Alt Sorgular

Alt sorgular, dört SQL veri ifadesinin **(insert, update, delete, select)** tamamında kullanabileceğiniz güçlü bir araçtır. Bu bölümde, alt sorguların verileri filtrelemek, değerler oluşturmak ve geçici veri kümeleri oluşturmak için nasıl kullanılabileceği gösterilecektir.

Alt sorgu nedir?

Alt sorgu, başka bir SQL deyiminde bulunan bir sorgudur. Bir alt sorgu her zaman parantez içine alınır ve genellikle üst sorgu ifadesinden önce yürütülür. Herhangi bir sorgu gibi, bir alt sorgu da şunlardan oluşabilen bir sonuç kümesi döndürür:

• Tek bir sütunlu tek bir satır

• Tek bir sütunlu birden çok satır

• Birden çok sütunu olan birden çok satır

Alt sorgu tarafından üretilen veriler üst sorgu tarafından işlendikten sonra sunucu tarafından bellekten atılır.

```
mysql> SELECT customer_id, first_name, last_name
    -> FROM customer
    -> WHERE customer_id = (SELECT MAX(customer_id) FROM customer);
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         599 | AUSTIN     | CINTRON   |
+-------------+------------+-----------+
1 row in set (0.27 sec)
```

Bu örnekte, alt sorgu, **customer** tablosundaki **customer_id** sütununda bulunan maksimum değeri döndürür ve ardından üst sorgu da bu bilgiyi kullanarak, ilgili müşteriyle ilgili verileri döndürür. Bir alt sorgunun ne yaptığı konusunda kafanız karıştıysa, ne döndürdüğünü görmek için alt sorguyu kendi başına (parantezler olmadan) çalıştırabilirsiniz:

```
mysql> SELECT MAX(customer_id) FROM customer;
+------------------+
| MAX(customer_id) |
+------------------+
|              599 |
+------------------+
1 row in set (0.00 sec)
```

Kaynak: Alan Beaulieu, 2020, Learning SQL, 3rd Edition, O'Reilly.

Bu alt sorgu, eşitlik koşulundaki ifadelerden biri olarak kullanılmasına izin veren tek sütunlu tek bir satır döndürür. Bu durumda, alt sorgunun döndürdüğü değeri alabilir ve aşağıdaki gibi, üst sorgudaki filtre koşulunun sağdaki ifadesinin yerine yazabilirsiniz:

```
mysql> SELECT customer_id, first_name, last_name
    -> FROM customer
    -> WHERE customer_id = 599;
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         599 | AUSTIN     | CINTRON   |
+-------------+------------+-----------+
1 row in set (0.00 sec)
```

Alt sorgu tipleri

İlişkili Olmayan Alt Sorgular

Bölümün başlarındaki örnek, ilişkisiz bir alt sorgudur; tek başına yürütülebilir ve üst sorgudan hiçbir referansı yoktur. Karşılaştığınız çoğu alt sorgu, sık sık ilişkili alt sorguları kullanan **update** veya **delete** ifadeleri yazmadığınız sürece ilişkisiz türden olacaktır. Bu sorgular aynı zamanda tek bir satır ve sütun içeren bir sonuç kümesi döndürür. Bu tür alt sorgu, **skaler alt sorgu** olarak tanımlanır ve karşılaştırma operatörleri **(=, <>, <, >, <=, >=)** kullanılarak bir koşulun her iki tarafında görünebilir.

```
mysql> SELECT city_id, city
    -> FROM city
    -> WHERE country_id <>
    ->  (SELECT country_id FROM country WHERE country = 'India');
+---------+----------------------------+
| city_id | city                       |
+---------+----------------------------+
|       1 | A Corua (La Corua)         |
|       2 | Abha                       |
|       3 | Abu Dhabi                  |
|       4 | Acua                       |
|       5 | Adana                      |
|       6 | Addis Abeba                |
...
|     595 | Zapopan                    |
|     596 | Zaria                      |
|     597 | Zeleznogorsk               |
|     598 | Zhezqazghan                |
|     599 | Zhoushan                   |
|     600 | Ziguinchor                 |
+---------+----------------------------+
540 rows in set (0.02 sec)
```

Bu sorgu, Hindistan'da olmayan tüm şehirleri döndürür. İfadenin son satırında bulunan alt sorgu, Hindistan için **country_id**'yi ve üst sorgu, bu **country_id**'ye sahip olmayan tüm şehirleri döndürür. Bu örnekteki alt sorgu oldukça basit olsa da, alt sorgular daha da kadar karmaşık olabilir ve mevcut tüm sorgu cümleciklerini kullanabilirler.

