sıklıkla kullanılır. Örneğin, bir telefon rehberi oluşturmak istediğinizi varsayalım.

Üç aşamada sıralı bir dizi (soyadı, ad, adın baş harfi) üretebilmeniz yararlı olacaktır:

* + 1. göre sırala.
    2. Eşleşen içinde, ORDER BY ad.
    3. Eşleşen ad ve soyadlar içinde, ortanca baş harfe göre SIRALA.

Bu tür çok düzeyli bir sıralama dizisi **basamaklı sır**alama dizisi olarak bilinir ve ORDER BY cümlesinden sonra virgüllerle ayrılmış birkaç özniteliğin listelenmesiyle kolayca oluşturulabilir.

Basamaklı sıra dizisi, herhangi bir telefon rehberinin temelini oluşturur. Basamaklı sıra

dizisini göstermek için EMPLOYEE tablosunda aşağıdaki SQL komutunu kullanın:

SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_INITIAL, EMP\_AREACODE, EMP\_PHONE

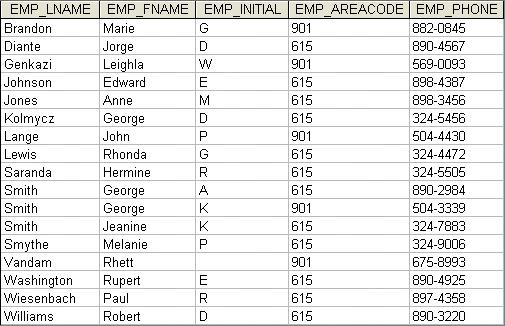
FROM ÇALIŞAN

ORDER BY EMP\_LNAME, EMP\_FNAME, EMP\_INITIAL;

Bu komut Şekil 7.10'da gösterilen sonuçları verir.

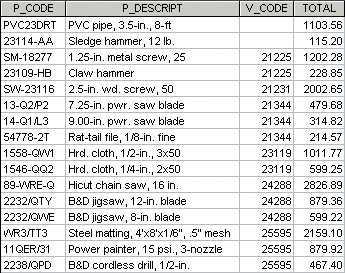
### basamaklı sıra dizisi

Tüm soyadların alfabetik olarak sıralandığı ve soyadlar içinde tüm ilk sıralandığı bir liste gibi bir dizi satır için iç içe sıralama dizisi.



ORDER BY cümlesi, özellikle DESC niteleyicisi çağrılabildiği için birçok uygulamada kullanışlıdır. Örneğin, en yeni öğelerin önce listelenmesi standart bir prosedürdür. Tipik olarak, fatura son ödeme tarihleri azalan sırada listelenir. Ya da bütçeleri incelemek istiyorsanız, önce en büyük bütçe satır kalemlerini listelemek muhtemelen yararlı olacaktır.

ORDER BY cümlesini diğer SQL işlemleriyle de kullanabilirsiniz. Örneğin, aşağıdaki komut dizisinde türetilmiş bir özniteliğin kullanıldığına dikkat edin:



**Not**

Sıralama sütununda null'lar varsa, RDBMS'ye bağlı olarak bunlar ya ilk ya da son sırada yer alır. Oracle, ORDER BY cümlesindeki null'ların sıralama davranışını değiştirmek için NULLS FIRST veya NULLS LAST seçeneğinin eklenmesini destekler. Örneğin, Oracle'da aşağıdaki komut satıcı kodlarını büyükten küçüğe doğru sıralanmış olarak döndürür, ancak null satıcı kodları listede en son görünür.

SEÇİNİZ FROM ORDER BY

V\_CODE, P\_DESCRIPT

ÜRÜN

V\_CODE DESC NULLS LAST;

SEÇİNİZ P\_CODE, P\_DESCRIPT, V\_CODE, P\_PRICE \* P\_QOH AS TOTAL FROM ÜRÜN

ORDER BY V\_CODE, TOPLAM DESC;

Çıktı Şekil 7.11'de gösterilmektedir. Sorgu sonuçları V\_CODE'a göre artan sırada sıralanır ve ardından eşleşen satıcı kodu değerleri içinde sonuçlar türetilmiş toplam değer özniteliğine göre azalan sırada sıralanır.

**Şekil 7.11 Türetilmiş Bir Özniteliğe Göre Sıralama**

**7-6 WHERE Cümlesi Seçenekleri**

Bu bölümde, arama kriterlerine kısıtlamalar ekleyerek SELECT komutuna nasıl ince ayar yapacağınızı öğreneceksiniz. Uygun arama koşullarıyla birlikte kullanıldığında SELECT, verileri bilgiye dönüştürmenizi sağlayan inanılmaz güçlü bir araçtır. Örneğin, aşağıdaki gibi soruları yanıtlayan sorgular oluşturabilirsiniz: "Belirli bir satıcı tarafından hangi ürünler tedarik edildi?", "Hangi ürünlerin fiyatı 10 doların altında?" ve "5 Ocak 2022 ile 20 Mart 2022 tarihleri arasında belirli bir satıcı tarafından tedarik edilen kaç ürün satıldı?"

