Урок 4

Агрегация данных

## 

[Группировка данных](#_qjnqizjxpwjm)

[Агрегатные функции](#_c84or328wnrq)

[Специальные возможности GROUP BY](#_3tb0lm9pg2u0)

[Используемые источники](#_2xcytpi)

# 

# 

# Группировка данных

В одном из предыдущих роликов мы с вами сталкивались с результирующими таблицами, которые содержат повторяющиеся значения:

|  |
| --- |
| SELECT catalog\_id FROM products; |

Мы уже знаем механизм получения уникальных значений при помощи ключевого слова **DISTINCT**:

|  |
| --- |
| SELECT DISTINCT catalog\_id FROM products; |

При помощи вычисляемых столбцов мы и сами можем создавать такие повторяющиеся комбинации:

|  |
| --- |
| SELECT id, name, id % 3 FROM products ORDER BY id % 3; |

В языке SQL для работы с такими группами предназначено специальное ключевое слово **GROUP BY**. Например, задачу получения уникальных значений можно решить следующим образом:

|  |
| --- |
| SELECT catalog\_id FROM products GROUP BY catalog\_id; |

В качестве значений для создания групп могут выступать не только столбцы таблицы, но и вычисляемые значения. Например, давайте разделим пользователей в таблице на три группы: родившихся в 80-х, 90-х и 2000-х годах.

|  |
| --- |
| SELECT id, name, SUBSTRING(birthday\_at, 1, 3) FROM users; |

Здесь мы преобразуем календарный тип **DATETIME** поля **birthday\_at** к строковому значению и при помощи функции **SUBSTRING** извлекаем первые три цифры года рождения. Давайте назначим вычисляемому значению псевдоним при помощи ключевого слова **AS** и отсортируем значения при помощи **ORDER BY**.

|  |
| --- |
| SELECT id, name, SUBSTRING(birthday\_at, 1, 3) AS decade FROM users ORDER BY decade; |

Обратите внимание, что мы можем использовать псевдоним **decade** в конструкции **ORDER BY**.

При помощи конструкции **GROUP BY** мы можем сгруппировать поля по декадам:

|  |
| --- |
| SELECT SUBSTRING(birthday\_at, 1, 3) AS decade FROM users GROUP BY decade; |

Если мы попытаемся вывести имена пользователей, мы потерпим неудачу:

|  |
| --- |
| SELECT id, name, SUBSTRING(birthday\_at, 1, 3) AS decade FROM users GROUP BY decade; |

Каждая из групп содержит в себе несколько пользователей и непонятно, какого из них следует выводить. Ранее MySQL выводила случайного пользователя, однако сейчас такое поведение отменено. Такой режим по-прежнему можно включить, СУБД даже подсказывает в сообщении об ошибке, как это можно сделать. Однако лучше этого не делать, чтобы ваш SQL-код оставался совместимым с другими СУБД.

Какую пользу можно извлечь из сгруппированных значений? MySQL предоставляет несколько функций, которые называются агрегатными. Они позволяют работать с содержимым групп, полученных **GROUP BY**. Например, мы можем подсчитать количество записей внутри каждой из групп:

|  |
| --- |
| SELECT COUNT(\*), SUBSTRING(birthday\_at, 1, 3) AS decade FROM users GROUP BY decade; |

Таким образом, у нас 3 пользователя родились в 80-х, два в 90-х и один в 2000-х. Полученные значения мы по-прежнему можем сортировать при помощи конструкции **ORDER BY**:

|  |
| --- |
| SELECT  COUNT(\*),  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade ORDER BY  decade DESC; |

Причем сортировать можно не группируемому значению, но и по любому другому полю. Например, давайте назначим функции **COUNT()** псевдоним **total** и отсортируем результаты по этому значению:

|  |
| --- |
| SELECT  COUNT(\*) AS total,  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade ORDER BY  total DESC; |

Порядок следования ключевых слов важен: мы не можем размещать ключевое слово **ORDER BY** раньше **GROUP BY**, иначе мы получаем сообщение об ошибке. Это же относится к ключевому слову **LIMIT**, которое всегда должно располагаться после всех остальных ключевых слов.

Когда в запросе мы не используем конструкцию **GROUP BY**, вся таблица рассматривается как одна большая группа. Поэтому, если мы применим функцию **COUNT(\*)** к таблице **users**, мы можем получить количество всех пользователей, без учета года их рождения:

|  |
| --- |
| SELECT COUNT(\*) FROM users; |

Следующий ролик будет полностью посвящен агрегационным функциям. Пока давайте посмотрим, какие задачи можно решать при помощи групп. Посмотреть содержимое группы мы можем при помощи специальной функции **GROUP\_CONCAT**:

|  |
| --- |
| SELECT  GROUP\_CONCAT(name),  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade; |

Таким образом, мы можем получить список всех пользователей в каждой из групп.