Bir eşitlik koşulunda bir alt sorgu kullanırsanız ve alt sorgu birden fazla satır döndürürse, bir hata alırsınız. Örneğin, önceki sorguyu alt sorgu Hindistan dışındaki tüm ülkeleri döndürecek şekilde değiştirirseniz, aşağıdaki hatayı alırsınız.

```
mysql> SELECT city_id, city
    -> FROM city
    -> WHERE country_id <>
    ->  (SELECT country_id FROM country WHERE country <> 'India');
ERROR 1242 (21000): Subquery returns more than 1 row
```

Alt sorguyu tek başına çalıştırırsanız, aşağıdaki sonuçları görürsünüz:

```
mysql> SELECT country_id FROM country WHERE country <> 'India';
+------------+
| country_id |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
...
|        106 |
|        107 |
|        108 |
|        109 |
+------------+
108 rows in set (0.00 sec)
```

Çok Satırlı, Tek Sütunlu Alt Sorgular

Alt sorgunuz birden fazla satır döndürürse, önceki örnekte gösterildiği gibi bunu bir eşitlik koşulunun bir tarafında kullanamazsınız. Ancak, bu tür alt sorgularla koşullar oluşturmak için kullanabileceğiniz dört ek işleç vardır.

Tek bir değeri bir dizi değere eşitleyemeseniz de, bir dizi değer içinde tek bir değerin bulunup bulunmadığını kontrol edebilirsiniz. Sonraki örnek, bir alt sorgu kullanmasa da, bir değerler kümesi içinde bir değer aramak için **in** operatörünü kullanan bir koşulun nasıl oluşturulacağını gösterir:

```
mysql> SELECT country_id
    -> FROM country
    -> WHERE country IN ('Canada','Mexico');
+------------+
| country_id |
+------------+
|         20 |
|         60 |
+------------+
2 rows in set (0.00 sec)
```

Koşulun sol tarafındaki ifade ülke sütunu, koşulun sağ tarafındaki ifade ise bir dizi **string**'tir(dize). **in** operatörü, dizelerden herhangi birinin ülke sütununda bulunup bulunmadığını kontrol eder; eğer öyleyse, koşul karşılanır ve satır sonuç kümesine eklenir. Aşağıdaki gibi iki eşitlik koşulu kullanarak aynı sonuçları elde edebilirsiniz:

```
mysql> SELECT country_id
    -> FROM country
    -> WHERE country = 'Canada' OR country = 'Mexico';
+------------+
| country_id |
+------------+
|         20 |
|         60 |
+------------+
2 rows in set (0.00 sec)
```

Küme yalnızca iki ifade içerdiğinde bu yaklaşım makul görünse de, küme düzinelerce (veya yüzlerce, binlerce, vb.) değer içeriyorsa, **in** operatörünü kullanan tek bir koşulun neden tercih edildiğini anlamak kolaydır.

Bir koşulun bir tarafında kullanmak için ara sıra bir dizi dize, tarih veya sayı oluşturacak olsanız da, bir veya daha fazla satır döndüren bir alt sorgu kullanarak kümeyi oluşturma olasılığınız daha yüksektir. Aşağıdaki sorgu, Kanada veya Meksika'daki tüm şehirleri döndürmek için filtre koşulunun sağ tarafında bir alt sorgu bulunan **in** operatörünü kullanır.

```
mysql> SELECT city_id, city
    -> FROM city
    -> WHERE country_id IN
    ->  (SELECT country_id
    ->   FROM country
    ->   WHERE country IN ('Canada','Mexico'));
```

```
+---------+---------------------------+
| city_id | city                      |
+---------+---------------------------+
|     179 | Gatineau                  |
|     196 | Halifax                   |
|     300 | Lethbridge                |
|     313 | London                    |
|     383 | Oshawa                    |
|     430 | Richmond Hill             |
|     565 | Vancouver                 |
...
|     452 | San Juan Bautista Tuxtepec |
|     541 | Torren                    |
|     556 | Uruapan                   |
|     563 | Valle de Santiago         |
|     595 | Zapopan                   |
+---------+---------------------------+
37 rows in set (0.00 sec)
```

Bir değer kümesi içinde bir değerin olup olmadığını görmenin yanı sıra, **not** operatörünü kullanarak tersini kontrol edebilirsiniz. **in** yerine **not in** kullanan önceki sorgunun başka bir sürümü:

```
mysql> SELECT city_id, city
    -> FROM city
    -> WHERE country_id NOT IN
    ->  (SELECT country_id
    ->   FROM country
    ->   WHERE country IN ('Canada','Mexico'));
+---------+---------------------------+
| city_id | city                      |
+---------+---------------------------+
|       1 | A Corua (La Corua)        |
|       2 | Abha                      |
|       3 | Abu Dhabi                 |
|       5 | Adana                     |
|       6 | Addis Abeba               |
...
|     596 | Zaria                     |
|     597 | Zeleznogorsk              |
|     598 | Zhezqazghan               |
|     599 | Zhoushan                  |
|     600 | Ziguinchor                |
+---------+---------------------------+
563 rows in set (0.00 sec)
```

