# Gruplama

Veriler genellikle herhangi bir veritabanı kullanıcısının ihtiyaç duyduğu en düşük ayrıntı düzeyinde depolanır; Muhasebede bireysel müşteri işlemlerine bakılması gerekiyorsa, veritabanında bireysel işlemleri saklayan bir tablo olması gerekir. Ancak bu, tüm kullanıcıların verilerinin veritabanında olabilecek tüm ayrıntıları ile depolanması gerektiği anlamına gelmez. Bu bölümün odak noktası, kullanıcıların veri tabanında depolanandan daha yüksek bir ayrıntı düzeyinde etkileşime girmesine izin vermek için verilerin nasıl gruplanabileceği ve kümelenebileceği üzerinedir.

Bazen verilerinizde, aradığınız sonuçları oluşturabilmeniz için veritabanı sunucusunun verileri olgunlaştırmasını gerektirecek eğilimleri bulmak isteyeceksiniz. Örneğin, en iyi müşterilerinize ücretsiz kiralama kuponları göndermekten sorumlu olduğunuzu varsayalım. Ham verilere bakmak için basit bir sorgu düzenleyebilirsiniz.

```
mysql> SELECT customer_id FROM rental;
+-------------+
| customer_id |
+-------------+
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
...
|         599 |
|         599 |
|         599 |
|         599 |
|         599 |
|         599 |
+-------------+
16044 rows in set (0.01 sec)
```

16.000'den fazla kiralama kaydına yayılan 599 müşteri ile, ham verilere bakarak hangi müşterilerin en çok film kiraladığını belirlemek mümkün değil. Bunun yerine, **group by** cümlesini kullanarak veritabanı sunucusundan verileri sizin için gruplamasını isteyebilirsiniz. İşte aynı sorgu, ancak kiralama verilerini müşteri kimliğine göre gruplandırmak için yan tümceler kullanılıyor:

```
mysql> SELECT customer_id
    -> FROM rental
    -> GROUP BY customer_id;
+-------------+
| customer_id |
+-------------+
|           1 |
|           2 |
|           3 |
|           4 |
|           5 |
|           6 |
...
|         594 |
|         595 |
|         596 |
|         597 |
|         598 |
|         599 |
+-------------+
599 rows in set (0.00 sec)
```

Sonuç kümesi, tam 16.044 satır yerine 599 satır ile sonuçlanan, **customer_id** sütunundaki her farklı değer için bir satır içerir. Sonuç setinin daha küçük olmasının nedeni, bazı müşterilerin birden fazla film kiralamasıdır. Her müşterinin kaç film kiraladığını görmek için, her gruptaki satır sayısını saymak için **select** yan tümcesindeki bir kümeleme(aggregate) metodunu kullanabilirsiniz:

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> GROUP BY customer_id;
+-------------+----------+
| customer_id | count(*) |
+-------------+----------+
|           1 |       32 |
|           2 |       27 |
|           3 |       26 |
|           4 |       22 |
|           5 |       38 |
|           6 |       28 |
|         594 |       27 |
|         595 |       30 |
|         596 |       28 |
|         597 |       25 |
|         598 |       22 |
|         599 |       19 |
+-------------+----------+
599 rows in set (0.01 sec)
```

Kümeleme metodu **count()** her gruptaki satır sayısını sayar ve yıldız işareti sunucuya gruptaki her şeyi saymasını söyler. Sonuçlara bakıldığında, costumer_id değeri 1 ola müşterinin 32 film ve costumer_id 597 olan müşteri tarafından 25 filmin kiralandığını görebilirsiniz. Hangi müşterilerin en çok film kiraladığını belirlemek için, sıralama yapabilirsiniz:

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> GROUP BY customer_id
    -> ORDER BY 2 DESC;
+-------------+----------+
| customer_id | count(*) |
+-------------+----------+
|         148 |       46 |
|         526 |       45 |
|         236 |       42 |
|         144 |       42 |
|          75 |       41 |
...
|         248 |       15 |
|         110 |       14 |
|         281 |       14 |
|          61 |       14 |
|         318 |       12 |
+-------------+----------+
599 rows in set (0.01 sec)
```

Artık sonuçlar sıralandığına göre, en çok filmi costumer_id 148 olan müşterinin (46), en az filmi ise costumer_id 318' olan müşterinin (12) kiraladığını kolayca görebilirsiniz.

