ГЛАВА 4

Основы работы с запросами

Запрос — это псевдоним оператора `SELECT`, который состоит из шести основных предложений. В каждом разделе данной главы подробно рассматривается то или иное предложение:

1) `SELECT`;
2) `FROM`;
3) `WHERE`;
4) `GROUP BY`;
5) `HAVING`;
6) `ORDER BY`.

В последнем разделе этой главы рассматривается предложение `LIMIT`, которое поддерживается MySQL, PostgreSQL и SQLite.

Примеры кода, приведенные в этой главе, ссылаются на четыре таблицы:

1) `waterfall` — водопады на Верхнем полуострове штата Мичиган;
2) `owner` — владельцы водопадов;
3) `county` — округа, в которых расположены водопады;
4) `tour` — экскурсионные туры, состоящие из нескольких остановок у водопадов.

Ниже приведен пример запроса, в котором используются шесть основных предложений. За ним следуют результаты запроса, которые также называются *результирующим набором*.

```
-- Туры с двумя и более общественными водопадами
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
FROM     tour t LEFT JOIN waterfall w
         ON t.stop = w.id
WHERE    w.open_to_public = 'y'
GROUP BY t.name
HAVING   COUNT(*) >= 2
ORDER BY tour_name;

tour_name   num_waterfalls
----------  ---------------
M-28                      6
Munising                  6
US-2                      4
```

Запрос к базе данных означает получение данных из базы, обычно из таблицы или нескольких таблиц.

Можно запросить представление вместо таблицы. Представления выглядят как таблицы и являются производными от них, но сами не содержат никаких данных. Дополнительную информацию о представлениях можно найти в разделе «Представления» главы 5.

Предложение SELECT

В предложении `SELECT` указываются столбцы, которые должен возвращать оператор.

В предложении `SELECT` за ключевым словом `SELECT` следует список имен столбцов и/или выражений, разделенных запятыми. Каждое имя столбца и/или выражение становится столбцом в результатах.

Выбор столбцов

В простейшем предложении SELECT перечисляются имена одного или нескольких столбцов из таблиц, указанных в предложении FROM:

```
SELECT id, name
FROM owner;

id     name
-----  ----------------
    1  Pictured Rocks
    2  Michigan Nature
    3  AF LLC
    4  MI DNR
    5  Horseshoe Falls
```

Выбор всех столбцов

Чтобы вернуть все столбцы из таблицы, можно не записывать имя каждого столбца, а использовать одну звездочку (*):

```
SELECT *
FROM owner;

id     name              phone          type
-----  ----------------  -------------  --------
    1  Pictured Rocks    906.387.2607   public
    2  Michigan Nature   517.655.5655   private
    3  AF LLC private
    4  MI DNR            906.228.6561   public
    5  Horseshoe Falls   906.387.2635   private
```

Звездочка — полезное сокращение при тестировании запросов, так как позволяет вам сэкономить много времени на вводе текста. Однако использовать ее в производственном коде рискованно: столбцы таблицы могут меняться со временем, и это приведет к сбою кода, если их окажется меньше или больше, чем ожидалось.

Выбор выражений

Помимо простого перечисления столбцов, в предложении SELECT можно перечислить и более сложные выражения, которые будут возвращаться в качестве столбцов в результатах.

Следующий оператор содержит выражение для расчета сокращения численности населения на 10 %, округленное до нуля знаков после запятой:

```
SELECT name, ROUND(population * 0.9, 0)
FROM county;

name         ROUND(population * 0.9, 0)
----------   --------------------------
Alger                              8876
Baraga                             7871
Ontonagon                          7036
...
```

Выбор функций

Выражения в списке SELECT обычно ссылаются на столбцы таблиц, из которых производится выборка, но бывают и исключения. Например, широко используемой функцией, которая не ссылается ни на какие таблицы, является функция, возвращающая текущую дату:

```
SELECT CURRENT_DATE;

CURRENT_DATE
-------------
2021-12-01
```

Приведенный выше код работает в MySQL, PostgreSQL и SQLite.

Эквивалентный код, работающий в других РСУБД, можно найти в разделе «Функции даты и времени» главы 7.

Большинство запросов содержат как предложение SELECT, так и предложение FROM, но при использовании определенных функций базы данных, таких как CURRENT_DATE, требуется только SELECT.

Кроме того, в предложение `SELECT` можно добавлять выражения, которые являются *подзапросами* (запрос, вложенный в другой запрос). Более подробная информация приведена в подразделе «Выбор подзапросов» далее в этой главе.

Псевдонимы столбцов

Псевдоним столбца предназначен для того, чтобы дать временное имя любому столбцу или выражению, перечисленному в предложении `SELECT`. Это временное имя, или псевдоним столбца, затем отображается как имя столбца в результатах.

Обратите внимание, что такое изменение имени не является постоянным, поскольку имена столбцов в исходных таблицах остаются прежними. Псевдоним существует только внутри запроса.

Данный код выводит на экран три столбца:

```
SELECT id, name,
ROUND(population * 0.9, 0)
FROM county;

id     name         ROUND(population * 0.9, 0)
-----  -----------  --------------------------
    2  Alger                              8876
    6  Baraga                             7871
    7  Ontonagon                          7036
...
```

Допустим, мы хотим переименовать названия столбцов в результатах. Имя `id` — слишком неоднозначное, и мы хотели бы дать более описательное. А `ROUND(population * 0.9, 0)` — слишком длинное, и мы хотели бы дать столбцу более простое имя.