in operatörü, bir ifade kümesinde bir ifadenin bulunup bulunmadığını görmek için kullanılırken, **all** operatörü, tek bir değer ile bir kümedeki her değer arasında karşılaştırma yapmanıza olanak tanır. Böyle bir koşul oluşturmak için, **all** operatörüyle birlikte karşılaştırma operatörlerinden **(=, <>, <, >, vb.)** birini

kullanmanız gerekecektir. Örneğin, bir sonraki sorgu, hiç ücretsiz film kiralama hizmeti almamış tüm müşterileri bulur:

```
mysql> SELECT first_name, last_name
    -> FROM customer
    -> WHERE customer_id <> ALL
    ->  (SELECT customer_id
    ->   FROM payment
    ->   WHERE amount = 0);
+-------------+--------------+
| first_name  | last_name    |
+-------------+--------------+
| MARY        | SMITH        |
| PATRICIA    | JOHNSON      |
| LINDA       | WILLIAMS     |
| BARBARA     | JONES        |
...
| EDUARDO     | HIATT        |
| TERRENCE    | GUNDERSON    |
| ENRIQUE     | FORSYTHE     |
| FREDDIE     | DUGGAN       |
| WADE        | DELVALLE     |
| AUSTIN      | CINTRON      |
+-------------+--------------+
576 rows in set (0.01 sec)
```

Alt sorgu, bir film kiralaması için 0 dolar ödeyen müşteriler için **customer_id** kümesini döndürür ve üst sorgu, **customer_id'si** alt sorgu tarafından döndürülen kümede olmayan tüm müşterilerin adlarını döndürür. Bu sorgu, **not in** operatörünü kullanan sonraki örnekle aynı sonuçları üretir.

```
SELECT first_name, last_name
FROM customer
WHERE customer_id NOT IN
 (SELECT customer_id
  FROM payment
  WHERE amount = 0)
```

Bir değeri bir değer kümesiyle karşılaştırmak için **not in** veya **<> all** kullanırken, sunucu sol taraftaki değeri eşitlediğinden, değerler kümesinin boş bir değer içermediğinden emin olmalısınız. İfadenin kümenin her bir üyesine aktarılması ve bir değeri boş değere eşitleme girişimi boş bir sonuç verir. Bu nedenle, aşağıdaki sorgu boş bir küme döndürür:

```
mysql> SELECT first_name, last_name
    -> FROM customer
    -> WHERE customer_id NOT IN (122, 452, NULL);
Empty set (0.00 sec)
```

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> GROUP BY customer_id
    -> HAVING count(*) > ALL
    ->  (SELECT count(*)
    ->   FROM rental r
    ->     INNER JOIN customer c
    ->     ON r.customer_id = c.customer_id
    ->     INNER JOIN address a
    ->     ON c.address_id = a.address_id
    ->     INNER JOIN city ct
    ->     ON a.city_id = ct.city_id
    ->     INNER JOIN country co
    ->     ON ct.country_id = co.country_id
    ->   WHERE co.country IN ('United States','Mexico','Canada')
    ->   GROUP BY r.customer_id
    ->   );
+-------------+----------+
| customer_id | count(*) |
+-------------+----------+
|         148 |       46 |
+-------------+----------+
1 row in set (0.01 sec)
```

Bu örnekteki alt sorgu, Kuzey Amerika'daki tüm müşteriler için toplam film kiralama sayısını döndürür ve üst sorgu, toplam film kiralama sayısı Kuzey Amerika müşterilerinden herhangi birini aşan tüm müşterileri döndürür.

all operatörü gibi, **any** operatörü de bir değerin bir dizi değerin üyeleriyle karşılaştırılmasına izin verir; Ancak **all** operatöründen farklı olarak, **any** operatörünün kullanıldığı bir koşul, tek bir karşılaştırmanın uygun olduğu anda **true** olarak değerlendirilir. Bu örnek, toplam film kiralama ödemeleri Bolivya, Paraguay veya Şili'deki tüm müşteriler için toplam ödemeleri aşan tüm müşterileri bulacaktır:

```
mysql> SELECT customer_id, sum(amount)
    -> FROM payment
    -> GROUP BY customer_id
    -> HAVING sum(amount) > ANY
    ->  (SELECT sum(p.amount)
    ->   FROM payment p
    ->     INNER JOIN customer c
    ->     ON p.customer_id = c.customer_id
    ->     INNER JOIN address a
    ->     ON c.address_id = a.address_id
    ->     INNER JOIN city ct
    ->     ON a.city_id = ct.city_id
    ->     INNER JOIN country co
    ->     ON ct.country_id = co.country_id
    ->   WHERE co.country IN ('Bolivia','Paraguay','Chile')
    ->   GROUP BY co.country
    ->  );
```

```
+-------------+-------------+
| customer_id | sum(amount) |
+-------------+-------------+
|         137 |      194.61 |
|         144 |      195.58 |
|         148 |      216.54 |
|         178 |      194.61 |
|         459 |      186.62 |
|         526 |      221.55 |
+-------------+-------------+
6 rows in set (0.03 sec)
```

Çok Sütunlu Alt Sorgular

Şimdiye kadar, bu bölümdeki alt sorgu örnekleri tek bir sütun ve bir veya daha fazla satır döndürdü. Ancak belirli durumlarda, iki veya daha fazla sütun döndüren alt sorgular kullanabilirsiniz.

