ГЛАВА 8

Группировка и агрегация

Данные обычно хранятся на самом низком уровне детализации, необходимом пользователям любой базы данных; если Вере из бухгалтерии нужно просмотреть отдельные транзакции клиентов, в базе данных должна быть таблица, в которой хранятся отдельные транзакции. Это, однако, не означает, что все пользователи должны иметь дело с данными в том виде, в котором они хранятся в базе данных. В этой главе основное внимание уделяется тому, как данные могут быть сгруппированы и агрегированы, чтобы позволить пользователям взаимодействовать с ними на более высоком уровне детализации, чем используемый для хранения в базе данных.

Концепции группировки

Иногда может понадобиться обнаружить определенные тенденции в данных. Для этого потребуется небольшая обработка данных сервером, чтобы вы могли сгенерировать искомые результаты. Например, предположим, что вы отвечаете за отправку купонов на бесплатную аренду по адресам ваших лучших клиентов. Вы можете выполнить простой запрос, чтобы просмотреть необработанные данные:

```
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)
```

При наличии 599 клиентов, имеющих более 16 000 записей об аренде, просмотрев необработанные данные, невозможно определить, какие клиенты взяли напрокат больше всего фильмов. Вместо этого вы можете попросить сервер базы данных сгруппировать данные с помощью предложения `group by`. Вот тот же запрос, но с использованием предложения `group by` для группировки данных о прокате по идентификатору клиента:

```
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)
```

Результирующий набор содержит по одной строке для каждого отдельного значения в столбце `customer_id`, в результате получается 599 строк вместо общего количества в 16 044 строк. Причина меньшего результирующего набора в том, что некоторые заказчики брали напрокат более одного фильма. Чтобы увидеть, сколько фильмов было взято напрокат каждым клиентом, можно использовать *агрегатную функцию* в предложении `select` для подсчета количества строк в каждой группе:

```
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)
```

Агрегатная функция `count()` подсчитывает количество строк в каждой группе, а звездочка сообщает серверу, что нужно считать все, что есть в группе. Используя комбинацию `group by` и агрегатную функцию `count()`, вы можете сгенерировать точные данные, необходимые для ответа на имеющийся бизнес-вопрос.

Глядя на результаты, можно увидеть, что 32 фильма были взяты напрокат клиентом с идентификатором 1, а 25 фильмов — клиентом с идентификатором 597. Чтобы определить, какие клиенты взяли напрокат больше всего фильмов, просто добавим предложение `order by`:

```
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)
```

Теперь, когда результаты отсортированы, легко увидеть, что клиент с идентификатором 148 брал напрокат наибольшее количество фильмов (46), в то время как клиент с идентификатором 318 брал наименьшее количество фильмов (12).

При группировке данных вам может потребоваться отфильтровать нежелательные данные из набора результатов на основе групп данных, а не необработанных данных. Поскольку предложение `group by` выполняется после того, как вычислено предложение `where`, добавить условия фильтрации к предложению `where` для этой цели нельзя. Например, вот к чему приводит попытка отфильтровать клиентов, бравших напрокат менее 40 фильмов:

```
mysql> SELECT customer_id, count(*)
    -> FROM rental
    -> WHERE count(*) >= 40
    -> GROUP BY customer_id;
ERROR 1111 (HY000): Invalid use of group function[1]
```

Вы не можете обратиться к агрегатной функции `count(*)` в предложении `where`, потому что во время вычисления предложения `where` группы еще не были сгенерированы. Вместо этого вы должны поместить условия группового фильтра в предложение `having`. Вот как выглядит правильный запрос:

```
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)
```

Поскольку группы, содержащие менее 40 членов, отфильтрованы с помощью предложения `having`, результирующий набор теперь содержит только тех клиентов, которые брали напрокат 40 и более фильмов.

[1] Неверное применение групповой функции.

Агрегатные функции

Агрегатные функции выполняют определенные операции над всеми строками в группе. Несмотря на то что каждый сервер базы данных имеет собственный набор специализированных агрегатных функций, имеются распространенные агрегатные функции, реализованные всеми основными серверами, в частности следующие.

`max()`

Возвращает максимальное значение в наборе.

`min()`

Возвращает минимальное значение в наборе.

`avg()`

Возвращает усредненное значение набора.

`sum()`

Возвращает сумму значений набора.

`count()`

Возвращает количество значений в наборе.

