

**Not**

Bir sütun adının beklendiği her yerde bir ifade kullanabilirsiniz. Hangi ürünün en yüksek envanter değerine sahip olduğunu bilmek istediğinizi varsayalım. Cevabı bulmak için aşağıdaki sorguyu yazabilirsiniz:

SEÇİNİ

Z FROM NERED

\*

ÜRÜN

P\_QOH \* P\_PRICE 5 (SELECT MAX(P\_QOH \* P\_PRICE) FROM PRODUCT);

EN

Alt sorgular birleştirmelerle birlikte de kullanılabilir. Örneğin, aşağıdaki sorgu pençe çekiç

siparişi veren tüm müşterileri listeler:

SEÇİNİZ DISTINCT CUS\_CODE, CUS\_LNAME, CUS\_FNAME FROM CUSTOMER JOIN INVOICE USING (CUS\_CODE)

JOIN LINE USING (INV\_NUMBER) JOIN PRODUCT USING (P\_CODE)

NEREDE P\_CODE 5 (SELECT P\_CODE FROM PRODUCT WHERE

P\_ DESCRIPT 5 'Claw hammer'); Sorgunun sonucu Şekil 7.50'de gösterilmektedir.

**Şekil 7.50 Pençe Çekiç Siparişi Veren Müşteriler**

Yukarıdaki örnekte, iç sorgu pençe çekiç için P\_CODE'u bulur. P\_CODE daha sonra seçilen satırları LINE tablosundaki P\_CODE'un "Pençe çekici" için P\_CODE ile eşleştiği satırlarla sınırlamak için kullanılır. Önceki sorgunun bu şekilde yazılabileceğini unutmayın:

SEÇİNİZ DISTINCT CUSTOMER.CUS\_CODE, CUS\_LNAME, CUS\_FNAME FROM CUSTOMER JOIN INVOICE ON CUSTOMER.CUS\_CODE 5

INVOICE.CUS\_CODE

JOIN LINE ON INVOICE.INV\_NUMBER 5 .INV\_NUMBER JOIN PRODUCT ON PRODUCT.P\_CODE 5 LINE.P\_CODE

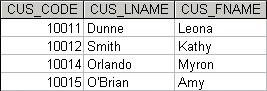
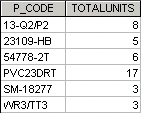
NEREDE P\_DESCRIPT 5 'Pençe çekici';

Orijinal sorgu birden fazla ürün tanımında "Claw hammer" dizesiyle karşılaşırsa bir hata mesajı alırsınız. Bir değeri bir değerler listesiyle karşılaştırmak için, bir sonraki bölümde gösterildiği gibi bir IN işleneni kullanmanız gerekir.

## b IN Alt Sorgular

Bir çekiç veya herhangi bir testere ya da testere bıçağı satın alan tüm müşterileri bulmak isteseydiniz ne olurdu? Ürün tablosunda iki tür çekiç vardır: pençe çekiç ve balyoz. Ayrıca, testere bıçakları ve dekupaj testereleri de dahil olmak üzere, ürün açıklamalarında birden fazla "testere" içeren ürün bulunmaktadır. Bu gibi durumlarda, P\_CODE'u tek bir ürün koduyla (tek bir değer) değil, ürün kodu değerlerinin bir listesiyle karşılaştırmanız gerekir. Tek bir özniteliği bir değerler listesiyle karşılaştırmak istediğinizde IN operatörünü kullanırsınız. P\_CODE değerleri bilinmediğinde

ancak bir sorgu kullanılarak türetilebiliyorsa, bir IN alt sorgusu kullanmanız gerekir. Aşağıdaki



örnekte çekiç, testere veya testere bıçağı satın almış tüm müşteriler listelenmektedir.

SEÇİNİZ DISTINCT CUSTOMER.CUS\_CODE, CUS\_LNAME, CUS\_FNAME FROM CUSTOMER JOIN INVOICE ON CUSTOMER.CUS\_CODE 5

INVOICE.CUS\_CODE

JOIN LINE ON INVOICE.INV\_NUMBER 5 LINE.INV\_NUMBER JOIN PRODUCT ON LINE.P\_CODE 5 PRODUCT.P\_CODE

NEREDE P\_CODE IN (SELECT P\_CODE FROM PRODUCT

WHERE P\_DESCRIPT LIKE '%hammer%' OR P\_DESCRIPT LIKE '%saw%');

Sorgunun sonucu Şekil 7.51'de gösterilmektedir.

**Şekil 7.51 IN Alt Sorgu Örneği**

## 7-9c HAVING Alt Sorguları

WHERE cümlesiyle alt sorguları kullanabildiğiniz gibi, HAV- ING cümlesiyle de bir alt sorgu kullanabilirsiniz. HAVING cümlesi, gruplandırılmış satırlara koşullu ölçütler uygulayarak GROUP BY sorgusunun çıktısını kısıtlamak için kullanılır. Örneğin, satılan toplam miktarı satılan ortalama miktardan fazla olan tüm ürünleri listelemek için aşağıdaki sorguyu yazarsınız:

SEÇİNİZ P\_CODE, SUM(LINE\_UNITS) AS TOTALUNITS FROM HAT

GROUP BY P\_CODE

SAHİP OLMAK SUM(LINE\_UNITS) . (SELECT AVG(LINE\_UNITS) FROM LINE);

Sorgunun sonucu Şekil 7.52'de gösterilmektedir.

## d Çok Satırlı Alt Sorgu Operatörleri: ALL ve ANY

Şimdiye kadar, bir değeri bir değerler listesiyle karşılaştırmak için bir IN alt sorgusu kullanmanız gerektiğini öğrendiniz. Ancak, IN alt sorgusu bir eşitlik operatörü kullanır; , yalnızca aşağıdaki satırları seçer

listedeki değerlerden en az birine eşittir. Bir değerin bir değerler listesiyle eşitsizlik karşılaştırması (. veya ,) yapmanız gerekirse ne olur?



Örneğin, hangi ürünlerin Florida'daki satıcılar tarafından sağlanan tüm bireysel ürünlerden daha pahalı olduğunu bilmek istediğinizi varsayalım:

SEÇİNİZ P\_CODE, P\_QOH \* P\_PRICE AS TOTALVALUE FROM ÜRÜN

NEREDE P\_QOH \* P\_PRICE . ALL (SELECT P\_QOH \* P\_PRICE

ÜRÜNDEN

WHERE V\_CODE IN (SELECT V\_CODE

FROM VENDOR WHERE V\_STATE 5 'FL'));

Sorgunun sonucu Şekil 7.53'te gösterilmektedir.

**Şekil 7.53 Çok Satırlı Alt Sorgu Örneği**

Şekil 7.53'teki sorgu ve çıktısı hakkında aşağıdaki önemli noktalara dikkat edin:

* Bu sorgu tipik bir iç içe sorgu örneğidir.
* Sorgu, ikinci bir SELECT alt sorgusu (sqB olarak adlandırın) içeren bir SELECT alt sorgusu (sqA olarak adlandırın) ile bir dış SELECT deyimine sahiptir.
* Son SELECT alt sorgusu (sqB) ilk olarak çalıştırılır ve Florida'daki tüm satıcıların bir listesini döndürür.
* İlk SELECT alt sorgusu (sqA), ikinci SELECT alt sorgusunun (sqB) çıktısını kullanır. sqA alt sorgusu, Florida'daki satıcılar tarafından sağlanan tüm ürünlerin maliyetlerinin listesini döndürür.
* ALL işlecinin kullanılması, eşittir dışında bir karşılaştırma işleci kullanarak tek bir değeri (P\_QOH \* P\_PRICE) ilk alt sorgu (sqA) tarafından döndürülen değerler listesiyle karşılaştırmanıza olanak tanır.
* Bir satırın sonuç kümesinde görünmesi için P\_QOH \* P\_PRICE kriterini karşılaması gerekir. Alt sorgu sqA tarafından döndürülen tek tek değerlerin TÜMÜ. sqA tarafından döndürülen değerler, ürün maliyetlerinin bir listesidir. Aslında, "'nden büyük" ifadesi "listedeki en yüksek ürün maliyetinden büyük" ifadesine eşdeğerdir. Aynı şekilde, "TÜMÜ'nden küçük" koşulu da "listedeki en düşük ürün maliyetinden küçük" koşuluna .

Bir diğer güçlü işleç, ALL çoklu satır işlecinin kuzeni olarak düşünebileceğiniz ANY çoklu satır işlecidir. ANY işleci, tek bir değeri bir değerler listesiyle karşılaştırmanıza ve yalnızca envanter maliyetinin listedeki herhangi bir değerden büyük veya küçük olduğu satırları seçmenize olanak tanır. IN operatörüne eşdeğer olan ANY operatörüne eşittir operatörünü de kullanabilirsiniz.

## e FROM Alt Sorgular

Şimdiye kadar SELECT ifadesinin WHERE, HAVING ve IN ifadeleri içinde alt sorguları nasıl kullandığını ve çok satırlı alt sorgular için ANY ve ALL operatörlerinin nasıl kullanıldığını gördünüz. Tüm bu durumlarda, alt sorgu koşullu bir ifadenin parçasıydı ve her zaman ifadenin sağ tarafında yer alıyordu. Bu bölümde, FROM cümlesinde alt sorguların nasıl kullanılacağını öğreneceksiniz.

Bildiğiniz gibi, FROM cümlesi verilerin tablo(lar)ı belirtir. SELECT deyiminin çıktısı başka bir tablo (ya da daha doğrusu "sanal" bir tablo) olduğundan, FROM cümlesinde bir SELECT alt sorgusu kullanabilirsiniz. Örneğin, 13-Q2/P2 *ve* 23109-HB ürünlerini satın alan tüm müşterileri bilmek istediğinizi varsayın. Tüm ürün satın alımları LINE tablosunda saklanır, dolayısıyla LINE tablosunda P\_CODE özniteliğini arayarak herhangi bir ürünü kimin satın aldığını kolayca bulabilirsiniz. Ancak bu durumda, yalnızca bir ürünü değil, her iki ürünü de satın alan tüm müşterileri bilmek istiyorsunuz. Aşağıdaki sorguyu yazabilirsiniz:



SEÇİNİZ DISTINCT CUSTOMER.CUS\_CODE, CUSTOMER.CUS\_LNAME FROM MÜŞTERİ KATILIMI

(SELECT INVOICE.CUS\_CODE FROM INVOICE JOIN LINE ON INVOICE.INV\_NUMBER 5 .INV\_NUMBER WHERE P\_CODE 5 '13-Q2/P2') CP1

ON CUSTOMER.CUST\_CODE 5 CP1.CUS\_CODE JOIN

(SELECT INVOICE.CUS\_CODE FROM INVOICE JOIN LINE ON INVOICE.INV\_NUMBER 5 .INV\_NUMBER WHERE P\_CODE 5 '23109-HB') CP2

CP1.CUS\_CODE 5 CP2.CUS\_CODE ÜZERINDE;

Sorgunun sonucu Şekil 7.54'te gösterilmektedir.

Şekil 7.54'te, ilk alt sorgunun 13-Q2/P2 ürününü satın alan tüm müşterileri döndürdüğüne, ikinci alt sorgunun ise 23109-HB ürününü satın alan tüm müşterileri döndürdüğüne dikkat edin. Dolayısıyla, bu FROM alt sorgusunda, CUSTOMER tablosunu iki sanal tablo ile birleştiriyorsunuz. Birleştirme koşulu, her tabloda (temel veya sanal) yalnızca eşleşen CUS\_CODE değerlerine sahip satırları seçer.