# 7-6a Koşullu Kısıtlamalarla Satır Seçme

Çıktıya dahil edilecek satırlara kısıtlamalar koyarak kısmi tablo içeriklerini seçebilirsiniz. Sorgu tarafından döndürülen satırları sınırlayan SELECT deyimine koşullu kısıtlamalar eklemek için **WHERE** cümlesini kullanın. Aşağıdaki sözdizimi, hangi satırların seçileceğini belirtmenizi sağlar:

SEÇİNİZ *sütun listesi*

FROM *tablelist*

[NEREDE *koşul listesi*]

[ORDER BY *sütun listesi* [ASC | DESC]];

SELECT deyimi, WHERE belirttiğiniz koşul(lar)la (koşullu ölçütler olarak da bilinir) eşleşen tüm satırları alır. SELECT deyiminin WHERE cümlesindeki *koşul listesi*, mantıksal operatörlerle ayrılmış bir veya daha fazla koşullu ifade ile temsil edilir. WHERE cümlesi isteğe bağlıdır. Hiçbir satır WHERE cümlesinde belirtilen ölçütlerle eşleşmezse, boş bir ekran veya hiçbir satırın alınmadığını belirten bir ileti görürsünüz. Örneğin, aşağıdaki sorguyu düşünün:

SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_PRICE, V\_CODE FROM ÜRÜN

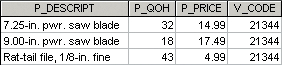
NEREDE V\_CODE 5 21344;

Bu sorgu, Şekil 7.12'de gösterildiği gibi, satıcı kodu 21344 olan ürünler için açıklama,

eldeki miktar, fiyat ve satıcı kodunu döndürür.

### NEREDE

Sorgu tarafından döndürülen satırları sınırlamak için bir SELECT deyimine koşullu kısıtlamalar ekleyen bir SQL cümlesi.



MS Access kullanıyorsanız, bu bölüm boyunca kodu oluşturmak için Access QBE (örnek sorgu) sorgu oluşturucusunu kullanabilirsiniz. Ancak, kod daha karmaşık hale geldikçe, Access tarafından oluşturulan kodun bölümde sunulandan giderek daha farklı hale geldiğini fark edebilirsiniz. Bunun nedeni, Access QBE'nin SQL'in kendi "yerel" sürümünü oluşturmasıdır. Şekil

7.13'ün alt kısmında gösterildiği gibi, Access SQL penceresinde standart SQL yazmayı da seçebilirsiniz. Şekilde Access QBE ekranı, SQL penceresinin QBE tarafından oluşturulan SQL'i ve değiştirilen SQL'in listesi gösterilmektedir.

Seçilen tablo içeriğine çok sayıda koşullu kısıtlama yerleştirilebilir. Örneğin, Tablo 7.5'te gösterilen karşılaştırma operatörleri çıktıyı kısıtlamak için kullanılabilir. *Not equal to için* iki seçenek olduğuna dikkat edin. Hem ,. hem de !5 iyi desteklenir ve aynı işlevi yerine getirir.

Aşağıdaki örnekte *not equal to* operatörlerinden biri kullanılmaktadır:

SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_PRICE, V\_CODE FROM ÜRÜN

NEREDE V\_CODE ,. 21344;

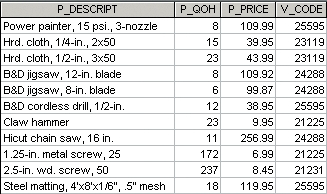
Şekil 7.14'te gösterilen çıktı, satıcı kodunun *olmadığı* tüm satırları listeler 21344.

Şekil 7.14'te, V\_CODE sütununda boş olan satırların (bkz. Şekil 7.2) SELECT komutunun

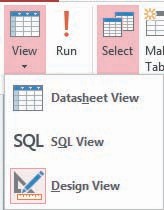
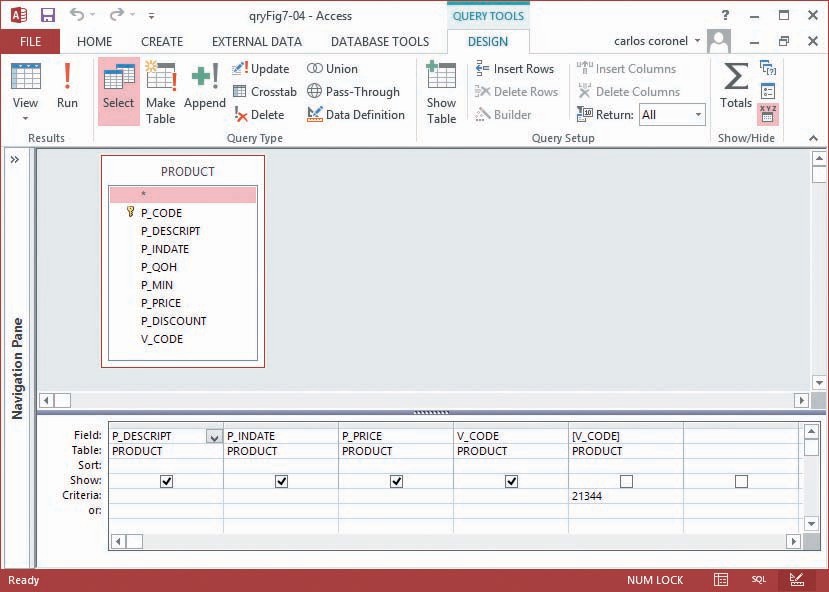
çıktısına dahil edilmediğine dikkat edin.