Вот как выглядит запрос, в котором используются все распространенные агрегатные функции для анализа данных по прокату фильмов:

```
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)
```

Результаты этого запроса говорят о том, что из 16 049 строк в таблице платежей максимальная выплаченная за прокат фильма сумма составила 11,99 доллара, минимальная — 0 долларов, средний платеж равен 4,20 доллара, а общая сумма всех платежей — 67 416,51 доллара. Надеюсь, этот пример даст вам представление о роли агрегатных функций. В следующих подразделах более подробно разъясняется, как эти функции можно использовать.

Неявная и явная группировка

В предыдущем примере каждое значение, возвращаемое запросом, генерируется агрегатной функцией. Поскольку предложения `group by` в запросе нет, существует единственная неявная группа (все строки в таблице `payment`).

Однако в большинстве случаев требуется получить дополнительные столбцы вместе со столбцами, генерируемыми агрегатными функциями. Например, можно расширить предыдущий запрос и выполнить те же пять агрегатных функций, но не для всех клиентов одновременно, а для каждого клиента. Для такого запроса нужно вместе с пятью агрегатными функциями выполнить выборку `customer_id`:

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

Однако, если вы попытаетесь выполнить такой запрос, то получите сообщение об ошибке:

```
ERROR 1140 (42000): In aggregated query without GROUP BY,
    expression #1 of SELECT list contains nonaggregated column[2]
```

Хотя для вас может быть очевидно, что вы хотите применения агрегатных функций к каждому найденному в таблице `payment` клиенту, этот запрос не выполняется, потому что в нем не указано *явно*, как должны быть сгруппированы данные. Следовательно, в запрос нужно добавить предложение `group by`, чтобы указать, к какой группе строк следует применять агрегатные функции:

```
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 |
+-----------+--------+--------+---------+--------+-------------+
```

[2] В агрегированном запросе без `GROUP BY` выражение №1 списка `SELECT` содержит неагрегированный столбец.

```
|           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)
```

При включении предложения `group by` сервер понимает, что надо сначала сгруппировать строки, имеющие одинаковое значение в столбце `customer_id`, а затем применить к каждой из 599 групп пять агрегатных функций.

Подсчет различных значений

При использовании функции `count()` для определения количества членов в каждой группе у вас есть выбор: подсчитать *все* элементы группы или подсчитать только *различные* значения столбца среди всех элементов группы.

Рассмотрим, например, следующий запрос, в котором функция `count()` и столбец `customer_id` используются двумя разными способами:

```
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)
```

В первом столбце запроса просто подсчитывается количество строк в таблице `payment`, в то время как во втором столбце исследуются значения в столбце `customer_id` и подсчитывается только количество уникальных значений. Таким образом, при указании ключевого слова `distinct` функция `count()` проверяет значения столбца для каждого члена группы, находя и удаляя дубликаты, а не просто подсчитывает количество значений в группе.

Использование выражений

Наряду с использованием столбцов в качестве аргументов агрегатных функций можно использовать и выражения. Например, можно найти максимальное количество дней между моментом, когда фильм был взят напрокат, и последующим его возвратом. Эту информацию можно получить с помощью следующего запроса:

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

Функция `datediff` используется для вычисления количества дней между датой возврата фильма и датой взятия его напрокат для каждой аренды фильма, а функция `max` возвращает максимальное найденное значение, которое в данном случае составляет 33 дня.

Хотя в этом примере используется довольно простое выражение, на практике выражения, используемые в качестве аргументов агрегатных функций, могут быть любой необходимой степени сложности, лишь бы они возвращали числа, строки или даты. В главе 11, "Условная логика", показано, как для того, чтобы определить, должна ли конкретная строка быть включена в агрегацию, можно использовать выражения `case` с агрегатными функциями.

Обработка значений `null`

При выполнении агрегации (на самом деле — любого вида числовых вычислений) всегда следует учитывать, как на результат вычислений могут повлиять значения `null`. Для иллюстрации я построю простую таблицу для хранения числовых данных и заполню ее множеством {1, 3, 5}:

```
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)
```

Рассмотрим следующий запрос, который выполняет пять агрегатных функций для указанного множества чисел:

```
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)
```

Результаты оказываются такими, как и следовало ожидать: и `count(*)`, и `count(val)` возвращают значение 3, `sum(val)` возвращает значение 9, `max(val)` возвращает 5, а `avg(val)` возвращает 3. Теперь я добавляю в таблицу `number_tbl` значение `null` и снова выполняю тот же запрос:

```
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)
```