## f Öznitelik Listesi Alt Sorguları

SELECT deyimi, sonuç kümesinde hangi sütunların yansıtılacağını belirtmek için öznitelik listesini kullanır. Bu sütunlar temel tabloların , hesaplanan öznitelikler veya bir toplama işlevinin sonucu olabilir. Öznitelik listesi, *satır içi alt* sorgu olarak da bilinen bir alt sorgu ifadesi de *içerebilir*. Öznitelik listesindeki bir alt sorgu bir değer döndürmelidir; aksi takdirde bir hata kodu oluşturulur. Örneğin, her bir ürünün fiyatı ile ortalama ürün fiyatı arasındaki farkı listelemek için basit bir satır içi sorgu kullanılabilir:

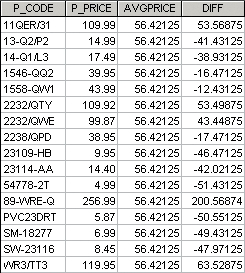
SEÇİNİZ P\_CODE, P\_PRICE, (SELECT AVG(P\_PRICE) FROM PRODUCT) AS AVGPRICE,

P\_PRICE - (SELECT AVG(P\_PRICE) FROM PRODUCT) AS DIFF FROM ÜRÜN;

Şekil 7.55 sorgunun sonucunu göstermektedir.

Şekil 7.55'te, satır içi sorgu çıktısının tek bir değer (ortalama ürün fiyatı) döndürdüğüne ve bu değerin her satırda aynı olduğuna dikkat edin. Ayrıca, sorgunun farkı hesaplarken sütun takma adları yerine tam ifadeyi kullandığına da dikkat edin. Aslında, fark ifadesinde diğer adı kullanmaya çalışırsanız bir hata mesajı alırsınız. Sütun takma adı şu şekilde olamaz

**Şekil 7.55 Satır İçi Alt Sorgu Örneği**



takma ad aynı öznitelik listesinde tanımlandığında öznitelik listesindeki hesaplamalarda kullanılır. Bu VTYS gereksinimi, VTYS'nin sorguları ayrıştırma ve yürütme şeklinin bir sonucudur.

Başka bir örnek, öznitelik listesi alt sorgularının ve sütun takma adlarının kullanımını anlamanıza yardımcı olacaktır. Örneğin, ürün kodunu, ürün bazında toplam satışları ve her bir ürünün satışlarının çalışan bazında katkısını bilmek istediğinizi varsayalım. Ürüne göre satışları almak için yalnızca LINE tablosunu kullanmanız gerekir. Çalışana göre katkıyı hesaplamak için, çalışan sayısını (EMPLOYEE tablosundan) bilmeniz gerekir. Tabloların yapılarını incelediğinizde, LINE ve EMPLOYEE tablolarının ortak bir niteliği paylaşmadığını görebilirsiniz. Aslında, ortak bir niteliğe ihtiyacınız yoktur. Yalnızca toplam çalışan sayısını bilmeniz gerekir, her bir ürünle ilgili toplam çalışanı değil. Dolayısıyla, sorguyu yanıtlamak için aşağıdaki kodu yazarsınız:

SEÇİNİZ P\_CODE, SUM(LINE\_UNITS \* LINE\_PRICE) AS SALES, (SELECT COUNT(\*) FROM EMPLOYEE) AS ECOUNT, SUM(LINE\_UNITS \* LINE\_PRICE)/(SELECT COUNT(\*) FROM EMPLOYEE) AS CONTRIB

FROM SATIR GRUPLAMA P\_CODE;

Sorgunun sonucu Şekil 7.56'da gösterilmektedir.

Şekil 7.56'da görebileceğiniz gibi, sonuç kümesindeki her satır için çalışan sayısı aynı kalır. Bu tür bir alt sorgunun kullanımı, sorgudaki bir ana tablo veya tablolarla doğrudan ilgili olmayan diğer tablolardaki verileri dahil etmeniz gereken belirli durumlarla sınırlıdır. Değer, programlama dilindeki bir sabit gibi her satır için aynı kalacaktır. (Satır içi alt sorguların başka bir kullanımını bu bölümün ilerleyen kısımlarında ilişkili alt sorguları incelerken öğreneceksiniz). Çalışan başına katkıyı hesaplayan ifadeyi yazmak için öznitelik listesinde bir diğer ad kullanamayacağınızı unutmayın.

Aynı sorguyu sütun takma adlarını kullanarak yazmanın bir başka yolu da FROM cümlesinde

aşağıdaki gibi bir alt sorgu kullanılmasını gerektirir:

SEÇİNİZ P\_CODE, SALES, ECOUNT, SALES/ECOUNT AS CONTRIB

FROM (SELECT P\_CODE, SUM(LINE\_UNITS \* LINE\_PRICE) AS SALES, (SELECT COUNT(\*) FROM EMPLOYEE) AS ECOUNT FROM LINE

GROUP BY P\_CODE);

**Şekil 7.56 Başka Bir Satır İçi Alt Sorgu Örneği**

Bu durumda, aslında iki alt sorgu kullanıyorsunuz. FROM cümlesindeki alt sorgu önce çalışır ve üç sütunlu bir sanal tablo döndürür: P\_CODE, SALES ve ECOUNT. FROM alt sorgusu, çalışan sayısını ECOUNT olarak döndüren satır içi bir alt sorgu içerir. Dış sorgu iç sorgunun çıktısını aldığından, artık sütun takma adlarını kullanarak dış alt sorgudaki sütunlara başvurabilirsiniz.

## g İlişkili Alt Sorgular

Şimdiye kadar öğrendiğiniz tüm alt sorgular bağımsız olarak yürütülür. , bir komut dizisindeki her bir alt sorgu birbiri ardına seri bir şekilde yürütülür. İlk olarak iç alt sorgu yürütülür; çıktısı dış sorgu tarafından kullanılır ve bu da son dış sorgu bitene kadar yürütülür (koddaki ilk SQL deyimi).

Buna karşılık, **ilişkili** bir **alt sorgu**, dış sorgudaki her satır için bir kez çalıştırılan bir alt sorgudur.

İşlem, bir programlama dilindeki tipik iç içe döngüye benzer. Örneğin:

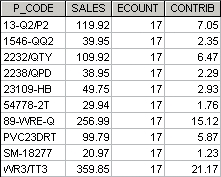
X 5 1 İÇİN 2

Y 5 1'DEN 3'E KADAR

PRINT "X 5 "X, "Y 5 "Y

**ilişkili alt sorgu** Dış sorgudaki her satır için bir kez çalıştırılan bir alt sorgu.

SON



SON

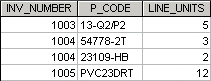
aşağıdaki çıktıyı verecektir:

|  |  |
| --- | --- |
| X 5 1 | Y 5 1 |
| X 5 1 | Y 5 2 |
| X 5 1 | Y 5 3 |
| X 5 2 | Y 5 1 |
| X 5 2 | Y 5 2 |
| X 5 2 | Y 5 3 |

X 5 1 TO 2 dış döngüsünün X 5 1'i ayarlayarak işlemi başlattığına ve ardından her X dış döngü değeri için Y 5 1 TO 3 iç döngüsünün tamamlandığına dikkat edin. İlişkisel DBMS, ilişkili alt sorgu sonuçları üretmek için aynı sırayı kullanır:

1. Dış sorguyu başlatır.
2. Dış sorgu sonuç kümesinin her satırı için, dış satırı iç sorguya geçirerek iç sorguyu çalıştırır.

Bu işlem, daha önce gördüğünüz gibi, ilişkisiz alt sorguların tersidir. Sorguya *ilişkili* alt sorgu denir çünkü iç sorgu dış sorguyla *ilişkilidir*; iç sorgu dış alt sorgunun bir sütununa başvurur.



İlişkili alt sorguyu görmek için, satılan birim değerinin *o* ürün için ortalama satılan birim değerinden büyük olduğu tüm *ürün* satışlarını bilmek istediğinizi varsayalım (t*üm* ürünlerin ortalamasının aksine). Bu durumda, aşağıdaki prosedürün tamamlanması gerekir:

* 1. Bir ürün için satılan ortalama birimleri hesaplayın.
  2. Adım 1'de hesaplanan ortalamayı her bir satış satırında satılan birimlerle karşılaştırın ve

ardından yalnızca satılan birim sayısının daha fazla olduğu satırları seçin.

Aşağıdaki ilişkili sorgu, önceki iki adımlı süreci tamamlar ve sonuçlar Şekil 7.57'de gösterilir.

SEÇİNİZ INV\_NUMBER, P\_CODE, LINE\_UNITS FROM LINE LS

NEREDE LS.LINE\_UNITS . (SELECT AVG(LINE\_UNITS) FROM LINE LA

BURADA LA.P\_CODE 5 LS.P\_CODE);

Şekil 7.57'de, LINE tablosunun birden fazla kez kullanıldığına dikkat edin, bu nedenle tablo takma adlarını kullanmanız gerekir. Bu durumda, iç sorgu, dış sorgu P\_CODE'un P\_CODE koduyla eşleşen ürünün ortalama satılan birimlerini hesaplar. , iç sorgu dış LINE tablosunda bulunan ilk ürün kodunu kullanarak bir kez çalışır ve bu ürün için ortalama satışı döndürür. Dış LINE satırında satılan birim sayısı hesaplanan ortalamadan büyük olduğunda, satır çıktıya eklenir. Ardından iç sorgu, bu kez dış LINE tablosunda bulunan ikinci ürün kodunu kullanarak tekrar çalışır. Bu işlem, iç sorgu dış LINE tüm satırlar için çalışana kadar tekrarlanır. Bu durumda, iç sorgu dış sorgudaki satır sayısı kadar tekrarlanacaktır.

Sonuçları doğrulamak ve alt sorguları nasıl dair bir örnek sunmak için, önceki sorguya

ilişkili bir satır içi alt sorgu ekleyebilirsiniz (bkz. Şekil 7.58).

SEÇİNİZ INV\_NUMBER, P\_CODE, LINE\_UNITS, (SELECT AVG(LINE\_UNITS)

LX HATTINDAN

WHERE LX.P\_CODE 5 LS.P\_CODE) AS AVG

FROM LINE LS

NEREDE LS.LINE\_UNITS . (SELECT AVG(LINE\_UNITS) FROM LINE LA

BURADA LA.P\_CODE 5 LS.P\_CODE);

Gördüğünüz gibi, yeni sorgu, her ürün için satılan ortalama birimleri hesaplayan ilişkili bir satır içi alt sorgu içeriyor. Yalnızca bir yanıt almakla kalmaz, aynı zamanda yanıtın doğru olduğunu da doğrulayabilirsiniz.

**Şekil 7.58 İki İlişkili Alt Sorgu**

İlişkili alt sorgular **EXISTS** özel operatörü ile de kullanılabilir. EXISTS özel operatörü,

başka bir sorgunun sonucuna bağlı olarak bir komutun çalıştırılması gerektiğinde kullanılabilir.

, bir alt sorgu herhangi bir satır döndürürse, ana sorguyu çalıştırın; aksi takdirde, çalıştırmayın. Örneğin, aşağıdaki sorgu tüm satıcıları listeleyecektir, ancak yalnızca sipariş edilecek ürünler varsa:

SEÇİNİZ \*

FROM SATICI

NEREDE VAR (SELECT \* FROM PRODUCT WHERE P\_QOH ,5 P\_MIN);

EXISTS özel operatörü aşağıdaki örnekte tüm satıcıları listelemek için kullanılır, ancak yalnızca

eldeki miktara sahip ve minimum iki katından az ürün varsa:

SEÇİNİZ \*

FROM SATICI

NEREDE VAR (SELECT \* FROM PRODUCT WHERE P\_QOH , P\_MIN \* 2);

Gösterildiği gibi, EXISTS özel operatörü ilişkisiz alt sorgularla kullanılabilir, ancak neredeyse her zaman ilişkili alt sorgularla kullanılır. Örneğin, son zamanlarda sipariş vermiş olan tüm müşterilerin adlarını bilmek istediğinizi varsayalım. Bu durumda, Şekil 7.59'da gösterilen ilk alt sorgu gibi ilişkili bir alt sorgu kullanabilirsiniz.