Функция **GROUP\_CONCAT** допускает задание разделителя, для этого внутри функции используется ключевое слово **SEPARATOR**. Давайте зададим в качестве разделителя пробел:

|  |
| --- |
| SELECT  GROUP\_CONCAT(name SEPARATOR ' '),  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade; |

Ключевое слово **ORDER BY** позволяет отсортировать значения в рамках возвращаемой строки. Давайте отсортируем имена пользователей в обратном порядке:

|  |
| --- |
| SELECT  GROUP\_CONCAT(name ORDER BY name DESC SEPARATOR ' '),  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade; |

Функция **GROUP\_CONCAT** имеет ограничения: она может извлекать из группы максимум 1000 элементов. Впрочем, это значение можно увеличить за счет изменения параметра сервера **group\_concat\_max\_len**. Как это сделать, мы подробнее будем разбирать в одной из следующих тем.

# Агрегатные функции

Количество записей в таблице можно узнать при помощи функции **COUNT()**, которая принимает в качестве аргумента имя столбца. Функция возвращает число строк в таблице, значения столбца для которых отличны от **NULL**:

|  |
| --- |
| SELECT COUNT(id) FROM catalogs; |

В качестве параметра функции наряду с именами столбцов может выступать символ звездочки (\*). При использовании символа **\*** будет возвращено число строк таблицы независимо от того, принимают какие-то из них значение **NULL** или нет.

|  |
| --- |
| SELECT COUNT(\*) FROM catalogs; |

Функции вроде **COUNT** называются агрегационными. Дело в том, что их значение изменяется при использовании конструкции **GROUP BY**. Конструкция **GROUP BY** разбивает таблицу на отдельные группы, например:

|  |
| --- |
| SELECT catalog\_id FROM products; SELECT catalog\_id FROM products GROUP BY catalog\_id; |

Функция **COUNT()** возвращает результат для каждой из этих групп.

|  |
| --- |
| SELECT catalog\_id, COUNT(\*) AS total FROM products GROUP BY catalog\_id; |

Давайте подробнее остановимся на реакции агрегационных функций на NULL-значение. Для этого создадим таблицу, содержащую два столбца — **id** и **value**:

|  |
| --- |
| CREATE TABLE tbl (  id INT NOT NULL,  value INT DEFAULT NULL ); INSERT INTO tbl VALUES (1, 230); INSERT INTO tbl VALUES (2, NULL); INSERT INTO tbl VALUES (3, 405); INSERT INTO tbl VALUES (4, NULL);  SELECT \* FROM tbl; |

Давайте применим функцию **COUNT()** к обоим столбцам:

|  |
| --- |
| SELECT COUNT(id), COUNT(value) FROM tbl; |

Это связано с тем, что **COUNT** игнорирует NULL-поля.

|  |
| --- |
| SELECT COUNT(\*) FROM tbl; |

Если вместо имени столбца используется звездочка, значения **NULL** не влияют на результат:

|  |
| --- |
| SELECT  id,  catalog\_id FROM  products; |

Давайте попробуем подсчитать количество элементов для полей **id** и **catalog\_id**:

|  |
| --- |
| SELECT  COUNT(id) AS total\_ids,  COUNT(catalog\_id) AS total\_catalog\_ids FROM  products; |

Результаты совпадают, однако если мы добавим в функцию **COUNT** ключевое слово **DISTINCT**, мы можем добиться того, что будут подсчитываться только уникальные значения:

|  |
| --- |
| SELECT  COUNT(DISTINCT id) AS total\_ids,  COUNT(DISTINCT catalog\_id) AS total\_catalog\_ids FROM  products; |

Итак, у нас 7 товарных позиций в двух каталогах.

Функции **MIN()** и **MAX()** возвращают минимальное и максимальное значения столбца:

|  |
| --- |
| SELECT  MIN(price) AS min,  MAX(price) AS max FROM  products; |

Здесь мы извлекаем минимальную и максимальную цену в интернет-магазине. При группировки по полю **catalog\_id** мы получим максимальную и минимальную цену в рамках каждого из разделов каталога:

|  |
| --- |
| SELECT  catalog\_id,  MIN(price) AS min,  MAX(price) AS max FROM  products GROUP BY  catalog\_id; |

Агрегационные функции, можно применять только после ключевого слова **SELECT**. Попытка использования функций **MIN()** и **MAX()** в выражении **WHERE** приведет к ошибке.

|  |
| --- |
| SELECT \* FROM products WHERE price = MAX(price); |

Решить эту задачу проще всего с использованием сортировки, используя ключевое слово **ORDER BY**:

|  |
| --- |
| SELECT id, name, price FROM products ORDER BY price DESC LIMIT 1; |