```
mysql> SELECT fa.actor_id, fa.film_id
    -> FROM film_actor fa
    -> WHERE fa.actor_id IN
    ->  (SELECT actor_id FROM actor WHERE last_name = 'MONROE')
    ->   AND fa.film_id IN
    ->  (SELECT film_id FROM film WHERE rating = 'PG');
+----------+---------+
| actor_id | film_id |
+----------+---------+
|      120 |      63 |
|      120 |     144 |
|      120 |     414 |
|      120 |     590 |
|      120 |     715 |
|      120 |     894 |
|      178 |     164 |
|      178 |     194 |
|      178 |     273 |
|      178 |     311 |
|      178 |     983 |
+----------+---------+
11 rows in set (0.00 sec)
```

Bu sorgu, Monroe soyadına sahip tüm aktörleri ve PG olarak derecelendirilen tüm filmleri tanımlamak için iki alt sorgu kullanır ve daha sonra üst sorgu, Monroe adlı bir aktörün bir PG filminde göründüğü tüm durumları almak için bu bilgiyi kullanır. Ancak, iki tek sütunlu alt sorguyu tek bir, çok sütunlu alt sorguda birleştirebilir ve sonuçları **film_actor** tablosundaki iki sütunla karşılaştırabilirsiniz. Bunu yapmak için, filtre koşulunuz **film_actor** tablosundaki her iki sütunu parantez içinde ve alt sorgu tarafından döndürülen sırayla, aşağıdaki gibi adlandırmalıdır:

```
mysql> SELECT actor_id, film_id
    -> FROM film_actor
    -> WHERE (actor_id, film_id) IN
    ->  (SELECT a.actor_id, f.film_id
    ->   FROM actor a
    ->      CROSS JOIN film f
    ->   WHERE a.last_name = 'MONROE'
    ->   AND f.rating = 'PG');
+----------+---------+
| actor_id | film_id |
+----------+---------+
|      120 |      63 |
|      120 |     144 |
|      120 |     414 |
|      120 |     590 |
|      120 |     715 |
|      120 |     894 |
|      178 |     164 |
|      178 |     194 |
|      178 |     273 |
|      178 |     311 |
|      178 |     983 |
+----------+---------+
11 rows in set (0.00 sec)
```

Sorgunun bu sürümü, önceki örnekle aynı işlevi yerine getirir, ancak her biri tek bir sütun döndüren iki alt sorgu yerine iki sütun döndüren tek bir alt sorgu kullanılır. Bu sürümdeki alt sorgu, çapraz birleştirme (cross join) adı verilen bir birleştirme türünü kullanır.

İlişkili Alt Sorgular

Şimdiye kadar gösterilen tüm alt sorgular, üst sorgulardan bağımsızdır, yani bunları kendi başınıza yürütebilir ve sonuçları inceleyebilirsiniz. Öte yandan, ilişkili bir alt sorgu, bir veya daha fazla sütuna referans veren bir üst sorgu ifadesine bağlıdır. İlişkili olmayan bir alt sorgunun aksine, ilişkili bir alt sorgu, üst sorgunun yürütülmesinden önce bir kez yürütülmez; bunun yerine, ilişkili alt sorgu her aday satır (nihai sonuçlara dahil edilebilecek satırlar) için bir kez yürütülür. Örneğin, aşağıdaki sorgu, her müşteri için film kiralama sayısını saymak için ilişkili bir alt sorgu kullanır ve ardından, üst sorgu tam olarak 20 film kiralamış olan müşterileri alır:

```
mysql> SELECT c.first_name, c.last_name
    -> FROM customer c
    -> WHERE 20 =
    ->  (SELECT count(*) FROM rental r
    ->   WHERE r.customer_id = c.customer_id);
```