SEÇİNİZ CUS\_CODE, CUS\_LNAME, CUS\_FNAME FROM MÜŞTERİ

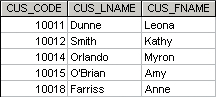
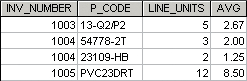
NEREDE EXISTS (SELECT CUS\_CODE

FROM INVOICE

WHERE INVOICE.CUS\_CODE 5 CUSTOMER.CUS\_CODE);

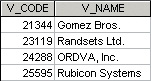
**VAR**

SQL'de, bir alt sorgunun herhangi bir satır döndürüp döndürmediğini kontrol eden bir karşılaştırma operatörü.



Minimum eldeki miktar değerine yaklaşan ürünleri sipariş etmek için hangi satıcılarla iletişime geçmeniz gerektiğini bilmek istediğinizi varsayalım. Özellikle, eldeki miktarı minimum miktarın iki katından az olan ürünler için satıcı kodunu ve satıcı adını bilmek istiyorsunuz. Bu soruyu yanıtlayan sorgu aşağıdaki gibidir (bkz. Şekil 7.60).

**Şekil 7.60 İletişim Kurulacak Satıcılar**



SEÇİNİZ V\_CODE, V\_NAME

FROM SATICI

NEREDE VAR (SELECT \*

ÜRÜNDEN

BURADA P\_QOH , P\_MIN \* 2

VE VENDOR.V\_CODE 5 PRODUCT.V\_CODE);

Şekil 7.60'da şunu not edin:

1. İç ilişkili alt sorgu ilk satıcıyı kullanarak çalışır.
2. Herhangi bir ürün koşulla eşleşirse (eldeki miktar mini miktarın iki katından azsa), satıcı

kodu ve adı çıktıda listelenir.

1. İlişkili alt sorgu ikinci satıcıyı kullanarak çalışır ve tüm satıcılar kullanılana kadar işlem kendini tekrarlar.

# SQL İşlevleri

Veritabanlarındaki veriler, kritik iş bilgilerinin temelini oluşturur. Verilerden bilgi üretmek genellikle birçok veri manipülasyonu gerektirir. Bazen bu tür veri manipülasyonu veri öğelerinin ayrıştırılmasını içerir. Örneğin, bir çalışanın doğum tarihi bir gün, bir ay ve bir yıla bölünebilir. Bir ürün üretim kodu (örneğin, SE-05-2-09-1234-1- 3/12/18-19:26:48) üretim bölgesi, tesis, vardiya, üretim hattı, çalışan numarası, tarih ve saati kaydedecek şekilde tasarlanabilir. Yıllardır geleneksel programlama dilleri, programcıların veri ayrıştırma öncesi gibi veri dönüşümlerini gerçekleştirmelerini sağlayan özel işlevlere sahip olmuştur. Eğer modern bir programlama dili biliyorsanız, bu bölümdeki SQL fonksiyonlarının size tanıdık gelmesi çok muhtemeldir.

SQL fonksiyonları çok kullanışlı araçlardır. Doğum yılına göre sıralanmış tüm çalışanları listelemek istediğinizde ya da Pazarlama departmanınız posta koduna ve telefon numaralarının ilk üç hanesine göre sıralanmış tüm müşterilerin bir listesini oluşturmanızı istediğinde fonksiyonları kullanmanız gerekecektir. Bu iki durumda da, veritabanında bu şekilde bulunmayan veri öğelerini kullanmanız gerekecektir. Bunun yerine, mevcut bir öznitelikten türetilebilecek bir SQL fonksiyonuna ihtiyacınız olacaktır. Fonksiyonlar her zaman bir sayı, tarih veya dize değeri kullanır. Değer, komutun kendisinin bir parçası (bir sabit veya değişmez) veya bir tabloda bulunan bir öznitelik olabilir. Bu nedenle, bir fonksiyon bir SQL deyiminde bir değerin veya bir özniteliğin kullanılabileceği herhangi bir yerde görünebilir.

Aritmetik, trigonometrik, dize, tarih ve zaman fonksiyonları gibi birçok SQL fonksiyonu türü vardır. Bu bölümde tüm bu fonksiyonlar ayrıntılı olarak açıklanmayacak, ancak en kullanışlı olanları hakkında kısa bir genel bakış sunulacaktır.

**Not**

Ana DBMS satıcıları burada ele alınan SQL fonksiyonlarını desteklese de, sözdizimi veya destek derecesi muhtemelen farklılık gösterecektir. Aslında, DBMS satıcıları yeni müşterileri cezbetmek için ürünlerine her zaman kendi fonksiyonlarını eklerler. Bu bölümde ele alınan fonksiyonlar, DBMS'niz tarafından desteklenen fonksiyonların sadece küçük bir kısmını temsil etmektedir. Mevcut fonksiyonların tam listesi için DBMS SQL referans kılavuzunuzu okuyun.

## a Tarih ve Saat İşlevleri

Tüm SQL standardı DBMS'ler tarih ve saat fonksiyonlarını destekler. Tüm tarih fonksiyonları, tarih veya karakter veri türünden bir parametre alır ve bir değer (karakter, sayı veya tarih türü) döndürür. Ne yazık ki, tarih/saat veri türleri farklı DBMS sağlayıcıları tarafından farklı şekilde uygulanmaktadır. Sorun, ANSI SQL standardının tarih veri tiplerini tanımlaması, ancak bu veri tiplerinin nasıl saklanacağını belirtmemesi nedeniyle ortaya çıkmaktadır. Bunun yerine, satıcının bu konuyla ilgilenmesine izin verir.

Tarih/saat işlevleri satıcıdan satıcıya farklılık gösterdiğinden, bu bölümde MS Access, SQL Server ve Oracle için temel tarih/saat işlevleri ele alınacaktır. Tablo 7.8, seçilen MS Access ve SQL Server tarih/zaman işlevlerinin bir listesini göstermektedir.

|  |  |  |
| --- | --- | --- |
| **Tablo 7.8** | **Seçilen MS Access ve SQL Server Tarih/Saat İşlevleri** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **CONVERT (MS SQL Server)**  Convert, daha sonra tartışılacağı gibi çok çeşitli veri türü dönüşümleri gerçekleştirmek için kullanılabilir. Tarih verilerini biçimlendirmek için de kullanılabilir. Sözdizimi:  CONVERT(varchar(length), date\_value, fmt\_code) fmt\_code 5 kullanılan format; olabilir:  1: MM/DD/YY  101: AA/GG/YYYY  2: YY.MM.DD  102: YYYY.MM.DD  3: GG/AA/YY  103: GG/AA/YYYY | | Tüm ürünler için ürün kodunu ve ürünün stoğa en son alındığı tarihi görüntüler: SEÇİNİZ P\_CODE, CONVERT(VARCHAR(8), P\_INDATE, 1)  FROM ÜRÜN;  SEÇİNİZ P\_CODE, CONVERT(VARCHAR(10), P\_INDATE, 102) FROM ÜRÜN; |
| **YIL**  Dört basamaklı bir yıl döndürür Sözdizimi: YEAR(date\_value) | | 1982'de doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, YEAR(EMP\_DOB) AS YEAR  FROM ÇALIŞAN  NEREDE YIL(EMP\_DOB) 5 1982; |
| **AY**  İki basamaklı bir ay kodu döndürür Sözdizimi:  AY(tarih\_değeri) | | Kasım ayında doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, MONTH(EMP\_DOB) AS MONTH  FROM ÇALIŞAN  NEREDE AY(EMP\_DOB) 5 11; |
| **GÜN**  Günün sayısını döndürür Sözdizimi: GÜN(tarih\_değeri) | | Ayın 14. gününde doğan tüm çalışanları listeler: SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB,  DAY(EMP\_DOB) AS DAY  FROM ÇALIŞAN  NEREDE DAY(EMP\_DOB) 5 14; |

*(devam ediyor)*

|  |  |  |
| --- | --- | --- |
| **Tablo 7.8** | **Seçilen MS Access ve SQL Server Tarih/Saat İşlevleri *(Devamı)*** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **DATE() MS Access GETDATE() SQL Server**  Bugünün tarihini döndürür | | Noel'e kaç gün kaldığını listeler (MS Access'te):  SELECT #25-Dec-2022# - DATE();  İki özelliğe dikkat edin:   * Access ve MS SQL Server'da kabul edilebilir olan FROM cümlesi yoktur. * Access'te tarih aritmetiği yaptığınız için Noel tarihi sayı işaretleri (#) içine alınır. MS SQL Server'da:   Geçerli sistem tarihini almak için GETDATE() işlevini kullanın. Tarihler arasındaki farkı hesaplamak  için DATEDIFF fonksiyonunu kullanın (aşağıya bakın). |
| **DATEADD SQL Server**  Bir tarihe seçilen zaman dilimlerinin sayısını ekler Sözdizimi:  DATEADD(datepart, sayı, tarih) | | Belirli bir tarihe bir dizi tarih bölümü ekler. Tarih bölümleri dakika, saat, gün, hafta, ay, çeyrek veya  yıl olabilir. Örneğin:  SEÇİNİZ DATEADD(day,90, P\_INDATE) AS DueDate FROM ÜRÜN;  Yukarıdaki örnek P\_INDATE öğesine 90 gün ekler. MS Access'te aşağıdakileri kullanın:  SEÇİNİZ P\_INDATE190 AS  DueDate FROM ÜRÜN; |
| **DATEDIFF SQL Server**  İki tarihi çıkarır Sözdizimi: DATEDIFF(datepart, startdate, enddate) | | Seçilen bir tarih parçasında ifade edilen iki tarih arasındaki farkı döndürür. Örneğin: SEÇİNİZ DATEDIFF(day, P\_INDATE, GETDATE()) AS DaysAgo  FROM ÜRÜN;  MS Access'te aşağıdakileri kullanın:  SEÇİNİZ DATE()− P\_INDATE AS DaysAgo FROM ÜRÜN; |

Tablo 7.9, Oracle'da kullanılan eşdeğer tarih/zaman fonksiyonlarını göstermektedir. Oracle'ın bir tarihin çeşitli bölümlerini ayıklamak için aynı işlevi (TO\_CHAR) kullandığına dikkat edin. Ayrıca, karakter dizelerini tarih aritmetiğinde kullanılabilecek geçerli bir Oracle tarih biçimine dönüştürmek için başka bir işlev (TO\_DATE) kullanılır.

|  |  |  |
| --- | --- | --- |
| **Tablo 7.9** | **Seçilen Oracle Tarih/Saat Fonksiyonları** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **TO\_CHAR**  Tarih değerinden karakter dizesi veya biçimlendirilmiş  bir dize döndürür Sözdizimi: TO\_CHAR(date\_value, fmt) fmt 5 kullanılan format; : MONTH: ayın adı  MON: üç harfli ay adı MM: iki  basamaklı ay adı  D: haftanın günü için sayı DD: ayın günü için sayı DAY: haftanın gününün adı YYYY: dört basamaklı yıl değeri YY: iki basamaklı yıl değeri | | 1982'de doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, TO\_CHAR(EMP\_DOB, 'YYYY') AS YEAR FROM ÇALIŞAN  NEREDE TO\_CHAR(EMP\_DOB, 'YYYY') 5 '1982';  Kasım ayında doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, TO\_CHAR(EMP\_DOB, 'MM') AS MONTH FROM ÇALIŞAN  NEREDE TO\_CHAR(EMP\_DOB, 'MM') 5 '11';  Ayın 14. gününde doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, TO\_CHAR(EMP\_DOB, 'DD') AS DAY FROM ÇALIŞAN  NEREDE TO\_CHAR(EMP\_DOB, 'DD') 5 '14'; |