Даже при добавлении в таблицу значения `null` функции `sum()`, `max()` и `avg()` возвращают те же значения; такие результаты указывают на то, что эти функции игнорируют любые встречающиеся значения `null`. Функция `count(*)` теперь возвращает значение 4, которое является корректным, поскольку таблица `number_tbl` сейчас содержит четыре строки, а функция `count(val)` по-прежнему возвращает значение 3. Дело в том, что `count(*)` подсчитывает количество строк, тогда как `count(val)` подсчитывает

количество *значений*, содержащихся в столбце `val`, и игнорирует любые обнаруженные в нем значения `null`.

Генерация групп

Людей редко интересуют необработанные данные. Обычно те, кто заинтересован в анализе данных, манипулируют необработанными данными так, чтобы они удовлетворяли их потребностям. Распространенными примерами манипуляции являются:

- генерация итогов для географического региона, например общий объем продаж в Европе;
- выявление отклоняющихся значений, таких как лучший продавец 2020 года;
- определение частотных показателей, например количества фильмов, взятых напрокат за месяц.

Чтобы ответить на эти типы запросов, нужно запросить сервер базы данных сгруппировать строки вместе по одному или нескольким столбцам или выражениям. Как вы уже видели в приводимых примерах, механизм для группировки данных в запросе — это предложение `group by`. Из этого раздела вы узнаете, как группировать данные по одному или нескольким столбцам, как группировать данные с помощью выражений и как создавать сводки внутри групп.

Группировка по одному столбцу

Группы из одного столбца — это самый простой и наиболее часто используемый тип группировки. Если, например, вы хотите найти количество фильмов, связанных с каждым актером, нужна группировка по единственному столбцу `film_actor.actor_id`:

```
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)
```

Этот запрос генерирует 200 групп, по одной для каждого актера, а затем суммирует количество фильмов для каждого участника группы.

Многостолбцовая группировка

В некоторых случаях вам может потребоваться создавать группы, охватывающие более одного столбца. Расширяя предыдущий пример, представьте, что для каждого актера вы хотите найти общее число фильмов с разными рейтингами (G, PG, ...). В следующем примере показано, как этого добиться:

```
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)
```

Эта версия запроса генерирует 996 групп, по одной для каждой комбинации "актер/рейтинг фильма", полученной путем соединения таблицы `film_actor` с таблицей `film`. Столбец `rating` я добавил и в предложение `select`, и в предложение `group by`, поскольку значение `rating` извлекается из таблицы, а не генерируется с помощью агрегатной функции, такой как `max` или `count`.

Группировка с помощью выражений

Для группировки можно использовать не только данные столбцов, но и значения, генерируемые выражениями. Рассмотрим следующий запрос, который группирует прокат по годам:

```
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)
```

В этом запросе применено довольно простое выражение, которое использует функцию `extract()` для того, чтобы вернуть из даты только год для соответствующей группировки строк в таблице `rental`.

Генерация итоговых данных

В разделе "Многостолбцовая группировка" приведен пример, в котором подсчитывается количество фильмов для каждого актера и рейтинга фильмов. Пусть, однако, вместе с общим количеством для каждой комбинации "актер/рейтинг", нужно получить и общее количество для каждого отдельного актера. Можно выполнить дополнительный запрос и объединить результаты, можно загрузить результаты запроса в электронную таблицу или создать сценарий Python, программу Java или какой-либо иной механизм для получения этих данных и выполнения дополнительных вычислений. Но еще лучше использовать конструкцию `with rollup`, чтобы сервер базы данных сделал эту работу вместо вас. Вот измененный запрос, использующий `with rollup` в предложении `group by`:

```
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 WITH ROLLUP
    -> ORDER BY 1,2;
+----------+--------+----------+
| actor_id | rating | count(*) |
+----------+--------+----------+
| NULL     | NULL   |     5462 |
|        1 | NULL   |       19 |
|        1 | G      |        4 |
|        1 | PG     |        6 |
|        1 | PG-13  |        1 |
|        1 | R      |        3 |
|        1 | NC-17  |        5 |
|        2 | NULL   |       25 |
|        2 | G      |        7 |
|        2 | PG     |        6 |
|        2 | PG-13  |        2 |
|        2 | R      |        2 |
|        2 | NC-17  |        8 |
| ...                          |
|      199 | NULL   |       15 |
|      199 | G      |        3 |
|      199 | PG     |        4 |
|      199 | PG-13  |        4 |
|      199 | R      |        2 |
|      199 | NC-17  |        2 |
|      200 | NULL   |       20 |
|      200 | G      |        5 |
|      200 | PG     |        3 |
|      200 | PG-13  |        2 |
|      200 | R      |        6 |
|      200 | NC-17  |        4 |
+----------+--------+----------+
1197 rows in set (0.07 sec)
```