**Tablo 7.5 Karşı**

**laştırma Operatörleri**



**Şekil 7.14 21344 Dışındaki SATICI Kodları için ÜRÜN Öznitelikleri**



**Şekil 7.13 Microsoft Access QBE ve SQL**

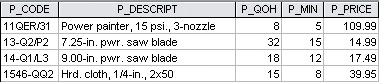
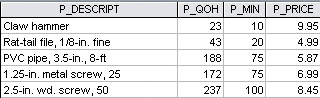
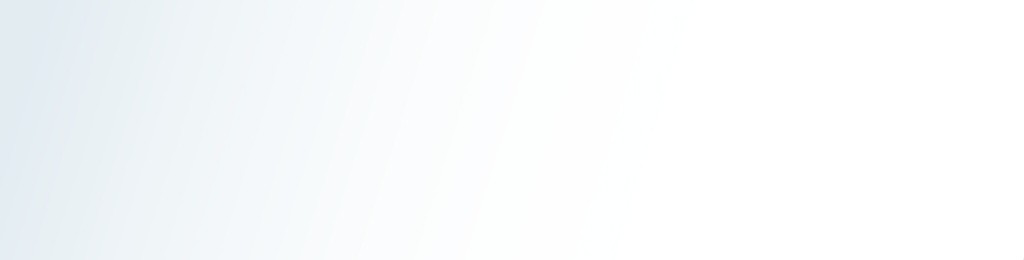
Sorgu görünümü seçenekleri

Microsoft Access tarafından oluşturulan SQL

Kullanıcı tarafından girilen SQL

|  |  |
| --- | --- |
| **Sembol** | **Anlamı** |
| 5 | Eşittir |
| , | Daha az |
| ,5 | Daha az veya eşit |
| > | Daha büyük |
| .5 | Daha büyük veya eşit |
| veya !5 | Eşit değil |

Aşağıdaki komut dizisi:



SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_MIN, P\_PRICE FROM ÜRÜN

NEREDE P\_PRICE ,5 10;

Şekil 7.15'te gösterilen çıktıyı verir.

**Şekil 7.15 P\_PRICE ile PRODUCT Tablo Özniteliklerini Seçin**

# 7-6b Karakter Özniteliklerinde Karşılaştırma Operatörlerini Kullanma

Bilgisayarlar tüm karakterleri sayısal American Standard Code for Infor- mation Interchange (ASCII) kodlarıyla tanımladığından, karşılaştırma operatörleri karakter tabanlı niteliklere kısıtlamalar koymak için bile kullanılabilir. Bu nedenle, komut:

SEÇİNİZ P\_CODE, P\_DESCRIPT, P\_QOH, P\_MIN, P\_PRICE FROM ÜRÜN

NEREDE P\_CODE , '1558-QW1';

doğru olacaktır ve P\_CODE'un alfabetik olarak 1558-QW1'den küçük olduğu tüm satırların bir listesini verecektir. (B harfinin ASCII kod değeri A harfinin değerinden büyük olduğu için, A'nın B'den küçük olduğu sonucu çıkar). Bu nedenle, çıktı Şekil 7.16'da gösterildiği gibi oluşturulacaktır.

**Şekil 7.16 ASCII Kodu Etkisi**

Dize (karakter) karşılaştırmaları soldan sağa doğru yapılır. Bu soldan sağa karşılaştırma özellikle isimler gibi nitelikleri karşılaştırırken kullanışlıdır. Örneğin, "Ardmore" dizesi "Aarenson" dizesinden *büyük,* ancak "Brown" dizesinden *küçük olarak* değerlendirilir; bu tür sonuçlar bir telefon rehberindeki gibi alfabetik listeler oluşturmak için kullanılabilir. Eğer 0-9 arası karakterler diziler olarak saklanırsa, aynı soldan sağa dizgi karşılaştırmaları belirgin anormalliklere yol açabilir. Örneğin, "5" karakterinin ASCII kodu, beklendiği gibi "4" karakterinin ASCII kodundan *büyüktür*. Yine de aynı "5", "44" dizesinden *daha büyük* olarak değerlendirilecektir çünkü "44" dizesindeki *ilk* karakter "5" dizesinden daha küçüktür.