*(devam ediyor)*

|  |  |  |
| --- | --- | --- |
| **Tablo 7.9** | **Seçilen Oracle Tarih/Saat Fonksiyonları *(Devamı)*** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **TO\_DATE**  Bir karakter dizesi ve bir tarih biçimi maskesi kullanarak bir tarih değeri döndürür; ayrıca bir tarihi biçimler arasında çevirmek için de kullanılır Sözdizimi:  TO\_DATE(char\_value, fmt) fmt 5 kullanılan format; : MONTH: ayın adı  MON: üç harfli ay adı MM: iki  basamaklı ay adı  D: haftanın günü için sayı DD: ayın günü için sayı DAY: haftanın gününün adı YYYY: dört basamaklı yıl değeri YY: iki basamaklı yıl değeri | | Şirketin 10. yıldönümü tarihinde (11/25/2022) çalışanların yaklaşık yaşlarını listeler: SEÇİNİZ EMP\_LNAME, EMP\_FNAME,  EMP\_DOB, '11/25/2022' AS ANNIV\_DATE,  (TO\_DATE('11/25/2022', 'MM/DD/YYYY')− EMP\_DOB)/365 AS YEARS FROM ÇALIŞAN  ORDER BY YILLAR;  Aşağıdakilere dikkat edin:   * '11/25/2022' bir metin dizesidir, tarih değildir. * TO\_DATE işlevi, metin dizesini tarih aritmetiğinde kullanılan geçerli bir Oracle tarihine çevirir.   Şükran Günü ile 2022 Noel'i arasında kaç gün var? SEÇİNİZ TO\_DATE('2022/12/25', 'YYYY/MM/DD') -  TO\_DATE('NOVEMBER 27, 2022', 'MONTH DD, YYYY')  FROM ÇİFT;  Aşağıdakilere dikkat edin:   * TO\_DATE fonksiyonu, metin dizesini tarih aritmetiğinde kullanılan geçerli bir Oracle tarihine çevirir. * DUAL, Oracle'ın sözde tablosudur ve yalnızca bir tabloya gerçekten ihtiyaç duyulmayan durumlarda   kullanılır. |
| **SYSDATE**  Bugünün tarihini döndürür | | Noel'e kaç gün kaldığını listeler:  SEÇİNİZ TO\_DATE('25-Dec-2022', 'DD-MON-YYY') - SYSDATE FROM ÇİFT;  İki şeye dikkat edin:   * DUAL, Oracle'ın sözde tablosudur ve yalnızca bir tabloya gerçekten ihtiyaç duyulmayan durumlarda   kullanılır.   * Noel tarihi, tarihi geçerli bir tarih biçimine çevirmek için bir TO\_DATE işlevi içine alınır. |
| **ADD\_MONTHS**  Bir tarihe ay veya yıl sayısı ekler Sözdizimi: ADD\_MONTHS(date\_value, n) n  5 ay sayısı | | Tüm ürünleri son kullanma tarihleriyle birlikte listeler (satın alma tarihinden itibaren iki yıl): SEÇİNİZ P\_CODE, P\_INDATE, ADD\_MONTHS(P\_INDATE,24)  FROM ÜRÜN  ORDER BY ADD\_MONTHS(P\_INDATE,24); |
| **SON\_GÜN**  Bir tarih içinde verilen ayın son gününün tarihini döndürür  Sözdizimi: LAST\_DAY(date\_value) | | Bir ayın son yedi günü içinde işe alınan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_HIRE\_DATE FROM ÇALIŞAN  NEREDE EMP\_HIRE\_DATE .5LAST\_DAY(EMP\_HIRE\_DATE)-7; |

Tablo 7.10 MySQL için eşdeğer fonksiyonları göstermektedir.

|  |  |  |
| --- | --- | --- |
| **Tablo 7.10** | **Seçilen MySQL Tarih/Saat İşlevleri** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **Date\_Format** | | Tüm ürünler için ürün kodunu ve ürünün stoğa en son alındığı tarihi görüntüler: |
| Tarih değerinden bir karakter dizesi veya  biçimlendirilmiş bir dize döndürür Sözdizimi: DATE\_FORMAT(date\_value, fmt) fmt 5 kullanılan format; olabilir: | | SEÇİNİZ P\_CODE, DATE\_FORMAT(P\_INDATE, '%m/%d/%y') FROM ÜRÜN;  SEÇİNİZ P\_CODE, DATE\_FORMAT(P\_INDATE, '%M %d, %Y') FROM ÜRÜN; |
| M: ayın adı | |  |
| m: iki basamaklı ay numarası | |  |
| b: kısaltılmış ay adı | |  |
| d: ayın gün sayısı | |  |
| W: hafta içi gün adı | |  |
| a: kısaltılmış hafta içi gün adı | |  |
| Y: dört basamaklı yıl | |  |
| y: iki basamaklı yıl | |  |

*(devam ediyor)*

|  |  |  |
| --- | --- | --- |
| **Tablo 7.10** | **Seçilen MySQL Tarih/Saat İşlevleri *(Devamı)*** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **YIL**  Dört basamaklı bir yıl döndürür Sözdizimi: YEAR(date\_value) | | 1982'de doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, YEAR(EMP\_DOB) AS YEAR  FROM ÇALIŞAN  NEREDE YIL(EMP\_DOB) 5 1982; |
| **AY**  İki basamaklı bir ay kodu döndürür Sözdizimi:  AY(tarih\_değeri) | | Kasım ayında doğan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB, MONTH(EMP\_DOB) AS MONTH  FROM ÇALIŞAN  NEREDE AY(EMP\_DOB) 5 11; |
| **GÜN**  Günün sayısını döndürür Sözdizimi: GÜN(tarih\_değeri) | | Ayın 14. gününde doğan tüm çalışanları listeler: SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_DOB,  DAY(EMP\_DOB) AS DAY  FROM ÇALIŞAN  NEREDE DAY(EMP\_DOB) 5 14; |
| **EKLE TARİHİ** | | Tüm ürünleri 30 gün boyunca rafta bulunacakları tarihle listeleyin. |
| Bir tarihe gün sayısı ekler Sözdizimi:  ADDDATE(date\_value, n) n  5 gün sayısı **DATE\_ADD**  Bir tarihe gün, ay veya yıl sayısı ekler. Bu, daha sağlam olması dışında ADDDATE'e benzer. Kullanıcının | | SEÇİNİZ P\_CODE, P\_INDATE, ADDDATE(P\_INDATE, 30) FROM ÜRÜN  ORDER BY ADDDATE(P\_INDATE, 30);  Tüm ürünleri son kullanma tarihleriyle birlikte listeler (satın alma tarihinden itibaren iki yıl): |
| eklenecek tarih birimini belirtmesine olanak tanır. | | SEÇİNİZ P\_CODE, P\_INDATE, DATE\_ADD(P\_INDATE, INTERVAL 2 YEAR) |
| Sözdizimi: | | FROM ÜRÜN |
| DATE\_ADD(date, INTERVAL n birim)  n 5 eklenecek sayı  birim 5 tarih birimi, : GÜN: | | ORDER BY DATE\_ADD(P\_INDATE, INTERVAL 2 YEAR); |
| n gün ekleyin | |  |
| WEEK: n hafta ekle | |  |
| MONTH: n ay ekle YEAR: | |  |
| n yıl ekle | |  |
| **SON\_GÜN** | | Bir ayın son yedi günü içinde işe alınan tüm çalışanları listeler:  SEÇİNİZ EMP\_LNAME, EMP\_FNAME, EMP\_HIRE\_DATE FROM ÇALIŞAN  NEREDE EMP\_HIRE\_DATE .5 DATE\_ADD(LAST\_DAY (EMP\_HIRE\_DATE), INTERVAL -7 DAY); |
| Bir tarih içinde verilen ayın son gününün tarihini | |
| döndürür | |
| Sözdizimi: | |
| LAST\_DAY(date\_value) | |

## b Sayısal İşlevler

Sayısal fonksiyonlar cebirsel, trigonometrik ve loga- ritmik gibi birçok şekilde gruplandırılabilir. Bu bölümde iki faydalı fonksiyon öğreneceksiniz. Bu bölümde daha önce gördüğünüz SQL toplama fonksiyonlarını bu sayısal fonksiyonlarla karıştırmayın. İlk grup bir değerler kümesi (birden fazla satır - dolayısıyla *toplama fonksiyonları* adı) üzerinde çalışırken, burada ele alınan sayısal fonksiyonlar tek bir satır üzerinde çalışır. Sayısal fonksiyonlar bir sayısal parametre alır ve bir değer döndürür. Tablo 7.11, mevcut sayısal fonksiyonların seçilmiş bir grubunu göstermektedir.

## c Dize İşlevleri

String manipülasyonları programlamada en çok kullanılan fonksiyonlar arasındadır. Herhangi bir programlama dili kullanarak bir rapor oluşturduysanız, karakter dizilerini düzgün bir şekilde sınıflandırmanın, isimleri büyük harfle yazdırmanın veya belirli bir niteliğin uzunluğunu bilmenin önemini bilirsiniz. Tablo 7.12 yararlı dize işleme fonksiyonlarının bir alt kümesini göstermektedir.

|  |  |  |
| --- | --- | --- |
| **Tablo 7.11** | **Seçilen Sayısal Fonksiyonlar** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **ABS** | | Oracle'da aşağıdakileri kullanın: |
| Bir sayının mutlak değerini verir Sözdizimi: | | SEÇİNİZ 1.95, -1.93, ABS(1.95), ABS(-1.93) |
| ABS(numeric\_value) | | FROM ÇİFT; |
|  | | MS Access, MySQL ve MS SQL Server'da aşağıdakileri kullanın: |
|  | | SEÇİNİZ 1.95, -1.93, ABS(1.95), ABS(-1.93); |
| **YUVARLAK**  Bir değeri belirtilen hassasiyete (basamak sayısı) yuvarlar Sözdizimi:  ROUND(numeric\_value, p) p  5 hassasiyet | | Bir ve sıfır ondalık basamağa yuvarlanmış ürün fiyatlarını listeler: SEÇİNİZ P\_CODE, P\_PRICE,  ROUND(P\_PRICE,1) AS PRICE1,  ROUND(P\_PRICE,0) AS PRICE0  FROM ÜRÜN; |
| **TAVAN/TAVAN/ZEMIN**  Sırasıyla, bir sayıdan büyük veya eşit en küçük tamsayıyı döndürür veya bir sayıya eşit veya ondan küçük en büyük tamsayıyı döndürür  Sözdizimi:  CEIL(numeric\_value) Oracle veya MySQL CEILING(numeric\_value) MS SQL Server veya MySQL FLOOR(numeric\_value) | | Ürün fiyatını, ürün fiyatına eşit veya daha büyük en küçük tamsayıyı ve ürün fiyatına eşit veya daha küçük en büyük tamsayıyı listeler.  Oracle veya MySQL'de aşağıdakileri kullanın:  SEÇİNİZ P\_PRICE, CEIL(P\_PRICE), FLOOR(P\_PRICE) FROM ÜRÜN;  MS SQL Server veya MySQL'de aşağıdakileri kullanın:  SEÇİNİZ P\_PRICE, CEILING(P\_PRICE), FLOOR(P\_PRICE) FROM ÜRÜN; |
|  | | MS Access bu işlevleri desteklemez. MySQL'in hem CEIL hem de CEILING'i desteklediğini unutmayın. |

|  |  |  |
| --- | --- | --- |
| **Tablo 7.12** | **Seçilen Dize İşlevleri** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **Birleştirme**  **|| Oracle**  1 **Access ve MS SQL Server & Access**  **CONCAT() MySQL**  İki farklı karakter sütunundan gelen verileri birleştirir ve tek bir sütun döndürür.  Sözdizimi:  strg\_value || strg\_value strg\_value 1 strg\_value & strg\_value CONCAT(strg\_value, )  CONCAT işlevi yalnızca iki dize değerini kabul edebilir, bu nedenle ikiden fazla değer birleştirilecekse iç içe CONCAT işlevleri gerekir. | | Tüm çalışan adlarını listeler (birleştirilmiş).  Oracle'da aşağıdakileri kullanın:  SEÇİNİZ EMP\_LNAME|| ', '|| EMP\_FNAME AS NAME FROM ÇALIŞAN;  Access ve MS SQL Server'da aşağıdakileri kullanın:  SEÇİNİZ EMP\_LNAME 1 ', ' 1 EMP\_FNAME AS NAME FROM ÇALIŞAN;  MySQL'de aşağıdakileri kullanın:  SEÇİNİZ CONCAT(CONCAT(EMP\_LNAME, ', '),  EMP\_FNAME AS NAME  FROM ÇALIŞAN; |
| **UPPER Oracle, MS SQL Server ve MySQL UCASE MySQL ve Access**  **LOWER Oracle, MS SQL Server ve MySQL LCASE MySQL ve Access**  Tümü büyük veya tümü küçük harflerden oluşan bir dize döndürür Sözdizimi:  UPPER(strg\_value) UCASE(strg\_value) LOWER(strg\_value) LCASE(strg\_value) | | Tüm çalışan adlarını büyük harflerle (birleştirilerek) listeler. Oracle'da aşağıdakileri kullanın:  SEÇİNİZ UPPER(EMP\_LNAME|| ', '|| EMP\_FNAME) AS NAME FROM ÇALIŞAN;  MS SQL Server'da aşağıdakileri kullanın:  SEÇİNİZ UPPER(EMP\_LNAME 1 ', ' 1 EMP\_FNAME) AS NAME FROM ÇALIŞAN;  Access'te aşağıdakileri kullanın:  SEÇİNİZ UCASE(EMP\_LNAME & ', ' & EMP\_FNAME) AS NAME FROM ÇALIŞAN;  MySQL'de aşağıdakileri kullanın:  SEÇİNİZ UPPER(CONCAT(CONCAT(EMP\_LNAME, ', '), EMP\_FNAME AS NAME FROM ÇALIŞAN; |