```
+------------+-------------+
| first_name | last_name   |
+------------+-------------+
| LAUREN     | HUDSON      |
| JEANETTE   | GREENE      |
| TARA       | RYAN        |
| WILMA      | RICHARDS    |
| JO         | FOWLER      |
| KAY        | CALDWELL    |
| DANIEL     | CABRAL      |
| ANTHONY    | SCHWAB      |
| TERRY      | GRISSOM     |
| LUIS       | YANEZ       |
| HERBERT    | KRUGER      |
| OSCAR      | AQUINO      |
| RAUL       | FORTIER     |
| NELSON     | CHRISTENSON |
| ALFREDO    | MCADAMS     |
+------------+-------------+
15 rows in set (0.01 sec)
```

Alt sorgunun en sonundaki **c.customer_id** referansı, alt sorguyu üst sorgu ile ilişkilendiren ifadedir; üst sorgu, alt sorgunun yürütülmesi için **c.customer_id** için değerler sağlamalıdır. Bu durumda, üst sorgu müşteri tablosundan 599 satırın tümünü alır ve her müşteri için uygun **customer_id** ileterek alt sorguyu her müşteri için bir kez yürütür. Alt sorgu 20 değerini döndürürse, filtre koşulu karşılanır ve satır sonuç kümesine eklenir.

İlişkili alt sorgu, üst sorgunun her satırı için bir kez yürütüleceğinden, üst sorgu çok sayıda satır döndürürse, ilişkili alt sorguların kullanımı performans sorunlarına neden olabilir.

Eşitlik koşullarının yanı sıra, burada gösterilen aralık koşulu gibi diğer koşul türlerinde ilişkili alt sorguları kullanabilirsiniz:

```
mysql> SELECT c.first_name, c.last_name
    -> FROM customer c
    -> WHERE
    ->  (SELECT sum(p.amount) FROM payment p
    ->   WHERE p.customer_id = c.customer_id)
    ->   BETWEEN 180 AND 240;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| RHONDA     | KENNEDY   |
| CLARA      | SHAW      |
| ELEANOR    | HUNT      |
| MARION     | SNYDER    |
| TOMMY      | COLLAZO   |
| KARL       | SEAL      |
+------------+-----------+
6 rows in set (0.03 sec)
```

Kaynak: Alan Beaulieu, 2020, Learning SQL, 3rd Edition, O'Reilly.

Önceki sorgudaki bu değişiklik, tüm film kiralamaları için toplam ödemeleri 180 ila 240 dolar arasında olan tüm müşterileri bulur. İlişkili alt sorgu 599 kez yürütülür (her müşteri satırı için bir kez) ve alt sorgunun her yürütmesi, verilen müşteri için toplam hesap bakiyesini döndürür.

Önceki sorgudaki bir diğer ince fark, alt sorgunun koşulun sol tarafında olmasıdır, bu biraz tuhaf görünebilir ancak tamamen geçerli bir yazımdır.

Eşitlik ve aralık koşullarında kullanılan ilişkili alt sorguları sık sık göreceksiniz, ancak ilişkili alt sorguları kullanan koşulları oluşturmak için kullanılan en yaygın operatör, **exists** operatördür. Miktar dikkate alınmaksızın bir ilişkinin var olduğunu belirlemek istediğinizde, **exists** operatörünü kullanırsınız; örneğin, aşağıdaki sorgu, kaç film kiralandığına bakılmaksızın 25 Mayıs 2005'ten önce en az bir film kiralamış tüm müşterileri bulur:

```
mysql> SELECT c.first_name, c.last_name
    -> FROM customer c
    -> WHERE EXISTS
    ->  (SELECT 1 FROM rental r
    ->   WHERE r.customer_id = c.customer_id
    ->     AND date(r.rental_date) < '2005-05-25');
+------------+-------------+
| first_name | last_name   |
+------------+-------------+
| CHARLOTTE  | HUNTER      |
| DELORES    | HANSEN      |
| MINNIE     | ROMERO      |
| CASSANDRA  | WALTERS     |
| ANDREW     | PURDY       |
| MANUEL     | MURRELL     |
| TOMMY      | COLLAZO     |
| NELSON     | CHRISTENSON |
+------------+-------------+
8 rows in set (0.03 sec)
```

exists operatörünü kullanarak, alt sorgunuz sıfır, bir veya daha fazla satır döndürebilir ve koşul, alt sorgunun bir veya daha fazla satır döndürüp döndürmediğini kontrol eder. Alt sorgunun **select** cümleciğine bakarsanız, onun tek bir değişmezden **(1)** oluştuğunu göreceksiniz; üst sorgudaki koşulun yalnızca kaç satır döndürüldüğünü bilmesi gerektiğinden, alt sorgunun döndürdüğü gerçek veriler önemsizdir.

Ayrıca, aşağıda gösterildiği gibi, satır döndürmeyen alt sorguları kontrol etmek için **not exists** kullanabilirsiniz:

Bu sorgu, R dereceli bir filmde hiç rol almamış tüm oyuncuları bulur.