Verileri gruplandırırken, ham veriler yerine veri gruplarına dayalı olarak sonuç kümenizden istenmeyen verileri filtrelemeniz gerekebilir. **group by** yan tümcesi, **where** yan tümcesi değerlendirildikten sonra çalıştığından, bu amaçla **where** yan tümcenize filtre koşulları ekleyemezsiniz. Örneğin, 40'tan az film kiralamış olan tüm müşterileri filtrelemek aşağıdaki sorgu işe yaramayacaktır:

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> WHERE count(*) >= 40
    -> GROUP BY customer_id;
ERROR 1111 (HY000): Invalid use of group function
```

Gruplar, **where** yan tümcesinin değerlendirildiği sırada henüz oluşturulmadığından, **where** yan tümcenizdeki **count(*)** kümeleme metoduna başvuramazsınız. Bunun yerine, **having** yan tümcesine grup filtresi koşullarınızı koymalısınız.

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> GROUP BY customer_id
    -> HAVING count(*) >= 40;
+-------------+----------+
| customer_id | count(*) |
+-------------+----------+
|          75 |       41 |
|         144 |       42 |
|         148 |       46 |
|         197 |       40 |
|         236 |       42 |
|         469 |       40 |
|         526 |       45 |
+-------------+----------+
7 rows in set (0.01 sec)
```

40'tan az üye içeren gruplar **having** aracılığıyla filtrelendiğinden, sonuç kümesi artık yalnızca 40 veya daha fazla film kiralamış olan müşterileri içermektedir.

# Fonksiyonlar

Film kiralama ödemeleriyle ilgili verileri analiz etmek için tüm genel metotoalrı kullanan bir sorgu:

```
mysql> SELECT MAX(amount) max_amt,
    ->   MIN(amount) min_amt,
    ->   AVG(amount) avg_amt,
    ->   SUM(amount) tot_amt,
    ->   COUNT(*) num_payments
    -> FROM payment;
+---------+---------+----------+----------+--------------+
| max_amt | min_amt | avg_amt  | tot_amt  | num_payments |
+---------+---------+----------+----------+--------------+
|   11.99 |    0.00 | 4.200667 | 67416.51 |        16049 |
+---------+---------+----------+----------+--------------+
1 row in set (0.09 sec)
```

# Açık(Implicit) ve Örtülü(Explicit) Gruplar

Önceki örnekte, sorgu tarafından döndürülen her değer, bir kümeleme metodu tarafından üretilir. **Group by** cümlesi olmadığından, tek bir örtük grup vardır (ödeme tablosundaki tüm satırlar).

Ancak çoğu durumda, kümeleme metotları tarafından oluşturulan sütunlarla birlikte ek sütunlar almak isteyeceksiniz. Örneğin, tüm müşteriler yerine her müşteri için aynı

beş kümeleme metodunu yürütmek üzere önceki sorguyu genişletmek isteseydiniz? Bu sorgu için, aşağıdaki gibi beş kümeleme metoduyla birlikte **customer_id** sütununu almak istersiniz:

```
SELECT customer_id,
  MAX(amount) max_amt,
  MIN(amount) min_amt,
  AVG(amount) avg_amt,
  SUM(amount) tot_amt,
  COUNT(*) num_payments
FROM payment;

ERROR 1140 (42000): In aggregated query without GROUP BY,
  expression #1 of SELECT list contains nonaggregated column
```