*(devam ediyor)*

|  |  |  |
| --- | --- | --- |
| **Tablo 7.12** | **Seçilen Dize İşlevleri *(Devamı)*** | |
| **Fonksiyon** | | **Örnek(ler)** |
|  | | Tüm çalışan adlarını tüm küçük harflerle (birleştirilerek) listeler. Oracle'da aşağıdakileri kullanın:  SEÇİNİZ LOWER(EMP\_LNAME|| ', '|| EMP\_FNAME) AS NAME FROM ÇALIŞAN;  MS SQL Server'da aşağıdakileri kullanın:  SEÇİNİZ LOWER(EMP\_LNAME 1 ', ' 1 EMP\_FNAME) AS NAME FROM ÇALIŞAN;  Access'te aşağıdakileri kullanın:  SEÇİNİZ LCASE(EMP\_LNAME & ', ' & EMP\_FNAME) AS NAME FROM ÇALIŞAN;  MySQL'de aşağıdakileri kullanın:  SEÇİNİZ LOWER(CONCAT(CONCAT(EMP\_LNAME, ', '), EMP\_FNAME AS NAME FROM ÇALIŞAN; |
| **SUBSTRING**  Verilen dize parametresinin bir alt dizesini veya bir kısmını döndürür  Sözdizimi:  SUBSTR(strg\_value, p, l) Oracle ve MySQL SUBSTRING(strg\_value,p,l) MS SQL Sunucu ve MySQL  MID(strg\_value,p,l) Erişim p  5 başlangıç konumu l 5 karakter uzunluğu  Karakter uzunluğu atlanırsa, fonksiyonlar dize değerinin kalanını döndürür. | | Tüm çalışan telefon numaralarının ilk üç karakterini listeler. Oracle veya MySQL'de aşağıdakileri kullanın:  SEÇİNİZ EMP\_PHONE, SUBSTR(EMP\_PHONE,1,3) AS PREFIX FROM ÇALIŞAN;  MS SQL Server veya MySQL'de aşağıdakileri kullanın:  SEÇİNİZ EMP\_PHONE, SUBSTRING(EMP\_PHONE,1,3) AS PREFIX FROM ÇALIŞAN;  Access'te aşağıdakileri kullanın:  SEÇİNİZ EMP\_PHONE, MID(EMP\_PHONE, 1,3) AS PREFIX FROM ÇALIŞAN; |
| **UZUNLUK**  Bir dize değerindeki karakter sayısını döndürür Sözdizimi: LENGTH(strg\_value) Oracle ve MySQL LEN(strg\_value) MS SQL Server ve Access | | Tüm çalışanların soyadlarını ve adlarının uzunluğunu soyadı uzunluğuna göre azalan  sırada listeler.  Oracle ve MySQL'de aşağıdakileri kullanın:  SEÇİNİZ EMP\_LNAME, LENGTH(EMP\_LNAME) AS NAMESIZE FROM ÇALIŞAN;  MS Access ve SQL Server'da aşağıdakileri kullanın:  SEÇİNİZ EMP\_LNAME, LEN(EMP\_LNAME) AS NAMESIZE FROM ÇALIŞAN; |

## d Dönüştürme Fonksiyonları

Dönüştürme fonksiyonları, belirli bir veri tipindeki bir değeri alıp başka bir veri tipindeki eşdeğer değere dönüştürmenize olanak tanır. Daha önce iki temel Oracle SQL dönüştürme fonksiyonu hakkında bilgi edinmiştiniz: TO\_CHAR ve TO\_DATE. TO\_CHAR fonksiyonunun bir tarih değeri aldığını ve bir gün, bir ay ya da bir yılı temsil eden bir karakter dizesi döndürdüğünü unutmayın. Aynı şekilde, TO\_DATE işlevi de bir tarihi temsil eden bir karakter dizesi alır ve Oracle biçiminde gerçek bir tarih döndürür. SQL Server, bir veri tipini diğerine dönüştürmek için CAST ve CONVERT fonksiyonlarını kullanır. Seçilen fonksiyonların bir özeti Tablo 7.13'te gösterilmektedir.

|  |  |  |
| --- | --- | --- |
| **Tablo 7.13** | **Seçilen Dönüştürme Fonksiyonları** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **Sayısal veya Tarih'ten Karakter'e: TO\_CHAR Oracle**  **CAST Oracle, MS SQL Server, MySQL CONVERT MS SQL Server, MySQL**  **CSTR Erişimi**  Sayısal veya tarih değerinden bir karakter dizesi döndürür.  Sözdizimi: TO\_CHAR(dönüştürülecek değer, fmt) fmt 5 kullanılan format; olabilir:  9 5 bir rakam görüntüler  0 5 önde gelen bir sıfır gösterir  , 5 virgülü görüntüler  . 5 ondalık noktayı gösterir  $ 5 dolar işaretini gösterir B  5 önde gelen boş  S 5 öncü işaret  MI 5 sondaki eksi işareti  CAST (dönüştürülecek değer AS char(uzunluk)) Oracle ve MS SQL Server'ın sayısal verileri sabit uzunluklu veya değişken uzunluklu karakter veri tipine dönüştürmek için CAST kullanabileceğini unutmayın.  MySQL değişken uzunluktaki karakter verilerine CAST yapamaz, sadece sabit uzunlukta yapabilir.  MS SQL Server:  CONVERT(varchar(length), value-to-convert) MySQL:  CONVERT(dönüştürülecek değer, char(uzunluk)) CAST ve CONVERT arasındaki temel fark, CONVERT'in verilerin karakter kümesini değiştirmek için de kullanılabilmesidir.  CSTR(dönüştürülecek değer) | | Biçimlendirilmiş değerleri kullanarak tüm ürün fiyatlarını, ürünün alındığı tarihi ve yüzde indirimi listeler. TO\_CHAR:  SEÇİNİZ P\_CODE, TO\_CHAR(P\_PRICE,'999.99') AS PRICE, (P\_INDATE, 'MM/DD/YYYY') AS INDATE, TO\_CHAR(P\_DISCOUNT,'0.99') AS DISC  FROM ÜRÜN;  Oracle ve MS SQL Server'da CAST:  SEÇİNİZ P\_CODE, CAST(P\_PRICE AS VARCHAR(8)) AS PRICE, CAST(P\_INDATE AS VARCHAR(20)) AS INDATE, CAST(P\_DISCOUNT AS (4)) DİSK OLARAK  FROM PRODUCT; MySQL'de CAST:  SEÇİNİZ P\_CODE, CAST(P\_PRICE AS CHAR(8)) AS PRICE, CAST(P\_INDATE AS CHAR(20)) AS INDATE, CAST(P\_DISCOUNT AS CHAR(4)) DİSK OLARAK  FROM ÜRÜN;  MS SQL Server'da CONVERT:  SEÇİNİZ P\_CODE, CONVERT(VARCHAR(8), P\_PRICE) AS PRICE, CONVERT(VARCHAR(20), P\_INDATE) AS INDATE, CONVERT(VARCHAR(4), P\_DISC) DİSK OLARAK  FROM PRODUCT; MySQL'de CONVERT:  SEÇİNİZ P\_CODE, CONVERT(P\_PRICE, CHAR(8)) AS PRICE, CONVERT(P\_INDATE, CHAR(20)) AS INDATE, CONVERT(P\_DISC, CHAR(4)) DİSK OLARAK  FROM ÜRÜN;  Erişimde CSTR:  SEÇİNİZ P\_CODE, CSTR(P\_PRICE) AS PRICE, CSTR(P\_INDATE) AS INDATE, CSTR(P\_DISC) AS DISCOUNT  FROM ÜRÜN; |
| **Dizeden Sayıya:**  **TO\_NUMBER Oracle**  **CAST Oracle, MS SQL Server, MySQL CONVERT MS SQL Server, MySQL**  **CINT Erişimi**  **CDEC Erişimi**  Bir karakter dizesinden bir sayı döndürür Sözdizimi:  Oracle: TO\_NUMBER(char\_value, fmt) fmt 5 kullanılan format; olabilir:  9 5 bir rakamı gösterir B 5 önde gelen boş  S 5 öncü işaret  MI 5 sondaki eksi işareti  CAST (sayısal veri türü olarak dönüştürülecek değer) INTEGER ve DECIMAL(l,d) türlerine ek olarak Oracle'ın NUMBER'ı ve MS SQL Server'ın NUMERIC'i desteklediğini  .  MS SQL Server:  CONVERT(value-to-convert, decimal(l,d)) MySQL:  CONVERT(dönüştürülecek değer, decimal(l,d)) Dönüştürülecek veri tipi dışında, bu fonksiyonlar yukarıda açıklananlarla aynı şekilde çalışır.  Access'teki CINT, tamsayı veri türündeki sayıyı  döndürürken, CDEC ondalık veri türünü döndürür. | | Metin biçiminde başka bir kaynaktan bir tabloya veri aktarırken metin dizelerini sayısal değerlere dönüştürür; örneğin, burada gösterilen sorgu, verilen biçim maskelerini kullanarak biçimlendirilmiş metni Oracle varsayılan sayısal değerlerine dönüştürmek için TO\_NUMBER işlevini kullanır.  TO\_NUMBER:  SEÇİNİZ TO\_NUMBER('−123.99', 'S999.99'),  TO\_NUMBER('99.78−', 'B999.99MI')  FROM ÇİFT;  OYUNCULAR:  SEÇİNİZ CAST('-123.99' AS DECIMAL(8,2)),  CAST('-99.78' AS DECIMAL(8,2));  CAST işlevi, karakter dizesi üzerindeki sondaki işareti desteklemez. CINT ve CDEC: SEÇİNİZ CINT('-123'), CDEC('-123.99'); |