Soldan sağa dize karşılaştırmaları nedeniyle, tarihler veya diğer sayılar karakter biçiminde saklandığında karşılaştırmalardan bazı beklenmedik sonuçlar alabilirsiniz. Örneğin, soldan sağa

ASCII karakter karşılaştırması, "01/01/2022" tarihinin 12/31/2021" tarihinden *önce* gerçekleştiği sonucuna varmaya zorlayacaktır. "01/01/2022" tarihindeki en soldaki 0" karakteri "12/31/2021tarihindeki en soldaki "1" karakterinden *küçük* olduğu için, "01/01/2022" tarihi 12/31/2021" tarihinden *küçüktür*. Doğal olarak, tarih dizeleri yyyy-mm-dd biçiminde saklanırsa, karşılaştırmalar uygun sonuçlar verecektir, ancak bu işletmelerde yaygın olmayan bir tarih sunumudur. Bu nedenle, mevcut tüm RDBMS'ler tarih veri türlerini destekler; bunları kullanmalısınız. Ayrıca, tarih veri türlerini kullanmak size tarih aritmetiği sağlar.

# 7-6c Tarihlerde Karşılaştırma Operatörlerini Kullanma

Tarih yordamları genellikle diğer SQL daha fazla yazılıma özgüdür. Örneğin, envanter stok tarihlerinin 20 Ocak 2022 veya sonrasında gerçekleştiği tüm satırları listelemek için kullanılan sorgu aşağıdaki gibidir:

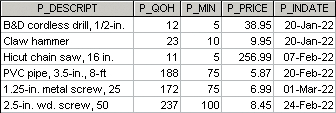
SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_MIN, P\_PRICE, P\_INDATE FROM ÜRÜN

NEREDE P\_INDATE .5 '2022-01-20';

MS Access kullanıcılarının tarihler için # sınırlayıcılarını kullanması gerektiğini unutmayın. Örneğin, önceki WHERE cümlesinde #20-Jan-22# kullanırsınız. Tarih kısıtlamalı çıktı Şekil 7.17'de gösterilmektedir. Oracle'da beklenen tarih biçimi gg-ay-yyyy'dir, bu nedenle WHERE cümlesi şu şekilde yazılır:

NEREDE P\_INDATE .5 '20-Jan-2022'

**Şekil 7.17 Seçilen ÜRÜN Tablo Öznitelikleri: Tarih Kısıtlaması**



### VEYA

Bir WHERE veya HAVING birden fazla koşullu ifadeyi bağlamak için kullanılan SQL mantıksal operatörü. Koşullu ifadelerden yalnızca birinin doğru olmasını gerektirir.

# 7-6d Mantıksal Operatörler: AND, OR ve NOT

Gerçek dünyada, bir veri araması normalde birden fazla koşul içerir. Örneğin, yeni bir ev satın alırken, belirli bir alan, belirli sayıda yatak odası, banyo odası, kat . ararsınız. Aynı şekilde SQL, mantıksal operatörleri kullanarak bir sorguya birden fazla koşul olanak tanır. Mantıksal operatörler AND, OR ve NOT'tur. Örneğin, V\_CODE 5 21344 *veya* V\_CODE 5 24288 için tablo içeriklerinin bir listesini istiyorsanız, aşağıdaki komut dizisinde olduğu gibi **OR** mantıksal operatörünü kullanabilirsiniz:

SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_PRICE, V\_CODE FROM ÜRÜN

NEREDE V\_CODE 5 21344 VEYA V\_CODE 5 24288;

Bu komut, Şekil 7.18'de gösterilen ve mantıksal kısıtlamayla eşleşen altı satırı oluşturur.

**Şekil 7.18 Mantıksal VEYA**

**AND** mantıksal operatörü OR ile aynı SQL sözdizimi gereksinimine sahiptir. Aşağıdaki komut, P\_PRICE değeri $100'dan büyük ve P\_QOH değeri 20'den küçük olan tüm satırların bir listesini oluşturur:

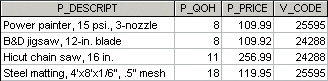
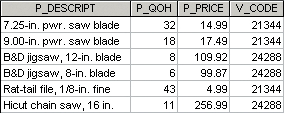
SEÇİNİZ P\_DESCRIPT, P\_QOH, P\_PRICE, V\_CODE FROM ÜRÜN

NEREDE P\_PRICE . 100 VE P\_QOH, 20;

Bu komut Şekil 7.19'da gösterilen çıktıyı üretir.

**VE**

Bir WHERE veya HAVING cümlesinde birden fazla koşullu ifadeyi bağlamak için kullanılan SQL mantıksal operatörü. Tüm koşullu ifadelerin doğru olarak değerlendirilmesini gerektirir.



**Şekil 7.19 Mantıksal VE**

Çıktıya daha fazla kısıtlama koymak için mantıksal VEYA ile mantıksal VE'yi birleştirebilirsiniz. Örneğin, aşağıdaki koşullar için bir tablo listesi istediğinizi varsayalım:

* V\_CODE 25595 ya *da* 24288'dir.
* *Ve* P\_PRICE 100$'dan büyüktür.