```
mysql> SELECT a.first_name, a.last_name
    -> FROM actor a
    -> WHERE NOT EXISTS
    ->  (SELECT 1
    ->   FROM film_actor fa
    ->     INNER JOIN film f ON f.film_id = fa.film_id
    ->   WHERE fa.actor_id = a.actor_id
    ->     AND f.rating = 'R');
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| JANE       | JACKMAN   |
+------------+-----------+
1 row in set (0.00 sec)
```

İlişkili Alt Sorguları Kullanarak Veri Manipülasyonu

Alt sorgular, **update, delete ve insert** ifadelerinde de yoğun olarak kullanılır, ilişkili alt sorgular **update, delete** ifadelerinde sıklıkla görünür. **Customer** tablosundaki **last_update** sütununu değiştirmek için kullanılan ilişkili bir alt sorgu örneği aşağıda verilmiştir:

```
UPDATE customer c
SET c.last_update =
 (SELECT max(r.rental_date) FROM rental r
  WHERE r.customer_id = c.customer_id);
```

Bu ifade, **rental** tablosundaki her müşteri için en son kiralama tarihini bularak müşteri tablosundaki her satırı değiştirir (**where** ibaresi olmadığı için). Her müşterinin en az bir film kiralamasını beklemek makul görünse de, **last_update** sütununu güncellemeye çalışmadan önce kontrol etmek en iyisi olacaktır; Aksi takdirde, alt sorgu hiçbir satır döndürmeyeceğinden, sütun **null** olarak ayarlanır. **update** ifadesinin başka bir versiyonu, bu sefer ikinci bir ilişkili alt sorgu ile bir **where** yan tümcesi kullanıyor:

```
UPDATE customer c
SET c.last_update =
 (SELECT max(r.rental_date) FROM rental r
  WHERE r.customer_id = c.customer_id)
WHERE EXISTS
 (SELECT 1 FROM rental r
  WHERE r.customer_id = c.customer_id);
```

İlişkili iki alt sorgu, **select** yan tümceleri dışında aynıdır. Ancak **set** yan tümcesindeki alt sorgu, yalnızca **update** ifadesinin **where** yan tümcesindeki koşul doğru olarak

değerlendirilirse (yani müşteri için en az bir kiralama bulunursa) yürütülür, böylece **last_update** sütunundaki verilerin üzerine yazılmasını önler.

İlişkili alt sorguların kullanımı, **delete** ifadelerinde de yaygındır. Örneğin, her ayın sonunda gereksiz verileri kaldıran bir veri bakım komut dosyası çalıştırabilirsiniz. Senaryo, müşteri tablosundan geçen yıl içinde hiç film kiralama yapılmamış satırları kaldıran aşağıdaki ifadeyi içerebilir:

```
DELETE FROM customer
WHERE 365 < ALL
 (SELECT datediff(now(), r.rental_date) days_since_last_rental
  FROM rental r
  WHERE r.customer_id = customer.customer_id);
```

Alt Sorgular Ne Zaman Kullanılır?

Veri Kaynakları Olarak Alt Sorgular

Bir alt sorgu, satırlar ve veri sütunları içeren bir sonuç kümesi oluşturduğundan, tablolarla birlikte **from** yan tümcenize alt sorguları dahil etmek tamamen geçerli bir yazımdır. İlk bakışta, pek pratik değeri olmayan ilginç bir özellik gibi görünse de, tabloların yanında alt sorguları kullanmak, sorgu yazarken kullanılabilecek en güçlü araçlardan biridir.

```
mysql> SELECT c.first_name, c.last_name,
    ->   pymnt.num_rentals, pymnt.tot_payments
    -> FROM customer c
    ->   INNER JOIN
    ->    (SELECT customer_id,
    ->       count(*) num_rentals, sum(amount) tot_payments
    ->     FROM payment
    ->     GROUP BY customer_id
    ->    ) pymnt
    ->   ON c.customer_id = pymnt.customer_id;
+------------+-------------+------------+-------------+
| first_name | last_name   | num_rentals | tot_payments |
+------------+-------------+------------+-------------+
| MARY       | SMITH       |          32 |       118.68 |
| PATRICIA   | JOHNSON     |          27 |       128.73 |
| LINDA      | WILLIAMS    |          26 |       135.74 |
| BARBARA    | JONES       |          22 |        81.78 |
| ELIZABETH  | BROWN       |          38 |       144.62 |
...
| TERRENCE   | GUNDERSON   |          30 |       117.70 |
| ENRIQUE    | FORSYTHE    |          28 |        96.72 |
| FREDDIE    | DUGGAN      |          25 |        99.75 |
| WADE       | DELVALLE    |          22 |        83.78 |
| AUSTIN     | CINTRON     |          19 |        83.81 |
+------------+-------------+------------+-------------+
599 rows in set (0.03 sec)
```

Bu örnekte, bir alt sorgu, film kiralama sayısı ve toplam ödemelerle birlikte **customer_id** sütununun bir listesini oluşturur.

Kaynak: Alan Beaulieu, 2020, Learning SQL, 3rd Edition, O'Reilly.