*(devam ediyor)*

|  |  |  |
| --- | --- | --- |
| **Tablo 7.13** | **Seçilen Dönüştürme İşlevleri *(Devamı)*** | |
| **Fonksiyon** | | **Örnek(ler)** |
| **CASE Oracle, MS SQL Server, MySQL DECODE Oracle**  **ANAHTAR ERİŞİMİ**  Bir özniteliği veya ifadeyi bir dizi değerle karşılaştırır ve ilişkili bir değer ya da eşleşme bulunamazsa varsayılan bir döndürür  Sözdizimi:  DECODE:  DECODE(e, x, y, d)  e 5 nitelik veya ifade  x 5 e ile karşılaştırılacak değer y 5 e  5 x içinde döndürülecek değer  d 5 e, x'e eşit değilse döndürülecek varsayılan değer CASE:  DURUM Koşul olduğunda THEN değer1 ELSE değer2 END **ŞALTER**:  **SWITCH(e1, x, e2, y, TRUE, d)**  e1 5 karşılaştırma ifadesi  x 5 e1 doğruysa döndürülecek değer e2 5 karşılaştırma ifadesi  y 5 e2 doğruysa döndürülecek değer  TRUE 5 sonraki değerin varsayılan olduğunu belirten anahtar sözcük  d 5 ifadelerden hiçbiri doğru değilse  döndürülecek varsayılan değer | | Aşağıdaki örnek, belirtilen eyaletler için satış vergisi oranını döndürür: V\_STATE öğesini 'CA' ile karşılaştırır; değerler eşleşirse .08 değerini döndürür.  V\_STATE'i 'FL' ile karşılaştırır; değerler eşleşirse .05 döndürür. V\_STATE değerini 'TN' ile karşılaştırır; değerler eşleşirse .085 döndürür. Eşleşme yoksa 0,00 (varsayılan değer) döndürür.  SEÇİNİZ V\_CODE, V\_STATE, DECODE(V\_STATE,'CA',.08,'FL',.05, 'TN',.085, 0.00) AS TAX  FROM SATICI; DURUM:  SEÇİNİZ V\_CODE, V\_STATE,  DURUM V\_STATE 5 'CA' OLDUĞUNDA .08  V\_STATE 5 'FL' OLDUĞUNDA .05  WHEN V\_STATE 5 'TN' THEN .085 ELSE 0.00 END AS TAX  FROM SATICI;  ŞALTER:  SEÇİNİZ V\_CODE, V\_STATE, SWITCH(V\_STATE 5'CA',.08, V\_STATE 5 'FL',.05, V\_STATE 5  'TN',.085, TRUE, 0.00) VERGI OLARAK  FROM SATICI; |

# İlişkisel Küme Operatörleri

Bölüm 3'te, sekiz genel ilişkisel operatör hakkında bilgi edindiniz. Bu bölümde, birleşim, kesişim ve fark ilişkisel operatörlerini uygulamak için üç SQL operatörünü (UNION, INTERSECT ve EXCEPT (MINUS)- nasıl kullanacağınızı öğreneceksiniz.

SQL veri işleme komutlarının küme odaklı olduğunu da öğrendiniz; yani, bir kerede tüm satır ve sütun kümeleri (tablolar) üzerinde çalışırlar. Yeni kümeler (veya ilişkiler) oluşturmak için iki veya daha fazla kümeyi birleştirebilirsiniz. UNION, INTERSECT ve EXCEPT (MINUS) deyimleri tam olarak bunu yapar. İlişkisel veritabanı terimlerinde, *kümeler*, *ilişkiler* ve *tablolar* kelimelerini birbirlerinin yerine kullanabilirsiniz, çünkü hepsi ilişkisel veritabanı kullanıcısına sunulduğu şekliyle veri kümesinin kavramsal bir görünümünü sağlar.

**Not**

SQL standardı, tüm DBMS'lerin veriler üzerinde gerçekleştirmesi gereken işlemleri tanımlar, ancak uygulama ayrıntılarını DBMS satıcılarına bırakır. Bu nedenle, bazı gelişmiş SQL özellikleri tüm DBMS uygulamalarında çalışmayabilir. Ayrıca, bazı DBMS satıcıları SQL standardında bulunmayan ek özellikler uygulayabilir. SQL standardı UNION, INTERSECT ve EXCEPT'i UNION, INTERSECT ve DIFFER- ENCE ilişkisel operatörleri için anahtar kelimeler olarak tanımlar ve bunlar MS SQL Server'da kullanılan isimlerdir. Ancak Ora- cle, EXCEPT yerine DIFFERENCE operatörünün adı olarak MINUS kullanır. Diğer RDBMS sağlayıcıları farklı bir operatör adı kullanabilir veya belirli bir operatörü hiç uygulamayabilir. Örneğin, Access ve MySQL INTERSECT veya DIF- FERENCE işlemleri için doğrudan desteğe sahip değildir, çünkü bu işlevsellik birleştirme ve alt sorgu kombinasyonları kullanılarak elde edilebilir. ANSI/ISO SQL standartları hakkında daha fazla bilgi edinmek ve en son standart belgeleri elektronik ortamda nasıl edinebileceğinizi öğrenmek için ANSI web sitesini (www

.ansi.org).

UNION, INTERSECT ve EXCEPT (MINUS) yalnızca ilişkiler *union uyumlu* olduğunda düzgün çalışır; bu da özniteliklerin sayısının aynı olması ve karşılık gelen veri türlerinin aynı olması gerektiği anlamına gelir. Uygulamada, bazı RDBMS satıcıları veri türlerinin uyumlu olmasını ancak tam olarak aynı olmamasını gerektirir. Örneğin, uyumlu veri türleri VARCHAR

(35) ve CHAR (15). Her iki nitelik de karakter (dize) değerlerini depolar; tek fark dize boyutudur. Uyumlu veri türlerine bir başka örnek de NUMBER ve SMALLINT'tir. Her iki veri türü de sayısal değerleri saklamak için kullanılır.

**Not**

Bazı DBMS ürünleri, birleştirme uyumlu tabloların ayn*ı* veri türlerine sahip olmasını gerektirebilir.

## BİRLİK

SaleCo'nun başka bir şirketi satın aldığını varsayalım. SaleCo'nun yönetimi, satın alınan şirketin müşteri listesinin kendi müşteri listesiyle düzgün bir şekilde birleştirildiğinden emin olmak istiyor. Bazı müşteriler her iki şirketten de mal satın almış olabileceğinden, iki liste ortak müşteriler içerebilir. SaleCo'nun yönetimi, iki müşteri listesi müşteri kayıtlarının yinelenmediğinden emin olmak istiyor. UNION sorgusu, yinelenen kayıtları hariç tutan birleşik bir müşteri listesi oluşturmak için mükemmel bir araçtır.

UNION deyimi, iki veya daha fazla sorgudaki satırları *çift satırları dahil etmeden* birleştirir.

UNION deyiminin sözdizimi şöyledir:

*sorgu* UNION *sorgu*

Başka bir deyişle, UNION deyimi iki SELECT sorgusunun çıktısını birleştirir. (SELECT deyimlerinin union uyumlu olması gerektiğini unutmayın. , aynı sayıda öznitelik ve benzer veri türleri döndürmelidirler).

SQL'de UNION deyiminin kullanımını göstermek için Ch07\_SaleCo veritabanındaki CUSTOMER ve CUSTOMER\_2 tablolarını kullanın. Birleştirilmiş CUSTOMER ve CUSTOMER\_2 kayıtlarını kopyalar olmadan göstermek için UNION sorgusu aşağıdaki gibi yazılır:

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

FROM MÜŞTERİ BİRLİĞİ

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

FROM MÜŞTERİ\_2;

Şekil 7.61'de CUSTOMER ve CUSTOMER\_2 tablolarının içerikleri ve UNION sorgusunun sonucu gösterilmektedir. Burada sonuçları göstermek için MS Access kullanılmış olsa da, benzer sonuçlar Oracle, MS SQL Server ve MySQL ile de elde edilebilir.

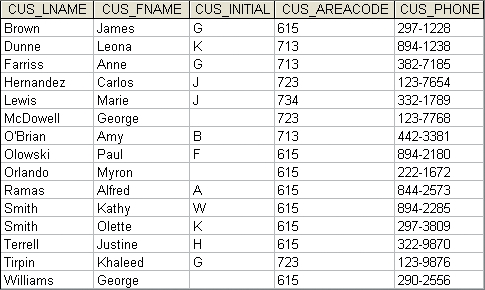
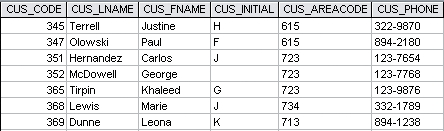
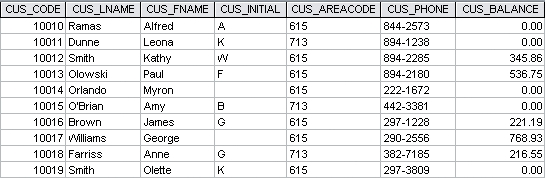
Şekil 7.61'de aşağıdakilere dikkat edin:

* CUSTOMER tablosu 10 satır içerirken, CUSTOMER\_2 tablosu yedi satır içerir.
* Müşteriler Dunne ve Olowski, CUSTOMER tablosunun yanı sıra CUSTOMER\_2 tablosuna da dahil edilmiştir.
* UNION sorgusu 15 kayıt verir çünkü Dunne ve Olowski müşterilerinin yinelenen kayıtları

dahil edilmemiştir. Kısacası, UNION benzersiz bir kayıt kümesi verir.

**UNION Sorgu Sonuçları**

**Şekil 7.61**



Tablo adı: CUSTOMER\_2

Veritabanı adı: Ch07\_SaleCo Sorgu adı: qryUNION-of-CUSTOMER-and-CUSTOMER\_2

Tablo adı: CUSTOMER

**Not**

SQL standardı, UNION SQL durumu kullanıldığında yinelenen satırların ortadan kaldırılmasını gerektirir. Ancak, bazı DBMS satıcıları bu standarda uymayabilir. UNION deyiminin desteklenip desteklenmediğini ve nasıl desteklendiğini görmek için DBMS kılavuzunuzu kontrol edin.

UNION deyimi ikiden fazla sorguyu birleştirmek için kullanılabilir. Örneğin, T1, T2, T3 ve T4 adında union uyumlu dört sorgunuz olduğunu varsayın. UNION deyimi ile dört sorgunun çıktısını tek bir sonuç kümesinde birleştirebilirsiniz. SQL deyimi şuna benzer olacaktır:

SELECT column-list FROM T1 UNION

SELECT column-list FROM T2 UNION

SELECT column-list FROM T3 UNION

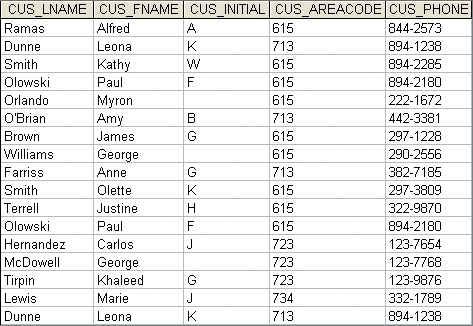
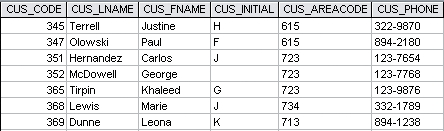
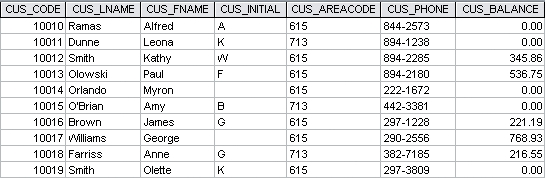
SELECT column-list FROM T4;

## b TÜM BİRLİK

SaleCo'nun yönetimi *hem* CUSTOMER *hem* de CUSTOMER\_2 listelerinde kaç müşteri olduğunu bilmek isterse, yinelenen satırları tutan bir ilişki üretmek için UNION ALL sorgusu kullanılabilir. Bu nedenle, aşağıdaki sorgu her iki sorgudaki tüm satırları (yinelenen satırlar dahil) tutacak ve 17 satır döndürecektir.

**UNION ALL Sorgu Sonuçları**

**Şekil 7.62**



Tablo adı: CUSTOMER\_2

Veritabanı adı: Ch07\_SaleCo Sorgu adı: qryUNION-ALL-of-CUSTOMER-and-CUSTOMER\_2

Tablo adı: CUSTOMER

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

FROM MÜŞTERİ BİRLİĞİ TÜM

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

FROM MÜŞTERİ\_2;

Önceki UNION ALL sorgusu çalıştırıldığında Şekil 7.62'de gösterilen sonuç elde edilir.