Функция **AVG()** возвращает среднее значение аргумента. Давайте подсчитаем среднюю цену товара:

|  |
| --- |
| SELECT AVG(price) FROM products; |

Так как мы не производим группировку, средняя цена вычисляется для всех товарных позиций, занесенных в таблицу **products.** При желании мы можем округлить полученное число, например до второго знака после запятой при помощи функции **ROUND**:

|  |
| --- |
| SELECT ROUND(AVG(price), 2) FROM products; |

Если мы добавим **GROUP BY**, например по полю **catalog\_id**, мы получим среднее цены для каждого из разделов:

|  |
| --- |
| SELECT  catalog\_id,  ROUND(AVG(price), 2) AS price FROM  products GROUP BY  catalog\_id; |

Внутри агрегационных функций допускается использовать вычисляемые значения. Например, мы можем увеличить значение цены на 20 %:

|  |
| --- |
| SELECT  catalog\_id,  ROUND(AVG(price \* 1.2), 2) AS price FROM  products GROUP BY  catalog\_id; |

Сумму всех значений столбца можно подсчитать при помощи функции **SUM()**:

|  |
| --- |
| SELECT SUM(price) FROM products; |

Она, так же как и все остальные агрегатные функции, будет подсчитывать только значения, отличные от **NULL**:

|  |
| --- |
| SELECT catalog\_id, SUM(price) FROM products GROUP BY catalog\_id; |

Так мы можем подсчитать сумму всех цен в каждом из разделов.

# Специальные возможности GROUP BY

Каждая строка результирующего запроса с **GROUP BY** представляет собой отдельную группу:

|  |
| --- |
| SELECT  COUNT(\*) AS total,  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade; |

Агрегационные функции позволяют получать результаты для каждой из групп в отдельности. Чаще при составлении условий требуется ограничить выборку по результату функции, например выбрать группы, где количество записей больше или равно двум.

Использование для этих целей конструкции **WHERE** приводит к ошибке. Для решения этой проблемы вместо ключевого слова **WHERE** используется ключевое слово **HAVING**, которое располагается вслед за конструкцией **GROUP BY**:

|  |
| --- |
| SELECT  COUNT(\*) AS total,  SUBSTRING(birthday\_at, 1, 3) AS decade FROM  users GROUP BY  decade HAVING  total >= 2; |

Допускается использование условия **HAVING** без группировки **GROUP BY**:

|  |
| --- |
| SELECT  \* FROM  users HAVING  birthday\_at >= '1990-01-01'; |

В этом случае каждая строка таблицы рассматривается как отдельная группа. Условие **HAVING** идеально подходит в ситуации, когда требуется обнаружить повторяющиеся значения:

|  |
| --- |
| TRUNCATE products; |

Давайте два раза вставим в таблицу **products** одни и те же значения:

|  |
| --- |
| INSERT INTO products  (name, description, price, catalog\_id) VALUES  ('Intel Core i3-8100', 'Процессор Intel', 7890.00, 1),  ('Intel Core i5-7400', 'Процессор Intel', 12700.00, 1),  ('AMD FX-8320E', 'Процессор AMD', 4780.00, 1),  ('AMD FX-8320', 'Процессор AMD', 7120.00, 1),  ('ASUS ROG MAXIMUS X HERO', 'Z370, Socket 1151-V2, DDR4, ATX', 19310.00, 2),  ('Gigabyte H310M S2H', 'H310, Socket 1151-V2, DDR4, mATX', 4790.00, 2),  ('MSI B250M GAMING PRO', 'B250, Socket 1151, DDR4, mATX', 5060.00, 2); SELECT id, name, catalog\_id FROM products; |

Существует много способов избавиться от таких дублей. Пока мы не знакомились с JOIN-соединениями и многотабличным запросом **DELETE**. Давайте решим эту задачу через промежуточную таблицу. Сформируем запрос на извлечение записей для размещения в промежуточной таблице:

|  |
| --- |
| SELECT  name, description, price, catalog\_id FROM  products GROUP BY  name, description, price, catalog\_id; |

Создадим таблицу **products\_new**, структура которой полностью повторяет структуру таблицы **products**:

|  |
| --- |
| CREATE TABLE products\_new (  id SERIAL PRIMARY KEY,  name VARCHAR(255) COMMENT 'Название',  description TEXT COMMENT 'Описание',  price DECIMAL (11,2) COMMENT 'Цена',  catalog\_id INT UNSIGNED,  created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,  updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  KEY index\_of\_catalog\_id (catalog\_id) ) COMMENT = 'Товарные позиции'; |