Aşağıdaki kod hatalı sonuçlar üretmektedir. Şekil 7.20'de gösterildiği gibi, P\_PRICE değerlerinden bazıları gerekli olan 100 $'dan az olsa bile, 25595 no'lu ven- dor'daki tüm satırlar sonuca dahil edilir. Bunun nedeni, DBMS'nin VE operatörünü VEYA operatöründen önce çalıştırmasıdır.

SEÇİNİZ P\_DESCRIPT, P\_PRICE, V\_CODE FROM ÜRÜN

NEREDE V\_CODE 5 25595 VEYA V\_CODE 5 24288 VE P\_PRICE . 100;

WHERE cümlesindeki koşullar, istenen sonucu üretmek için parantezler kullanılarak gruplandırılabilir. Gerekli listeleme aşağıdakiler kullanılarak üretilebilir:

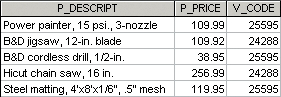
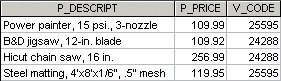
SEÇİNİZ P\_DESCRIPT, P\_PRICE, V\_CODE FROM ÜRÜN

NEREDE (V\_CODE 5 25595 VEYA V\_CODE 5 24288) VE P\_PRICE . 100;

**Şekil 7.20 AND ve OR'nin Yanlış Kombinasyonu**

Mantıksal kısıtlamaları birleştirmek için parantez kullanımına dikkat edin. Parantezleri nereye yerleştireceğiniz, mantıksal kısıtlamaların nasıl yürütülmesini istediğinize bağlıdır. Parantezler içinde listelenen koşullar her zaman önce yürütülür. Önceki sorgu Şekil 7.21'de gösterilen çıktıyı verir.

**Şekil 7.21 VE ve VEYA Koşullarının Doğru Kombinasyonu**



### Boole cebiri

OR, AND ve NOT mantıksal operatörlerini kullanan bir matematik dalı.

### DEĞİL

Verilen bir yüklemi olumsuzlayan bir SQL mantıksal işleci.

OR ve AND mantıksal operatörlerinin kullanımı, sorguya çok sayıda kısıtlama getirildiğinde oldukça karmaşık hale gelebilir. Aslında, matematikte **Boole cebiri** olarak bilinen bir uzmanlık alanı mantıksal operatörlerin kullanımına adanmıştır.

**NOT** mantıksal işleci, koşullu bir ifadenin sonucunu olumsuzlamak için kullanılır. Yani, SQL'de tüm koşullu ifadeler doğru veya yanlış olarak değerlendirilir. Bir ifade doğruysa, satır seçilir; bir ifade yanlışsa, satır seçilmez. NOT mantıksal operatörü genellikle belirli bir koşulla eşleşmeyen satırları bulmak için kullanılır. Örneğin, satıcı kodu 21344 *olmayan* tüm satırların bir listesini görmek istiyorsanız, aşağıdaki komut dizisini kullanın:

SEÇİNİZ \*

FROM ÜRÜN

NEREDE DEĞIL (V\_CODE 5 21344);

Koşulun parantez içine alındığına dikkat edin; bu uygulama isteğe bağlıdır, ancak anlaşılır olması için şiddetle tavsiye edilir. Mantıksal operatör NOT, AND ve OR ile birleştirilebilir.

# 7-6e Özel Operatörler

ANSI standardı SQL, WHERE cümlesiyle birlikte özel operatörlerin kullanılmasına izin verir. Bu özel operatörler şunları içerir:

***BETWEEN***: Bir öznitelik değerinin bir aralık içinde olup olmadığını kontrol etmek için kullanılır

***IN***: Bir öznitelik değerinin bir değer listesindeki herhangi bir değerle eşleşip eşleşmediğini kontrol etmek için kullanılır

***LIKE***: Bir öznitelik değerinin belirli bir dize kalıbıyla eşleşip eşleşmediğini kontrol etmek için kullanılır

***IS NULL***: Bir öznitelik değerinin null olup olmadığını kontrol etmek için kullanılır

## **BETWEEN Özel Operatörü**

Standart SQL uygulayan bir yazılım kullanıyorsanız, bir öznitelik değerinin bir değer aralığı

içinde olup olmadığını kontrol etmek için **BETWEEN** operatörü kullanılabilir. Örneğin, fiyatları

$50 ile $100 *arasında* olan tüm ürünleri listelemek istiyorsanız, aşağıdaki komut dizisini

kullanın:

SEÇİNİZ \*

FROM ÜRÜN

NEREDE P\_PRICE 50,00 ILE 100,00 ARASINDA;

### ARASINDA

SQL'de, bir değerin belirtilen değerler aralığında olup olmadığını kontrol etmek için kullanılan özel bir karşılaştırma operatörü.