UNION deyimi gibi, UNION ALL deyimi de ikiden sorguyu birleştirmek için kullanılabilir.

## c INTERSECT

SaleCo'nun yönetimi CUS- TOMER ve CUSTOMER\_2 tablolarında hangi müşteri kayıtlarının yinelendiğini bilmek isterse, INTERSECT deyimi iki sorgudan satırları birleştirmek için kullanılabilir ve yalnızca her iki kümede de görünen satırları döndürür. INTERSECT deyimi için sözdizimi şöyledir:

*sorgu* INTERSECT *sorgu*

Yinelenen müşteri kayıtlarının listesini oluşturmak için aşağıdaki komutu kullanabilirsiniz: SEÇİNİZ CUS\_LNAME, CUS\_FNAME, CUS\_INITIAL, CUS\_AREACODE,

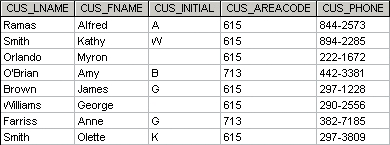
CUS\_PHONE

FROM MÜŞTERI KESIŞIMI

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

FROM MÜŞTERİ\_2;

INTERSECT deyimi, ek yararlı müşteri bilgileri oluşturmak için kullanılabilir. Örneğin, aşağıdaki sorgu 615 alan kodunda bulunan ve satın alma işlemi yapmış olan tüm müşterilerin müşteri kodlarını döndürür. (Bir müşteri satın alma işlemi yaptıysa, o müşteri için bir fatura kaydı ).



SEÇİNİZ CUS\_CODE FROM CUSTOMER WHERE CUS\_AREACODE 5 '615' INTERSECT

SEÇİNİZ DISTINCT CUS\_CODE FROM INVOICE;

Şekil 7.63 sonucu göstermektedir.

**Şekil 7.63 INTERSECT Sorgu Sonuçları**

## d HARİÇ (EKSİ)

SQL'deki EXCEPT deyimi iki sorgudaki satırları birleştirir ve yalnızca ilk kümede görünen ancak ikincisinde görünmeyen satırları döndürür. MS SQL Server'da EXCEPT deyimi ve Oracle'da MINUS deyimi için sözdizimi şöyledir:

*sorgu* EXCEPT *sorgu*

ve

*sorgu* EKSİ *sorgu*

Örneğin, SaleCo yöneticileri CUSTOMER tablosundaki hangi müşterilerin CUSTOMER\_2 tablosunda bulunmadığını öğrenmek isterlerse, Oracle'da aşağıdaki komutu kullanabilirler (bkz. Şekil 7.64).

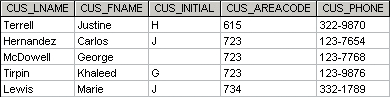
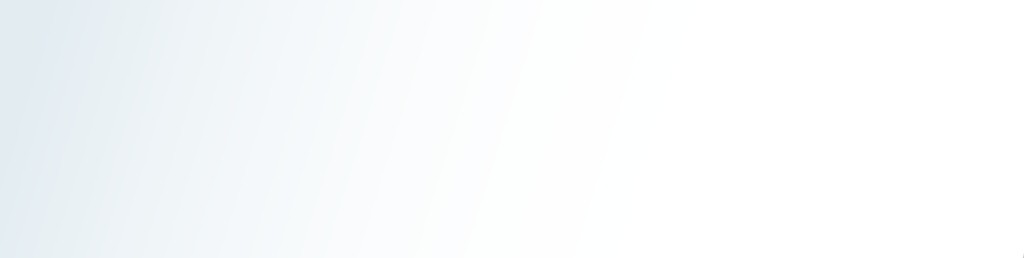
SEÇİNİZ CUS\_LNAME, CUS\_FNAME, CUS\_INITIAL, CUS\_AREACODE, CUS\_PHONE

FROM MÜŞTERİ EKSİ

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

FROM MÜŞTERİ\_2;

Yöneticiler CUSTOMER\_2 tablosundaki hangi müşterilerin CUSTOMER tablosunda



bulunmadığını bilmek isterlerse, sadece tablo isimlerini değiştirirler (bkz. Şekil 7.65):

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

FROM MÜŞTERI\_2 EKSI

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

FROM MÜŞTERİ;

**Şekil 7.65 CUSTOMER\_2 MINUS CUSTOMER Sorgu Sonuçları**

MS SQL Server kullanıcıları MINUS yerine EXCEPT anahtar sözcüğünü kullanabilir, ancak bunun dışında sözdizimi aynıdır. MINUS'u WHERE gibi çeşitli cümleciklerle birleştirerek yararlı bilgiler elde edebilirsiniz. Örneğin, aşağıdaki sorgu 615 alan kodundaki tüm müşterilerin müşteri kodlarını, alışveriş yapmış olanları çıkararak döndürür ve 615 alan kodundaki alışveriş yapmamış müşterileri bırakır.

SEÇİNİZ CUS\_CODE FROM CUSTOMER WHERE CUS\_AREACODE 5 '615' EXCEPT

SEÇİNİZ DISTINCT CUS\_CODE FROM INVOICE;

## e Sözdizimi Alternatifleri

DBMS'niz INTERSECT veya EXCEPT (MINUS) ifadelerini desteklemiyorsa, aynı çıktıyı elde etmek için alternatif sözdizimi kullanabilirsiniz. Örneğin, INTERSECT sorgusu:

SEÇİNİZ CUS\_AREACODE FROM CUSTOMER INTERSECT

SEÇİNİZ SATICIDAN V\_AREACODE;

INTERSECT operatörü kullanılmadan aşağıdaki şekilde yeniden üretilebilir: SEÇİNİZ DISTINCT CUS\_AREACODE

FROM CUSTOMER JOIN VENDOR ON CUS\_AREACODE 5 V\_AREACODE;

SQL, belirli bir sorunu çeşitli şekillerde çözmenize olanak tanır. IN ve NOT IN alt sorguları diğer INTERSECT ve MINUS sorgularının sonuçlarını elde etmek için kullanılabilir. Örneğin, aşağıdaki sorgu Şekil 7.63'te gösterilen INTERSECT sorgusu ile aynı sonuçları üretecektir:

SEÇİNİZ CUS\_CODE FROM CUSTOMER WHERE CUS\_AREACODE 5 '615' VE

CUS\_CODE IN (SELECT DISTINCT CUS\_CODE FROM INVOICE);

MINUS deyiminin aynı alternatifini kullanarak, aşağıdakileri girerek yukarıda gösterilen EXCEPT sorgusu için çıktı oluşturabilirsiniz:

SEÇİNİZ CUS\_CODE FROM CUSTOMER WHERE CUS\_AREACODE 5 '615' VE

CUS\_CODE NOT IN (SELECT DISTINCT CUS\_CODE FROM INVOICE);

# SELECT Sorguları Oluşturma

Bu bölümde gördüğünüz gibi, SQL dili hem basit hem de karmaşıktır. Her bir cümle ve fonksiyon kendi başına basittir ve iyi tanımlanmış bir görevi yerine getirir. Ancak, SQL dilinin esnekliği nedeniyle, bir bilgi talebini karşılamak için uygun cümle ve fonksiyonları birleştirmek oldukça karmaşık hale gelebilir. Bir sorgu oluşturmaya çalışırken, aşağıdakiler akılda tutulması gereken yararlı önerilerdir.

## a Verilerinizi Tanıyın

İçinde çalıştığınız veri modelini anlamanın önemi abartılamaz. Akademik kurslardaki veritabanları normalde iyi tasarlanmış, iyi yapılandırılmış ve en iyi uygulamaları takip eder. Gerçek dünya veritabanları dağınıktır. Tablo ve öznitelik adları genellikle şifreli, kafa karıştırıcı ve standartlaştırılmamıştır. Tablolar uygun kısıtlamalara sahip olmayabilir ve bazı durumlarda tanımlanmış bir birincil anahtarları bile olmayabilir! Bu ilişkiyi uygulamak için bir yabancı anahtara sahip olmayan ilgili veri tabloları bulmak nadir değildir.

Sorun, veritabanı uzmanlarının işlerini yapmakta yetersiz olmaları değildir. Unutmayın, çoğu veritabanı sistemi bir kuruluşta onlarca yıl hizmet vermeye devam eder. İşletme yıllar içinde değiştikçe, büyüdükçe, sözleşme yaptıkça, birleştikçe ve bölündükçe, dahili sistemlerin uyarlanması ve değiştirilmesi gerekir. Bu değişiklikler genellikle sistem içinde kurumsallaşan uzlaşmaları içerir. Örneğin, yazarlar bir sağlık kuruluşunda yıllar önce gerçekleşen bir şirket birleşmesi nedeniyle hasta tedavisiyle ilgili verileri içeren birden fazla tabloya sahip bir veritabanına aşinadır. Bir tabloda, PID (hasta kimliği) özniteliği tedavi gören kişi için bir tanımlayıcıdır. İkinci tabloda ise PID niteliği, tedavinin sigortasına fatura edildiği kişi için bir tanımlayıcıdır. Bu ortamdaki SQL programcıları çok sayıda kafa karıştırıcı tablo ve sütun adıyla uğraşır.

Yeni bir ortamı kavramak zor olabilir, ancak çalıştığı veri modelini bilmeyen bir SQL programcısı, soruları yanıtlamak için hangi verilerin mevcut olduğunu, verilerin nasıl ilişkili olduğunu veya bunlara nasıl erişileceğini bilemeyecektir. Yeni bir veritabanı uzmanı olarak, kendinizi yüzlerce tablo ile çalışmaktan sorumlu olduğunuz bir ortama atılmış bulabilirsiniz. Bu zaman alacaktır, ancak verileri öğrenmek ve anlamak için çalışmakta gayretli olun.

## b Sorunu Tanıyın

Veri modelini anlamanız gerektiği gibi, yanıtlamaya çalıştığınız soruyu da anlamanız gerekir. Bilgi raporlama talepleri çeşitli kaynaklardan gelecektir. Bazı talepler tek seferlik olaylardır, bazıları ise bir uygulama veya veri analizi süreci içinde devam eden işlemlerin bir parçası haline gelecektir. Bilgi talepleri genellikle belirsizdir ve talebi yapan kişi bunun farkında olmasa bile birden fazla yoruma tabidir. Örneğin, bir pazarlama müdürünün şirketin belirli bir ürünü sattığı ortalama fiyatı bilmek istediği bir senaryo düşünün. Ürünün 10 kez satıldığını varsayalım

şu değerlerle: 10$, 10$, 10$, 20$, 10$, 10$, 30$, 10$, 10$ ve 10$. Pazarlama müdürü hangisini istiyordu?

* Gerçekleşen tüm satışlar için ortalama fiyat:

10 1 10 1 10 1 10 1 10 1 10 1 10 1 10 1 20 1 30 5 130 / 10 5 $13

Olarak kodlanmıştır: SELECT AVG(SATIŞ\_FIYATI)

* Herhangi bir satışın gerçekleştiği fiyatların ortalaması: 10 1 20 1 30 5 60 / 3 5 $20

Olarak kodlanmıştır: SELECT AVG(DISTINCT SALE\_PRICE)

Yalnızca 10 satırlık veriyle bile, olası yanıtlar arasındaki belirgin fark hızla ortaya çıkmaktadır. Bu konu çok önemlidir çünkü pazarlama müdürü talebin ne kadar muğlak olduğunu düşünmemiş olabilir. Bir cevap sunulduğunda, kararlar sunulan bilgilere göre verilecektir. Yönetici ve programcı arasında iletişimsizlik varsa, işletme önemli sonuçları olan kötü bir karar verebilir.