Ödeme tablosunda bulunan her müşteriye kümeleme metotlarının uygulanmasını istediğinizi açıkça görseniz de, verilerin nasıl gruplandırılması gerektiğini açıkça belirtmediğiniz için bu sorgu başarısız olur. Bu nedenle, kümeleme işlevlerinin hangi satır grubuna uygulanacağını belirtmek için bir **group by** cümlesi eklemeniz gerekir:

```
mysql> SELECT customer_id,
    ->   MAX(amount) max_amt,
    ->   MIN(amount) min_amt,
    ->   AVG(amount) avg_amt,
    ->   SUM(amount) tot_amt,
    ->   COUNT(*) num_payments
    -> FROM payment
    -> GROUP BY customer_id;
+-------------+---------+---------+----------+---------+--------------+
| customer_id | max_amt | min_amt | avg_amt  | tot_amt | num_payments |
+-------------+---------+---------+----------+---------+--------------+
|           1 |    9.99 |    0.99 | 3.708750 |  118.68 |           32 |
|           2 |   10.99 |    0.99 | 4.767778 |  128.73 |           27 |
|           3 |   10.99 |    0.99 | 5.220769 |  135.74 |           26 |
|           4 |    8.99 |    0.99 | 3.717273 |   81.78 |           22 |
|           5 |    9.99 |    0.99 | 3.805789 |  144.62 |           38 |
|           6 |    7.99 |    0.99 | 3.347143 |   93.72 |           28 |
...
|         594 |    8.99 |    0.99 | 4.841852 |  130.73 |           27 |
|         595 |   10.99 |    0.99 | 3.923333 |  117.70 |           30 |
|         596 |    6.99 |    0.99 | 3.454286 |   96.72 |           28 |
|         597 |    8.99 |    0.99 | 3.990000 |   99.75 |           25 |
|         598 |    7.99 |    0.99 | 3.808182 |   83.78 |           22 |
|         599 |    9.99 |    0.99 | 4.411053 |   83.81 |           19 |
+-------------+---------+---------+----------+---------+--------------+
599 rows in set (0.04 sec)
```

**Group by** yan tümcesinin dahil edilmesiyle, önce **customer_id** sütununda aynı değere sahip satırları birlikte gruplanır ve ardından 599 grubun her birine beş kümeleme metodu uygulanır.

# Grup İşlemleri

Her gruptaki üye sayısını belirlemek için **count()** metodu kullanırken, gruptaki tüm üyeleri sayma veya grubun tüm üyeleri arasında bir sütun için yalnızca farklı değerleri sayma seçeneğiniz vardır.

Örneğin, **costumer_id** sütunuyla birlikte **count()** işlevini iki farklı şekilde kullanan aşağıdaki sorguyu göz önünde bulundurun:

```
mysql> SELECT COUNT(customer_id) num_rows,
    ->   COUNT(DISTINCT customer_id) num_customers
    -> FROM payment;
+----------+---------------+
| num_rows | num_customers |
+----------+---------------+
|    16049 |           599 |
+----------+---------------+
1 row in set (0.01 sec)
```

Sorgudaki ilk sütun sadece ödeme tablosundaki satır sayısını sayarken, ikinci sütun **costumer_id** sütunundaki değerleri inceler ve yalnızca benzersiz değerlerin sayısını sayar. Bu nedenle, ayrı belirterek**, count()** işlevi, gruptaki değerlerin sayısını basitçe saymak yerine, yinelenenleri bulmak ve kaldırmak için grubun her üyesi için bir sütunun değerlerini inceler.

Bir filmin kiralanması ile sonradan iade edilmesi arasındaki maksimum gün sayısını bulmak isteyebilirsiniz. Bunu aşağıdaki sorgu ile başarabilirsiniz:

```
mysql> SELECT MAX(datediff(return_date,rental_date))
    -> FROM rental;
+---------------------------------------+
| MAX(datediff(return_date,rental_date)) |
+---------------------------------------+
|                                    33 |
+---------------------------------------+
1 row in set (0.01 sec)
```

**datediff**, her kiralama için dönüş tarihi ile kiralama tarihi arasındaki gün sayısını hesaplamak için kullanılır ve **max**, bu durumda 33 gün olan en yüksek değeri döndürür.

Kümeleme metotlarını veya herhangi bir sayısal hesaplama türünü gerçekleştirirken, her zaman boş değerlerin hesaplamanızın sonucunu nasıl etkileyebileceğini düşünmelisiniz.

```
mysql> CREATE TABLE number_tbl
    ->  (val SMALLINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO number_tbl VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO number_tbl VALUES (3);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO number_tbl VALUES (5);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT COUNT(*) num_rows,
    ->   COUNT(val) num_vals,
    ->   SUM(val) total,
    ->   MAX(val) max_val,
    ->   AVG(val) avg_val
    -> FROM number_tbl;
+----------+----------+-------+---------+---------+
| num_rows | num_vals | total | max_val | avg_val |
+----------+----------+-------+---------+---------+
|        3 |        3 |     9 |       5 |  3.0000 |
+----------+----------+-------+---------+---------+
1 row in set (0.08 sec)

mysql> INSERT INTO number_tbl VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT COUNT(*) num_rows,
    ->   COUNT(val) num_vals,
    ->   SUM(val) total,
    ->   MAX(val) max_val,
    ->   AVG(val) avg_val
    -> FROM number_tbl;
+----------+----------+-------+---------+---------+
| num_rows | num_vals | total | max_val | avg_val |
+----------+----------+-------+---------+---------+
|        4 |        3 |     9 |       5 |  3.0000 |
+----------+----------+-------+---------+---------+
1 row in set (0.00 sec)
```