**Not**

BETWEEN özel operatörünü kullanırken, her zaman önce alt aralık değerini belirtin. Yukarıdaki komutun WHERE cümlesi şu şekilde yorumlanır:

BURADA P\_PRICE ,5 50 VE P\_PRICE ,5 100

Önce yüksek aralıklı değeri listelerseniz, WHERE cümlesi şu şekilde yorumlanacağı için DBMS boş bir sonuç kümesi döndürür:

BURADA P\_PRICE .5 100 VE P\_PRICE ,5 50

Açıkçası, hiçbir ürünün hem 100'den büyük de 50'den küçük bir fiyatı olamaz. Bu nedenle, hiçbir satırın kriterlere uyması mümkün değildir.

DBMS'niz BETWEEN'i desteklemiyorsa, kullanabilirsiniz: SEÇİNİZ \*

FROM ÜRÜN

NEREDE P\_PRICE 5. 50.00 VE P\_PRICE ,5 100.00;

## **IN Özel Operatörü**

Mantıksal OR kullanımını gerektiren birçok sorgu, özel **IN** operatörünün yardımıyla daha kolay bir

şekilde ele alınabilir. Örneğin, aşağıdaki sorgu:

SEÇİNİZ \*

FROM ÜRÜN

NEREDE V\_CODE 5 21344 VEYA V\_CODE 5 24288;

ile daha verimli bir şekilde ele alınabilir:

SEÇİNİZ \*

FROM ÜRÜN

NEREDE V\_CODE (21344, 24288) IÇINDE;

IN işlecinin bir değer listesi kullandığına dikkat edin. Listedeki tüm değerler aynı veri türünde olmalıdır. Değer listesindeki her değer öznitelikle (bu durumda V\_CODE) karşılaştırılır. V\_CODE değeri listedeki değerlerden herhangi biriyle eşleşirse, satır seçilir. Bu örnekte, seçilen satırlar yalnızca V\_CODE değerinin 21344 veya 24288 olduğu satırlar olacaktır.

Kullanılan öznitelik bir karakter veri tipindeyse, liste değerleri tek tırnak işareti içine alınmalıdır. Örneğin, tablo oluşturulduğunda V\_CODE karakter verisi olarak tanımlanmış olsaydı, önceki sorgu şöyle olurdu:

SEÇİNİZ \*

FROM ÜRÜN

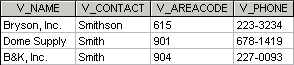
NEREDE V\_CODE ('21344', '24288') IÇINDE;

### İÇİNDE

SQL'de, bir değerin belirtilen değerler listesinde olup olmadığını kontrol etmek için kullanılan bir karşılaştırma operatörü.

IN operatörü, daha sonraki bir bölümde ele alınacak olan alt sorgularla birlikte kullanıldığında

özellikle değerlidir.



### LIKE

SQL'de, bir özniteliğin metin değerinin belirtilen bir dize deseniyle eşleşip eşleşmediğini kontrol etmek için kullanılan bir karşılaştırma operatörü.

## **LIKE Özel Operatör**

**LIKE** özel operatörü, dize nitelikleri içindeki kalıpları bulmak için joker karakterlerle birlikte kullanılır. Standart SQL, dizenin tamamı bilinmediğinde eşleştirme yapmak için yüzde işareti (%) ve alt çizgi (\_) joker karakterlerini kullanmanıza izin verir:

* *aşağıdaki* veya *önceki* tüm karakterlerin uygun olduğu anlamına gelir. Örneğin:

Ď 'J%' Johnson, Jones, Jernigan, July ve J-231Q'yu içerir.

Ď 'Jo%' Johnson ve Jones'u içerir.

Ď '%n' Johnson ve Jernigan'ı içerir.

* \_ alt çizgi yerine herhangi *bir* karakterin kullanılabileceği anlamına gelir. Örneğin:

Ď '\_23-456-6789' 123-456-6789, 223-456-6789 ve 323-456-6789'u içerir.

Ď '\_23-\_56-678\_' 123-156-6781, 123-256-6782 ve 823-956-6788'i içerir.

Ď '\_o\_es' Jones, Cones, Cokes, totes ve rolleri içerir.

**Not**

MS Access gibi bazı RDBMS'ler % ve \_ yerine \* ve ? joker karakterlerini kullanır.

Örneğin, aşağıdaki sorgu *Smith* ile başlayan kişiler için tüm VENDOR satırlarını bulacaktır.

SEÇİNİZ V\_NAME, V\_CONTACT, V\_AREACODE, V\_PHONE FROM SATICI

NEREDE V\_CONTACT LIKE 'Smith%';

Şekil 7.22, sonuçların "Smith" ve "Smithson" adlı kişileri içerdiğini göstermektedir.

Çoğu SQL uygulamasının büyük/küçük harfe duyarlı arama yaptığını unutmayın. Örneğin, soyadları için bir aramada *joker* arama sınırlayıcısı 'jo%' kullanırsanız Oracle *Jones*'u içeren bir sonuç vermez; *Jones* büyük *J* ile başlar ve *joker* aramanız küçük *j* ile başlar. Öte yandan, MS Access aramaları büyük/küçük harfe duyarlı değildir.