```
mysql> SELECT customer_id, count(*) num_rentals, sum(amount) tot_payments
    -> FROM payment
    -> GROUP BY customer_id;
+-------------+-------------+--------------+
| customer_id | num_rentals | tot_payments |
+-------------+-------------+--------------+
|           1 |          32 |       118.68 |
|           2 |          27 |       128.73 |
|           3 |          26 |       135.74 |
|           4 |          22 |        81.78 |
...
|         596 |          28 |        96.72 |
|         597 |          25 |        99.75 |
|         598 |          22 |        83.78 |
|         599 |          19 |        83.81 |
+-------------+-------------+--------------+
599 rows in set (0.03 sec)
```

Alt sorguya **pymnt** adı verilir ve müşteri tablosuna **customer_id** sütunu aracılığıyla birleştirilir. Üst sorgu daha sonra müşteri tablosundan müşterinin adını, **pymnt** alt sorgusundaki özet sütunlarıyla birlikte alır.

from yan tümcesinde kullanılan alt sorgular ilişkisiz olmalıdır önce alt sorgu yürütülür ve veriler, üst sorgu yürütmeyi bitirene kadar bellekte tutulur. Alt sorgular, sorgu yazarken büyük esneklik sunar, çünkü istediğiniz verinin hemen hemen her görünümünü oluşturmak için mevcut tabloları gelişmiş versiyonlarını sunar ve ardından sonuçları diğer tablolara veya alt sorgularla birleştirebilir.

Veri Üretimi

Var olan verileri özetlemek için alt sorguları kullanmanın yanı sıra, veritabanınızda hiçbir biçimde var olmayan verileri oluşturmak için alt sorguları kullanabilirsiniz. Örneğin, müşterilerinizi film kiralamalarına harcanan para miktarına göre gruplamak isteyebilirsiniz, ancak veritabanınızda saklanmayan grup tanımlarını kullanmak isteyebilirsiniz. Müşterilerinizi Tabloda gösterilen gruplara ayırmak istediğinizi varsayalım.

Group name	Lower limit	Upper limit
Small Fry	0	$74.99
Average Joes	$75	$149.99
Heavy Hitters	$150	$9,999,999.99

Bu grupları tek bir sorgu içinde oluşturmak için bu üç grubu tanımlamaya ihtiyacınız olacak. İlk adım, grup tanımlarını oluşturan bir sorgu tanımlamaktır:

```
mysql> SELECT 'Small Fry' name, 0 low_limit, 74.99 high_limit
    -> UNION ALL
    -> SELECT 'Average Joes' name, 75 low_limit, 149.99 high_limit
    -> UNION ALL
    -> SELECT 'Heavy Hitters' name, 150 low_limit, 9999999.99 high_limit;
+--------------+-----------+------------+
| name         | low_limit | high_limit |
+--------------+-----------+------------+
| Small Fry    |         0 |      74.99 |
| Average Joes |        75 |     149.99 |
| Heavy Hitters |       150 | 9999999.99 |
+--------------+-----------+------------+
3 rows in set (0.00 sec)
```

Üç ayrı sorgunun sonuçlarını tek bir sonuç kümesinde birleştirmek için **union all** operatörü kullanıldı. Her sorgu üç sabit değer alır ve üç sorgudan elde edilen sonuçlar, üç satır ve üç sütunlu bir sonuç kümesi oluşturmak için bir araya getirilir. Artık istediğiniz grupları oluşturmak için bir sorgunuz var ve müşteri gruplarınızı oluşturmak için bunu başka bir sorgunun **from** yan tümcesine yerleştirebilirsiniz:

```
mysql> SELECT pymnt_grps.name, count(*) num_customers
    -> FROM
    ->  (SELECT customer_id,
    ->     count(*) num_rentals, sum(amount) tot_payments
    ->   FROM payment
    ->   GROUP BY customer_id
    ->  ) pymnt
    ->   INNER JOIN
    ->  (SELECT 'Small Fry' name, 0 low_limit, 74.99 high_limit
    ->   UNION ALL
    ->   SELECT 'Average Joes' name, 75 low_limit, 149.99 high_limit
    ->   UNION ALL
    ->   SELECT 'Heavy Hitters' name, 150 low_limit, 9999999.99 high_limit
    ->  ) pymnt_grps
    ->   ON pymnt.tot_payments
    ->     BETWEEN pymnt_grps.low_limit AND pymnt_grps.high_limit
    -> GROUP BY pymnt_grps.name;
+--------------+---------------+
| name         | num_customers |
+--------------+---------------+
| Average Joes |           515 |
| Heavy Hitters |            46 |
| Small Fry    |            38 |
+--------------+---------------+
3 rows in set (0.03 sec)
```

from yan tümcesi iki alt sorgu içerir; **pymnt** adlı ilk alt sorgu, her müşteri için toplam film kiralama sayısını ve toplam ödemeleri döndürürken, **pymnt_grps** adlı ikinci alt sorgu, üç müşteri grubunu oluşturur. İki alt sorgu, her müşterinin üç gruptan

hangisine ait olduğu bulunarak birleştirilir ve ardından her gruptaki müşteri sayısını saymak için satırlar grup adına göre gruplandırılır.

Elbette, bir alt sorgu kullanmak yerine grup tanımlarını tutmak için kalıcı (veya geçici) bir tablo oluşturmak tercihe bağlıdır. Bu yaklaşımı kullanarak, bir süre sonra veritabanınızın küçük özel amaçlı tablolarla dolup taştığını görebilir ve bunların çoğunun neden oluşturulduğunu harlamayabilirsiniz. Ancak alt sorguları kullanarak, yalnızca yeni verileri depolamak için bir iş ihtiyacı olduğunda tabloların bir veritabanına eklendiği bir ilkeye bağlı kalabilirsiniz.

Görev odaklı alt sorgular

Her müşterinin adını, şehri, toplam kiralama sayısını ve toplam ödeme tutarını gösteren bir rapor oluşturmak istediğinizi varsayalım. Bunu, ödeme, müşteri, adres ve şehir tablolarına bakarak ve ardından müşterinin adı ve soyadına göre gruplandırarak başarabilirsiniz:

```
mysql> SELECT c.first_name, c.last_name, ct.city,
    ->   sum(p.amount) tot_payments, count(*) tot_rentals
    -> FROM payment p
    ->   INNER JOIN customer c
    ->   ON p.customer_id = c.customer_id
    ->   INNER JOIN address a
    ->   ON c.address_id = a.address_id
    ->   INNER JOIN city ct
    ->   ON a.city_id = ct.city_id
    -> GROUP BY c.first_name, c.last_name, ct.city;
+-------------+------------+-----------------+--------------+-------------+
| first_name  | last_name  | city            | tot_payments | tot_rentals |
+-------------+------------+-----------------+--------------+-------------+
| MARY        | SMITH      | Sasebo          |       118.68 |          32 |
| PATRICIA    | JOHNSON    | San Bernardino  |       128.73 |          27 |
| LINDA       | WILLIAMS   | Athenai         |       135.74 |          26 |
| BARBARA     | JONES      | Myingyan        |        81.78 |          22 |
...
| TERRENCE    | GUNDERSON  | Jinzhou         |       117.70 |          30 |
| ENRIQUE     | FORSYTHE   | Patras          |        96.72 |          28 |
| FREDDIE     | DUGGAN     | Sullana         |        99.75 |          25 |
| WADE        | DELVALLE   | Lausanne        |        83.78 |          22 |
| AUSTIN      | CINTRON    | Tieli           |        83.81 |          19 |
+-------------+------------+-----------------+--------------+-------------+
599 rows in set (0.06 sec)
```

Bu sorgu istenen verileri döndürür, ancak sorguya yakından bakarsanız, müşteri, adres ve şehir tablolarının yalnızca görüntüleme amacıyla gerekli olduğunu ve ödeme tablosunun gruplamaları oluşturmak için gereken sütunlara sahip olduğunu göreceksiniz. Bu nedenle, grupları oluşturma görevini bir alt sorguya devredebilir ve

ardından istenen sonucu elde etmek için diğer üç tabloyu alt sorgu tarafından oluşturulan tabloyla birleştirebilirsiniz.

```
mysql> SELECT c.first_name, c.last_name,
    ->   ct.city,
    ->   pymnt.tot_payments, pymnt.tot_rentals
    -> FROM
    ->  (SELECT customer_id,
    ->     count(*) tot_rentals, sum(amount) tot_payments
    ->   FROM payment
    ->   GROUP BY customer_id
    ->  ) pymnt
    ->   INNER JOIN customer c
    ->   ON pymnt.customer_id = c.customer_id
    ->   INNER JOIN address a
    ->   ON c.address_id = a.address_id
    ->   INNER JOIN city ct
    ->   ON a.city_id = ct.city_id;
+-------------+------------+-----------------+--------------+-------------+
| first_name  | last_name  | city            | tot_payments | tot_rentals |
+-------------+------------+-----------------+--------------+-------------+
| MARY        | SMITH      | Sasebo          |       118.68 |          32 |
| PATRICIA    | JOHNSON    | San Bernardino  |       128.73 |          27 |
| LINDA       | WILLIAMS   | Athenai         |       135.74 |          26 |
| BARBARA     | JONES      | Myingyan        |        81.78 |          22 |
...
| TERRENCE    | GUNDERSON  | Jinzhou         |       117.70 |          30 |
| ENRIQUE     | FORSYTHE   | Patras          |        96.72 |          28 |
| FREDDIE     | DUGGAN     | Sullana         |        99.75 |          25 |
| WADE        | DELVALLE   | Lausanne        |        83.78 |          22 |
| AUSTIN      | CINTRON    | Tieli           |        83.81 |          19 |
+-------------+------------+-----------------+--------------+-------------+
599 rows in set (0.06 sec)
```

Bu sürüm daha hızlı çalışabilir, çünkü gruplandırma birden çok uzun dize sütunu **(customer.first_name, customer.last_name, city.city)** yerine tek bir sayısal sütunda **(customer_id)** yapılıyor.