Теперь в результирующем наборе имеется 201 дополнительная строка, по одной для каждого из 200 различных актеров и одна общая (для всех актеров вместе). В столбце `rating` для итоговых значений для 200 актеров предоставляется значение `null`, поскольку выполняется накопление по всем рейтингам. Взглянув, например, на первую строку для `actor_id`, равного `200`, вы увидите, что всего с этим актером связано 20 фильмов; это значение равно сумме итогов по каждому рейтингу (4 NC-17 + 6 R + 2 PG-13 + 3 PG + 5 G). Для строки общего итога (первая строка вывода) значение `null` предоставлено как для столбца `actor_id`, так и для столбца `rating`; сумма в первой строке вывода равна 5462, что соответствует числу строк в таблице `film_actor`.

В Oracle Database нужно использовать немного другой синтаксис, указывающий, что вы хотите выполнить сведение итоговых данных. Предложение `group by` из предыдущего запроса при использовании Oracle выглядело бы следующим образом:

```
GROUP BY ROLLUP(fa.actor_id, f.rating)
```

Преимущество этого синтаксиса в том, что он позволяет выполнять сведение для подмножества столбцов в предложении `group by`. Если группировка выполняется по столбцам `a`, `b` и `c`, например, можно указать, что сервер должен выполнять сведение только для столбцов `b` и `c` с помощью следующей конструкции:

```
GROUP BY a, ROLLUP(b, c)
```

Если, помимо итогов по актерам, вы хотите подсчитать итоги по рейтингу, можете использовать конструкцию `with cube`, которая будет генерировать итоговые строки для *всех* возможных комбинаций столбцов группировки. К сожалению, конструкция `with cube` недоступна в MySQL версии 8.0, но доступна в SQL Server и Oracle Database.

Условия группового фильтра

В главе 4, "Фильтрация", вы познакомились с типами условий фильтрации и узнали, как их использовать в предложении `where`. При группировке данных также можно применить фильтрующее условие к данным *после* того, как были сгенерированы группы. Эти типы условий фильтрации должны быть размещены в предложении `having`. Рассмотрим следующий пример:

```
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)
```

Этот запрос имеет два условия фильтрации: одно — в предложении `where`, которое отфильтровывает любые фильмы с рейтингом, отличным от G или PG, и еще одно — в предложении `having`, которое отфильтровывает всех актеров, снявшихся менее чем в 10 фильмах. Таким образом, один из фильтров действует на данные *до* их группировки, а другой — *после* того, как группы были созданы. Если вы по ошибке поместите оба фильтра в предложение `where`, то получите сообщение об ошибке:

```
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[3]
```

Этот запрос не работает, потому что нельзя включать агрегатную функцию в предложение `where` запроса. Это связано с тем, что фильтры в предложении `where` вычисляются до группировки, поэтому сервер еще не в состоянии выполнять какие-либо функции для групп.

При добавлении фильтров в запрос, который включает предложение `group by`, хорошо подумайте, действует ли фильтр на необработанные данные, — в этом случае он должен принадлежать предложению `where`. Если же фильтр относится к сгруппированным данным, он должен принадлежать предложению `having`.

[3] Неверное применение групповой функции.

Проверьте свои знания

Предлагаемые здесь упражнения призваны закрепить понимание вами возможностей группировки и агрегации SQL. Ответы к упражнениям вы найдете в приложении Б.

УПРАЖНЕНИЕ 8.1

Создайте запрос, который подсчитывает количество строк в таблице `payment`.

УПРАЖНЕНИЕ 8.2

Измените запрос из упражнения 8.1 так, чтобы подсчитать количество платежей, произведенных каждым клиентом. Выведите идентификатор клиента и общую уплаченную сумму для каждого клиента.

УПРАЖНЕНИЕ 8.3

Измените запрос из упражнения 8.2, включив в него только тех клиентов, у которых имеется не менее 40 выплат.