Örneğin, Oracle'da aşağıdaki sorguyu yazdığınızı varsayalım:

SEÇİNİZ V\_NAME, V\_CONTACT, V\_AREACODE, V\_PHONE FROM SATICI

NEREDE V\_CONTACT 'SMITH%' GIBI;

Karakter tabanlı sorgular büyük/küçük harfe duyarlı olabileceğinden hiçbir satır döndürülmeyecektir. , büyük harfli bir karakterin küçük harfli bir karakterden farklı bir ASCII kodu vardır, bu da *SMITH*,

*Smith* ve *smith* farklı (eşit olmayan) girişler olarak değerlendirilmelidir. Tabloda soyadı *SMITH* (tümü büyük harf) ile başlayan hiçbir satıcı bulunmadığından, sorguda kullanılan 'SMITH%' eşleştirilemez. Eşleştirmeler yalnızca sorgu girişi tam olarak tablo girişi gibi yazıldığında yapılabilir.

Microsoft SQL Server gibi bazı RDBMS'ler büyük/küçük harf duyarlılığını ortadan kaldırmak için gerekli dönüşümleri otomatik olarak yapar. Oracle gibi diğerleri, hem tablo hem de sorgu karakter girişlerini büyük harfe dönüştürmek için özel bir UPPER işlevi sağlar. (Dönüşüm yalnızca bilgisayarın belleğinde yapılır; dönüşümün değerin tabloda nasıl saklandığı üzerinde hiçbir etkisi yoktur). UPPER ve diğer birçok SQL fonksiyonu hakkında daha fazla bilgiyi bu bölümün ilerleyen kısımlarında öğreneceksiniz. Dolayısıyla, büyük/küçük harf duyarlılığına dayalı bir eşleşme yok sonucundan kaçınmak istiyorsanız ve RDBMS'niz UPPER fonksiyonunun kullanımına izin veriyorsa, aşağıdaki sorguyu kullanarak aynı sonuçları üretebilirsiniz:

SEÇİNİZ V\_NAME, V\_CONTACT, V\_AREACODE, V\_PHONE FROM SATICI

NEREDE UPPER(V\_CONTACT) LIKE 'SMITH%';

Yukarıdaki sorgu, *Smith*, *smith* ve *SMITH* gibi büyük veya küçük harf kombinasyonlarına bakılmaksızın, *Smith* ile başlayan bir soyadı içeren tüm satırları içeren bir liste üretir.

Mantıksal operatörler özel operatörlerle birlikte kullanılabilir. Örneğin, aşağıdaki sorgu:

SEÇİNİZ V\_NAME, V\_CONTACT, V\_AREACODE, V\_PHONE FROM SATICI

NEREDE V\_CONTACT 'Smith%' GİBİ DEĞİL;

adı *Smith* ile başlamayan tüm satıcıların bir çıktısını verecektir.

Bir kişinin adının *Johnson* mı yoksa *Johnsen* mi yazıldığını bilmediğinizi varsayalım. Joker karakter \_, her iki yazım için de bir eşleşme bulmanızı sağlar. Doğru arama aşağıdaki sorgu ile başlatılır:

SEÇİNİZ \*

FROM SATICI

NEREDE V\_CONTACT LIKE 'Johns\_n';

Böylece, joker karakterler yalnızca yaklaşık yazılışları bilindiğinde eşleştirme yapmanıza olanak tanır. Joker karakterler kombinasyonlar halinde kullanılabilir. Örneğin, '\_l%' dizesine dayalı joker karakter araması, hepsi ikinci karakter olarak "l" harfine sahip olan *Al*, *Alton*, *Elgin*, *Blakeston*, *blank*, *bloated* ve *eligible* dizelerini verebilir.

## **IS NULL Özel Operatörü**

Standart SQL, boş bir öznitelik değerini kontrol etmek için **IS NULL** kullanımına izin verir. Örneğin, atanmış bir satıcısı olmayan tüm ürünleri listelemek istediğinizi varsayalım (yani, V\_CODE özniteliği bir değer içermiyor). Böyle bir null giriş, Şekil 7.23'te gösterildiği gibi aşağıdaki komut dizisi kullanılarak bulunabilir.

SEÇİNİZ P\_CODE, P\_DESCRIPT, V\_CODE FROM ÜRÜN

NEREDE V\_CODE BOŞTUR;

SQL'in null'ları test etmek için özel bir operatör kullandığını unutmayın. Neden? Sadece "V\_CODE 5 NULL" gibi bir koşul giremez miydiniz? Teknik olarak NULL, 0 sayısı veya boşluk gibi bir "değer" değildir; bunun yerine NULL, herhangi bir değerin yokluğunu temsil eden bir niteliğin özel bir özelliğidir. Mantıksal karşılaştırmalar için NULL, Bilinmeyen olarak düşünülebilir. Bir WHERE cümlesinde V\_CODE 5 NULL kullanılırsa, V\_CODE için 21225 değeri değerlendirildiğinde, DBMS "21225 Bilinmiyor'a eşit mi?" diye sorar. Cevap

### IS NULL

SQL'de, bir niteliğin bir değere sahip olup olmadığını kontrol etmek için kullanılan bir karşılaştırma operatörü.