**sum(), max() ve avg()** metotlarının tümü, karşılaşılan **NULL** değerleri yok sayarlar. **count(*)** metodu 4 değerini döndürür; bu, **number_tbl** tablosu dört satır içerdiğinden kaynaklanır ancak, **count(val)** hala 3 değerini döndürür. Aradaki fark, **count(*)'**un satır sayısını saymasıdır, **count(val)** ise **val** sütununda bulunan değerlerin sayısını sayar ve karşılaşılan boş değerleri yok sayar.

# Tek Sütunlu Ve Çok Sütunlu Gruplama

Veri analiziyle uğraşan kişiler, ham verileri ihtiyaçlarına daha iyi uyacak şekilde manipüle etmek isteyecektir. Veri işleme için örnekler:

• Avrupadaki satışları bir coğrafi bölge için toplamlar halinde oluşturma

• 2020'nin en iyi satış elemanı gibi aykırı değerleri bulma

• Aylık kiralanan film sayısı gibi sıklıkların belirlenmesi

Bu tür sorguları yanıtlamak için, veritabanı sunucusundan satırları bir veya daha fazla sütun veya ifadeyle gruplandırmasını istemeniz gerekir. Birkaç örnekte zaten gördüğünüz gibi, **group by** cümlesi, bir sorgu içindeki verileri gruplandırma mekanizmasıdır.

Tek sütunlu gruplar, en basit ve en sık kullanılan gruplama türüdür. Örneğin, her aktörle ilişkili film sayısını bulmak istiyorsanız, film_actor.actor_id sütununda aşağıdaki gibi gruba ihtiyacınız vardır:

```
mysql> SELECT actor_id, count(*)
    -> FROM film_actor
    -> GROUP BY actor_id;
+----------+----------+
| actor_id | count(*) |
+----------+----------+
|        1 |       19 |
|        2 |       25 |
|        3 |       22 |
|        4 |       22 |
...
|      197 |       33 |
|      198 |       40 |
|      199 |       15 |
|      200 |       20 |
+----------+----------+
200 rows in set (0.11 sec)
```

Bu sorgu, her oyuncu için bir tane olmak üzere 200 grup oluşturur ve ardından grubun her bir üyesi için film sayısını toplar.

Bazı durumlarda, birden fazla sütuna yayılan gruplar oluşturmak isteyebilirsiniz. Önceki örneği genişleterek, her bir oyuncu için her film derecelendirmesi (G, PG, ...) için toplam film sayısını bulmak istediğinizi hayal edin.

```
mysql> SELECT fa.actor_id, f.rating, count(*)
    -> FROM film_actor fa
    ->   INNER JOIN film f
    ->   ON fa.film_id = f.film_id
    -> GROUP BY fa.actor_id, f.rating
    -> ORDER BY 1,2;
+----------+--------+----------+
| actor_id | rating | count(*) |
+----------+--------+----------+
|        1 | G      |        4 |
|        1 | PG     |        6 |
|        1 | PG-13  |        1 |
|        1 | R      |        3 |
|        1 | NC-17  |        5 |
|        2 | G      |        7 |
|        2 | PG     |        6 |
|        2 | PG-13  |        2 |
|        2 | R      |        2 |
|        2 | NC-17  |        8 |
...
|      199 | G      |        3 |
|      199 | PG     |        4 |
|      199 | PG-13  |        4 |
|      199 | R      |        2 |
|      199 | NC-17  |        2 |
|      200 | G      |        5 |
|      200 | PG     |        3 |
|      200 | PG-13  |        2 |
|      200 | R      |        6 |
|      200 | NC-17  |        4 |
+----------+--------+----------+
996 rows in set (0.01 sec)
```

Sorgu **film_actor** tablosunu **film** tablosuyla birleştirerek bulunan her **actor_id** ve film **rating** kombinasyonu için bir tane olmak üzere 996 grup oluşturur. **Rating** sütununu **select** yan tümcesine eklemenin yanı sıra, **rating** bir tablodan alındığından ve **maks** veya **count** gibi bir metotolar aracılığıyla oluşturulmadığından gruplama yapılabilir.