Теперь сформируем запрос **INSERT ... SELECT ...**, который будет вставлять в таблицу **products\_new** только уникальные значения из таблицы **products**. Так как поля **id**, **created\_at** и **updated\_at** не входят в групповой запрос, нам потребуется сформировать их снова, используя **NULL** и функцию **NOW()**:

|  |
| --- |
| INSERT INTO  products\_new SELECT  NULL, name, description, price, catalog\_id, NOW(), NOW() FROM  products GROUP BY  name, description, price, catalog\_id; |

Давайте посмотрим содержимое промежуточной таблицы:

|  |
| --- |
| SELECT id, name, catalog\_id FROM products\_new; |

Итак, выбраны только уникальные значения, и вместо 14 позиций у нас осталось 7. Теперь давайте уничтожим таблицы **products**:

|  |
| --- |
| DROP TABLE products; |

И переименуем таблицы **products\_new** в **products**. Для этого воспользуемся оператором **ALTER TABLE**.

|  |
| --- |
| ALTER TABLE products\_new RENAME products; SHOW TABLES; SELECT id, name, catalog\_id FROM products; |

Для группировки можно использовать вычисляемые значения. В таблице **users** для каждого из пользователей в поле **birthday\_at** указывается его дата рождения. Давайте извлечем года, на которые приходятся даты рождения, и случайного пользователя:

|  |
| --- |
| SELECT name, birthday\_at FROM users; |

Чтобы было интереснее, давайте добавим несколько записей с таким расчетом, чтобы дни рождения нескольких пользователей приходились на один год:

|  |
| --- |
| INSERT INTO users (name, birthday\_at) VALUES  ('Светлана', '1988-02-04'),  ('Олег', '1998-03-20'),  ('Юлия', '2006-07-12');  SELECT name, birthday\_at FROM users ORDER BY birthday\_at; |

Запрашиваем пользователей и видим, что на 1988, 1998 и 2006 приходится по две даты рождения. Теперь давайте извлечем из таблицы **users** года, на которые приходятся даты рождения:

|  |
| --- |
| SELECT YEAR(birthday\_at) FROM users ORDER BY birthday\_at; |

Избавляемся от повторов:

|  |
| --- |
| SELECT  YEAR(birthday\_at) AS birthday\_year FROM  users GROUP BY  birthday\_year ORDER BY  birthday\_year; |

Если мы сейчас внесем в SELECT-список имя пользователя, то потерпим неудачу. Обойти проблемы мы можем, вернув вместо имени пользователя какое-то агрегационное значение, например максимальное значение, что бы это не значило:

|  |
| --- |
| SELECT  MAX(name),  YEAR(birthday\_at) AS birthday\_year FROM  users GROUP BY  birthday\_year ORDER BY  birthday\_year; |

В данном случае нам все равно, какого мы пользователя вернем. На этот случай в MySQL предусмотрена специальная функция **ANY\_VALUE()**, которая возвращает случайное значение из группы:

|  |
| --- |
| SELECT  ANY\_VALUE(name),  YEAR(birthday\_at) AS birthday\_year FROM  users GROUP BY  birthday\_year ORDER BY  birthday\_year; |

Конструкция **WITH ROLLUP** позволяет добавить еще одну строку с суммой значений всех предыдущих строк. Возвращаясь к таблице пользователей **users**, подсчитываем количество пользователей, родившихся в 80-х, 90-х и 2000-х годах

|  |
| --- |
| SELECT  SUBSTRING(birthday\_at, 1, 3) AS decade,  COUNT(\*) FROM  users GROUP BY  decade; |

При помощи конструкции **WITH ROLLUP** мы можем добавить последнюю результирующую строку с количеством всех пользователей:

|  |
| --- |
| SELECT  SUBSTRING(birthday\_at, 1, 3) AS decade,  COUNT(\*) FROM  users GROUP BY  decade WITH ROLLUP; |

Как видно, в результирующую таблицу добавлена дополнительная строка с количеством всех пользователей и значением **NULL** для столбца **decade**. Значение **NULL** присваивается всем столбцам, кроме сгенерированного агрегатной функцией (в данном случае функцией **COUNT()**).

В MySQL версии 8.0 заменить значение **NULL** можно при помощи функции **GROUPING**, которая принимает в качестве аргумента имя столбца и возвращает **0**, если для него есть значение, и **1**, если в результирующей таблице поле принимает значение **NULL**. Применив функцию **IF**, можно заменить значение **NULL** каким-либо осмысленным текстом.

# Используемые источники

1. <https://dev.mysql.com/doc/refman/5.7/en/group-by-functions-and-modifiers.html>
2. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
3. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.
4. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с.
5. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.
6. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.
7. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.
8. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид Групп, 2011. — 336 с.