## c Her Seferinde Bir Madde Oluşturun

Sorunu anladıktan ve veri modelinizi bildikten sonra, sorunu verilerle eşleştirebilmeniz için asıl sorguyu oluşturabilirsiniz. Bir SELECT sorgusundaki cümleciklerin birlikte nasıl çalıştığını hatırlayarak, cümleciklerinizi aşağıdaki sırayla oluşturmanız yararlı olabilir:

* FROM
* NEREDE
* GROUP BY
* SAHİP OLMAK
* SEÇİNİZ
* ORDER BY

Hangi tabloların gerekli verileri içerdiğini anlamak gereksinimleri veri modeliyle eşleştirin. Performans nedenleriyle, sorgunuzu yanıtlamak için mümkün olan en küçük ilgili tablo kümesini kullanın. Örneğin, bir sorgu yalnızca satıcı kodu ve ürün tanımı özniteliklerini gerektiriyorsa, Şekil

7.1'e bakıldığında PRODUCT tablosunun bu özniteliklerin her ikisini de içerdiği görülür. Bu nedenle, VENDOR tablosunu sorguya dahil etmek için bir neden olmayacaktır. Gerekli tabloları birleştirmek için uygun FROM cümlesini yazın. SELECT sütun listesi için basit bir SELECT \* ile başlayabilirsiniz, böylece FROM cümlenizin amaçladığınız verileri alıp almadığını test edebilirsiniz. Gerektiğinde doğru dış birleştirmeleri kullandığınızdan emin olun. Performans nedenleriyle, iç birleştirme yeterli olacaksa dış birleştirme kullanmayın. Ardından, FROM cümlesi tarafından döndürülen tüm satırları sonucunuzda isteyip istemediğinize karar verin. Değilse, WHERE cümlesinde verileri yalnızca gereksinimleri karşılayan satırlarla kısıtlamak için kullanılabilecek bir veya daha fazla kriter yazın. Tüm satırlar , bir WHERE

cümlesi

gerekli değildir.

Sorgunuzun bir toplam değer döndürmesi gerekecek mi? Eğer öyleyse, verilerin gruplandırılacağı uygun atribütleri belirleyin. Herhangi bir toplu değer döndürülmeyecekse GROUP BY cümlesine gerek yoktur. GROUP BY cümlesi gerekli değilse HAVING cümlesi de gerekli değildir. HAVING cümlesinin grupları kısıtlamak için kullanıldığını unutmayın. Grup HAVING'e gerek yoktur. Sorgu bir GROUP BY cümlesi kullanıyorsa, tüm grupların yanıtta döndürülüp döndürülmeyeceğine karar verin. Eğer öyleyse, HAVING cümlesine gerek yoktur. Bazı grupların sonuca dahil edilmemesi gerekiyorsa, HAVING cümlesinde grupları yalnızca ilgilenilen gruplarla sınırlayan kriterler yazın. Ayrıca, HAVING cümlesinden bu yana

bir gruptaki tek tek satırları kısıtlayamaz, tüm gruba uygulanmalıdır; ölçüt bir toplama işlevi içermelidir. Tüm grup için geçerli olan ancak bir toplama işlevi içermeyen bir ölçüt yazabiliyorsanız, bu ölçüt muhtemelen WHERE cümlesine dahil edilmelidir.

Ardından, SELECT sütun listesinde döndürülmesi gereken öznitelikleri ve toplamları belirtin. Türetilmiş özniteliklerin döndürülmesi gerekiyorsa, bunları hesaplamak için formülleri SELECT'e eklemeyi unutmayın. Ayrıca, DISTINCT anahtar sözcüğünün gerekli olup olmadığını da göz önünde bulundurun. Performans nedenleriyle, gerekmiyorsa DISTINCT anahtar sözcüğünü dahil etmeyin. Sorgu, bastırılması gereken yinelenen çıktı satırları döndürüyorsa, SELECT anahtar sözcüğünden hemen sonra DISTINCT sözcüğünü yerleştirin. GROUP BY cümlesi, yinelenenleri, toplama işlevi tarafından tek bir satıra indirgenen tek bir koleksiyonda birleştireceğinden, bir toplama işlevi döndürülüyorsa normalde böyle bir durumun söz konusu olmaması gerektiğini unutmayın. Ancak, bir toplama işlevi kullanılıyorsa, toplamanın hesaplanması sırasında yinelenen değerlerin bastırılıp bastırılmayacağını düşünün ve eğer öyleyse, toplama işlevinin içine DISTINCT ekleyin.

Son olarak, nihai çıktıdaki satırların sıralamasını göz önünde bulundurun. Performans nedenleriyle, nihai çıktıdaki satırların sırası önemli değilse ORDER BY cümlesini atlayın. Ancak, sıralama önemliyse, sıralama için kullanılması gereken özniteliği veya öznitelikleri belirleyin. SELECT sorgusundaki herhangi bir cümlenin oluşturulması sırasında, sorgunun uygun şekilde kullanabilmesi için verilerin önceden işlenmesi gerektiğini belirlerseniz, bir alt sorgu gerekebilir.

# Özet

* SQL komutları iki genel kategoriye ayrılabilir: veri tanımlama dili (DDL) komutları ve veri manipülasyon dili (DML) komutları.
* ANSI standart veri tipleri tüm RDBMS sağlayıcıları tarafından farklı şekillerde desteklenmektedir. Veri türlerinin temel kategorileri karakter tarihi, sayısal veriler ve tarih verileridir.
* SELECT deyimi SQL'deki ana veri alma komutudur. Bir SELECT deyimi aşağıdaki sözdizimine sahiptir:

SEÇİNİZ *sütun listesi*

FROM *tablelist*

[NEREDE *koşul listesi*] [GROUP BY *sütun listesi*] [HAVING *koşul listesi*]

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

* Sütun listesi, virgülle ayrılmış bir veya daha fazla sütun adını temsil eder. Sütun listesi ayrıca hesaplanmış sütunları, takma adları ve toplama işlevlerini de içerebilir. Hesaplanan bir sütun bir ifade veya formülle temsil edilir (örneğin, P\_PRICE \* P\_QOH). FROM cümlesi tablo adlarının bir listesini içerir.
* Tabloları birleştiren işlemler iç birleştirme ve dış birleştirme olarak sınıflandırılabilir. İç birleştirme, yalnızca belirli bir kriteri karşılayan satırların seçildiği geleneksel birleştirmedir.

Dış birleştirme, eşleşen satırların yanı sıra bir tablo veya birleştirilecek her iki tablo için eşleşmeyen öznitelik değerlerine sahip satırları da döndürür.

* Doğal bir birleştirme, eşleşen sütunlarda eşleşen değerlere sahip tüm satırları döndürür ve yinelenen sütunları ortadan kaldırır. Bu sorgu tarzı, tablolar ortak bir adla ortak bir özniteliği paylaştığında kullanılır. Doğal birleştirme ve eski tarz birleştirme sözdizimi arasındaki önemli bir fark, doğal birleştirmenin ortak nitelikler için bir tablo niteleyicisinin gerektirmemesidir. Uygulamada, doğal birleştirmeler genellikle tavsiye edilmez çünkü ortak nitelik komut içinde belirtilmez ve sorguların anlaşılması ve sürdürülmesi daha zor hale gelir.
* Birleştirmeler USING ve ON gibi anahtar sözcükler kullanabilir. USING cümlesi kullanılırsa, sorgu yalnızca USING belirtilen sütunda eşleşen değerlere sahip satırları döndürür; bu sütun her iki tabloda da bulunmalıdır. ON cümlesi kullanılırsa, sorgu yalnızca belirtilen birleştirme koşulunu karşılayan satırları döndürür.
* ORDER BY cümlesi, bir SELECT deyiminin çıktısını sıralamak için kullanılır. ORDER BY cümlesi bir veya daha fazla sütuna göre sıralayabilir ve artan veya azalan sıralama kullanabilir.
* WHERE cümlesi, SELECT, UPDATE ve DELETE

deyimleriyle birlikte, aşağıdaki öğeleri kısıtlamak için

kullanılabilir

DDL komutundan etkilenen satırlar. Koşul listesi, mantıksal operatörlerle (AND, OR ve NOT) ayrılmış bir veya daha fazla koşullu ifadeyi temsil eder. Koşullu ifade, herhangi bir karşılaştırma operatörünün (5, ., ,,

.5, ,5 ve ,.) yanı sıra özel operatörler (BETWEEN, IS NULL, LIKE, IN ve EXISTS) içerebilir.

* + Toplama fonksiyonları (COUNT, MIN, MAX ve AVG) bir dizi satır üzerinde aritmetik hesaplamalar gerçekleştiren özel fonksiyonlardır. Toplama fonksiyonları, toplama hesaplamalarının çıktısını bir veya daha fazla özelliğe göre gruplandırmak için genellikle GROUP BY cümlesiyle birlikte kullanılır. HAVING cümlesi, yalnızca belirli bir koşulla eşleşen toplama satırlarını seçerek GROUP BY cümlesinin çıktısını kısıtlamak için kullanılır.
  + Alt sorgular ve ilişkili sorgular, işlenmiş *diğer* verilere dayalı olarak veri işlemek gerektiğinde kullanılır. , sorgu daha önce bilinmeyen ve başka bir sorgu tarafından oluşturulan sonuçları kullanır. Alt sorgular, bir SELECT deyimindeki FROM, WHERE, IN ve HAVING cümleleriyle birlikte kullanılabilir. Bir alt sorgu tek bir satır veya birden fazla satır döndürebilir.
  + Çoğu alt sorgu seri bir şekilde yürütülür. Yani, dış sorgu veri isteğini başlatır ve ardından iç alt sorgu yürütülür. Buna karşılık, ilişkili bir alt sorgu, dış sorgudaki her satır için bir kez çalıştırılan bir alt sorgudur. Bu işlem, bir programlama dilindeki tipik iç içe döngüye benzer. İlişkili bir alt sorgu, iç sorgunun ilişkili olması nedeniyle bu şekilde adlandırılır

# Anahtar Terimler

Dış sorguya - iç sorgu, dış alt sorgunun bir sütununa başvurur.

* SQL fonksiyonları verileri ayıklamak veya dönüştürmek için kullanılır. En sık kullanılan fonksiyonlar tarih ve saat fonksiyonlarıdır. Fonksiyon çıktısının sonuçları, değerleri bir veritabanı tablosunda saklamak, türetilmiş değişkenlerin hesaplanmasına temel teşkil etmek veya veri karşılaştırmalarına temel teşkil etmek için kullanılabilir. Fonksiyon formatları operatöre özel olabilir. Saat ve tarih fonksiyonlarının yanı sıra, sayısal ve dize fonksiyonlarının yanı sıra bir veri formatını diğerine dönüştüren dönüştürme fonksiyonları da vardır.
* SQL, yeni bir ilişki oluşturmak üzere iki sorgunun çıktısını birleştirmek için ilişkisel küme işleçleri sağlar. UNION ve UNION ALL küme işleçleri iki veya daha fazla sorgunun çıktısını birleştirir ve her iki sorgudaki tüm benzersiz (UNION) veya yinelenen (UNION ALL) satırları içeren yeni bir ilişki üretir. INTERSECT ilişkisel küme operatörü yalnızca ortak satırları seçer. EXCEPT (MINUS) küme operatörü yalnızca farklı olan satırları seçer. UNION, INTERSECT ve EXCEPT için union uyumlu ilişkiler gerekir.
* Etkili ve verimli SQL sorguları oluşturmak büyük bir beceri gerektirir. Karmaşık sorguları başarılı bir şekilde oluşturmak için, üzerinde çalıştığınız verileri ve çözülmesi gereken sorunu anlamanız gerekir. Sorgunun formülasyonu ile uğraşırken, sorgu bileşenlerini FROM, WHERE, GROUP BY, HAVING, SELECT ve ORDER BY sırasına

göre oluşturmak

yardımcı olur.

takma ad VE AVG

ARASINDA

Boole cebiri basamaklı sıra dizisi ilişkili alt sorgu COUNT

çapraz birleştirme

DISTINCT EXISTS

SAHIP OLARAK GRUPTAN IS NULL LIKE MAX

MIN NOT

OR ORDER BY

özyinelemeli sorgu öncelik kuralları SELECT

küme yönelimli alt sorgu SUM işlem NEREDE

joker karakter