**Şekil 7.23 Bir Satıcıyla İlişkili Olmayan Ürünler**



bilinmemektedir çünkü DBMS Bilinmeyen değerinin neyi temsil etmesi gerektiğini bilmemektedir. V\_CODE'da bir NULL bu koşul için değerlendirildiğinde, DBMS "Bilinmeyen Bilinmeyene eşit mi?" diye sorar. Cevap bilinmiyordur çünkü DBMS ilk Bilinmeyen değerinin hangi değeri temsil ettiğini veya ikinci değerinin hangi değeri temsil ettiğini bilmemektedir, bu nedenle aynı veya farklı değerleri temsil edip etmediklerini söyleyemez. WHERE cümlesinin sonuç sorgusuna bir satır dahil etmesi için kriterlerin True olarak değerlendirilmesi gerekir, bu nedenle False veya Unknown sonuçları dahil edilmez. Bu nedenle, WHERE V\_CODE 5 NULL hiçbir zaman herhangi bir satır döndürmez, çünkü V\_CODE için bir değer içerip içermediğine bakılmaksızın her satır Unknown olarak değerlendirilir. Bu nedenle, bunun yerine IS NULL işleci kullanılır.

## **NOT ile Özel Operatörler**

Daha önce tartışıldığı gibi, NOT mantıksal bağlayıcısı bir koşulu olumsuzlamak için kullanılabilir. NOT ayrıca özel operatörlerle birlikte de kullanılabilir. BETWEEN belirli bir değer aralığındaki değerlere sahip satırları döndürürken, NOT BETWEEN verilen aralığın dışında kalan değerlere sahip satırları döndürür. IN, verilen bir liste içindeki herhangi bir değerle eşleşen değerlere sahip satırları döndürür. NOT IN, verilen listedeki herhangi bir değerle eşleşmeyen değerlere sahip satırları döndürür. NOT IN ilk başta göründüğünden biraz daha zor olabilir. NOT IN'in bir satır döndürmesi için, listeyle karşılaştırılan değerin listedeki her değerle karşılaştırıldığında False olarak değerlendirilmesi gerekir. NULL'un bir değerin olmaması anlamına geldiğini unutmayın. Bir değer mantıksal olarak NULL ile karşılaştırıldığında, True veya False olarak değerlendirilmez, Unknown olarak değerlendirilir. Bu nedenle, NOT IN ile kullanılan değerler listesi bir null içeriyorsa, işleç herhangi bir satır döndürmez. LIKE, daha büyük bir metin dizesi içinde daha küçük bir metin dizesini bulmak için bir alt dize araması için kullanılır. NOT LIKE, daha küçük metin dizesini içermeyen satırları döndürür. IS NULL, belirtilen öznitelikte değeri olmayan satırları döndürür. NOT sözcüğünü işlecin önüne yerleştiren diğer özel işleçlerden farklı olarak IS NULL, IS NOT NULL işlecini üretmek için NOT sözcüğünü ortaya yerleştirir. IS NOT NULL, belirtilen öznitelikte herhangi bir değer içeren satırları, bu değerin ne bakmaksızın döndürür.

**7-7 JOIN İşlemleri**

Daha önce belirtildiği gibi, FROM cümlesinde tek bir tablo kullanılarak birçok sorgu yazılabilir. Ancak pratikte, daha karmaşık SELECT sorgularının birden fazla tablodan veri alması gerekecektir. Bölüm 3'te, birden fazla tablodan gelen verileri anlamlı bir şekilde birleştirmek için kullanılan JOIN operatörlerini incelediniz. Mevcut veritabanına yol açan veritabanı tasarım süreci birçok yönden bir ayrıştırma süreciydi - tasarımcı bir iş sorunuyla bütünleşik bir veri kümesini aldı ve bu verileri depolamak ve işlemek için esnek, istikrarlı bir yapı oluşturmak üzere bu verileri ayrı varlıklara ayrıştırdı. Şimdi ise programcı, birleştirmeleri kullanarak kullanıcıların bilgi ihtiyaçlarını karşılamak için veri parçalarını yeniden bütünleştirir. *İç birleştirmeler,* tablolardan yalnızca ortak bir değerle eşleşen satırları döndürür. *Dış birleştirmeler*, iç birleştirmeyle aynı eşleşen satırları ve ayrıca bir tablodan veya diğerinden eşleşmeyen satırları döndürür. (Çeşitli birleştirme türleri Bölüm 3'te sunulmuştur).

Birleştirme koşulu genellikle yabancı anahtar ile ilgili tabloların birincil anahtarı arasındaki eşitlik karşılaştırmasından oluşur. Örneğin, VENDOR ve PRODUCT tablolarını birleştirmek istediğinizi varsayalım. V\_CODE, PRODUCT tablosunda yabancı anahtar ve VENDOR tablosunda birincil anahtar olduğundan, bağlantı V\_CODE üzerinde kurulur. (Bkz. Tablo 7.6.)