# İfadelerden gelen değerlere göre gruplama

Verileri gruplamak için sütunları kullanmanın yanı sıra, ifadeler tarafından oluşturulan değerlere dayalı olarak gruplar oluşturabilirsiniz. Kiralamaları yıllara göre gruplandırmak istendiğinde:

```
mysql> SELECT extract(YEAR FROM rental_date) year,
    ->   COUNT(*) how_many
    -> FROM rental
    -> GROUP BY extract(YEAR FROM rental_date);
+------+----------+
| year | how_many |
+------+----------+
| 2005 |    15862 |
| 2006 |      182 |
+------+----------+
2 rows in set (0.01 sec)
```

Bu sorgu, **rental** tablosundaki satırları gruplandırmak için bir tarihin yalnızca yıl kısmını döndürmek için **Extract()** metodunu kullanan oldukça basit bir ifade kullanır.

# Gruplarda Filtreleme

Verileri gruplandırırken, gruplar oluşturulduktan sonra verilere filtre koşulları da uygulayabilirsiniz. **Having** yan tümcesi, bu tür filtre koşullarını yerleştirmeniz gereken yerdir:

```
mysql> SELECT fa.actor_id, f.rating, count(*)
    -> FROM film_actor fa
    ->   INNER JOIN film f
    ->   ON fa.film_id = f.film_id
    -> WHERE f.rating IN ('G','PG')
    -> GROUP BY fa.actor_id, f.rating
    -> HAVING count(*) > 9;
+----------+--------+----------+
| actor_id | rating | count(*) |
+----------+--------+----------+
|      137 | PG     |       10 |
|       37 | PG     |       12 |
|      180 | PG     |       12 |
|        7 | G      |       10 |
|       83 | G      |       14 |
|      129 | G      |       12 |
|      111 | PG     |       15 |
|       44 | PG     |       12 |
|       26 | PG     |       11 |
|       92 | PG     |       12 |
|       17 | G      |       12 |
|      158 | PG     |       10 |
|      147 | PG     |       10 |
|       14 | G      |       10 |
|      102 | PG     |       11 |
|      133 | PG     |       10 |
+----------+--------+----------+
16 rows in set (0.01 sec)
```

Bu sorgunun iki filtre koşulu vardır: biri **where** ile, G veya PG dışında bir puan alan tüm filmleri filtreleme ve, diğeri de **having** ile, 10'dan az filmde yer alan oyuncuları filtreleme. Böylece, filtrelerden biri gruplanmadan önce veriler üzerinde, diğer filtre gruplar oluşturulduktan sonra veriler üzerinde uygulanır. Her iki filtreyi de yanlışlıkla where yan tümcesine koyarsanız, aşağıdaki hatayı görürsünüz:

```
mysql> SELECT fa.actor_id, f.rating, count(*)
    -> FROM film_actor fa
    ->   INNER JOIN film f
    ->   ON fa.film_id = f.film_id
    -> WHERE f.rating IN ('G','PG')
    ->   AND count(*) > 9
    -> GROUP BY fa.actor_id, f.rating;
ERROR 1111 (HY000): Invalid use of group function
```

Bir sorgunun **where** yan tümcesine bir kümeleme metodu ekleyemediğiniz için bu sorgu başarısız olur. Bunun nedeni, **where** yan tümcesindeki filtrelerin gruplama gerçekleşmeden önce değerlendirilmesidir, bu nedenle sunucu gruplar üzerinde herhangi bir metodu gerçekleştiremez.

Group by cümlesi içeren bir sorguya filtreler eklerken, filtrenin ham veriler üzerinde mi, **(where)** yoksa gruplandırılmış veriler üzerinde mi, **(having)** olduğunu dikkatlice düşünün.