Чтобы создать псевдоним столбца, после имени столбца или выражения нужно добавить либо имя псевдонима, либо ключевое слово `AS` и имя псевдонима:

```
-- alias_name
SELECT id county_id, name,
       ROUND(population * 0.9, 0) estimated_pop
FROM county;
```

или:

```
-- AS alias_name
SELECT id AS county_id, name,
       ROUND(population * 0.90, 0) AS estimated_pop
FROM county;

county_id   name        estimated_pop
----------  ----------  --------------
         2  Alger                 8876
         6  Baraga                7871
         7  Ontonagon             7036
...
```

На практике при создании псевдонимов используются оба варианта. В случае предложения `SELECT` второй вариант более популярен, поскольку ключевое слово `AS` облегчает визуальный поиск имен столбцов и псевдонимов среди длинного списка имен столбцов.

Более старые версии PostgreSQL требуют использования AS при создании псевдонима столбца.

Псевдонимы столбцов не являются обязательными, однако настоятельно рекомендуется использовать их при работе с выражениями, чтобы дать адекватные имена столбцам в результатах.

Псевдонимы с учетом регистра и пунктуации

Как видно на примере псевдонимов столбцов `county_id` и `estimated_pop`, при именовании псевдонимов столбцов принято использовать строчные буквы с подчеркиванием вместо пробелов.

Вдобавок можно создавать псевдонимы, содержащие заглавные буквы, пробелы и знаки препинания, используя синтаксис двойных кавычек, как показано в данном примере:

```
SELECT id AS "Waterfall #",
  name AS "Waterfall Name"
FROM waterfall;

Waterfall #   Waterfall Name
-----------   --------------
          1   Munising Falls
          2   Tannery Falls
          3   Alger Falls
...
```

Уточнение столбцов

Допустим, вы пишете запрос, который извлекает данные из двух таблиц, и обе они содержат столбец `name`. Если бы вы просто включили `name` в предложение `SELECT`, то код не знал бы, к какой таблице вы обращаетесь.

Для решения этой проблемы можно *уточнить* имя столбца, указав имя его таблицы. Другими словами, можно присвоить столбцу префикс, указывающий, к какой таблице он принадлежит, используя *точечную нотацию*, как, например, `table_name.column_name`.

В примере ниже выполняется запрос к одной таблице, и хотя здесь нет необходимости определять столбцы, это показано для демонстрации. Вот как следует уточнить столбец по имени таблицы:

```
SELECT owner.id, owner.name
FROM owner;
```

Если в SQL возникает ошибка, связанная с неоднозначным именем столбца, это означает, что в нескольких таблицах в запросе есть столбец с одним и тем же именем и вы не указали, к какой комбинации таблиц и столбцов обращаетесь. Устранить ошибку можно, уточнив имя столбца.

Уточнение таблиц

Если вы уточняете имя столбца по имени таблицы, то можете также определить ее имя по имени ее базы данных или схемы. Этот запрос извлекает данные именно из таблицы `owner` в схеме `sqlbook`:

```
SELECT sqlbook.owner.id, sqlbook.owner.name
FROM sqlbook.owner;
```

Код получился длинным, так как `sqlbook.owner` повторяется несколько раз. Чтобы сэкономить на вводе текста, можно указать *псевдоним таблицы*. В следующем примере таблице `owner` присвоен псевдоним `o`:

```
SELECT o.id, o.name
FROM sqlbook.owner o;
```

или:

```
SELECT o.id, o.name
FROM owner o;
```

СРАВНЕНИЕ ПСЕВДОНИМОВ СТОЛБЦОВ С ПСЕВДОНИМАМИ ТАБЛИЦ

Псевдонимы столбцов определяются в предложении `SELECT` в целях переименования столбца в результатах. Обычно содержат ключевое слово `AS`, хотя это и не обязательно.

```
-- Псевдоним столбца
SELECT num AS new_col
FROM my_table;
```

Псевдонимы таблиц определяются в предложении `FROM` в целях создания временного псевдонима для таблицы. Обычно принято исключать ключевое слово `AS`, хотя работает и его добавление.

```
-- Псевдоним таблицы
SELECT *
FROM my_table mt;
```

Выбор подзапросов

Подзапрос — это запрос, вложенный в другой запрос. Подзапросы могут располагаться в различных предложениях, в том числе и в `SELECT`.

В следующем примере, помимо `id`, `name` и `population`, допустим, мы хотим увидеть еще и среднюю численность населения всех округов. Добавив подзапрос, мы создадим в результатах новый столбец для средней численности населения:

```
SELECT id, name, population,
       (SELECT AVG(population) FROM county)
       AS average_pop
FROM county;

id     name         population   average_pop
-----  ----------  -----------  ------------
    2  Alger              9862         18298
    6  Baraga             8746         18298
    7  Ontonagon          7818         18298
...
```

Здесь следует отметить несколько моментов.

- Подзапрос должен быть заключен в круглые скобки.
- При написании подзапроса в предложении `SELECT` настоятельно рекомендуется указывать псевдоним столбца, которым в данном случае является `average_pop`. Таким образом, столбец в результатах будет иметь простое имя.
- В столбце `average_pop` есть только одно значение, которое повторяется во всех строках. При добавлении подзапроса в предложение `SELECT` результат подзапроса должен возвращать один столбец и либо ноль, либо одну строку, как показано в этом подзапросе для расчета средней численности населения:

  ```
  SELECT AVG(population) FROM county;

  AVG(population)
  ---------------
            18298
  ```

- Если подзапрос вернул нулевые строки, то новый столбец будет заполнен значениями NULL.

НЕКОРРЕЛИРОВАННЫЕ И КОРРЕЛИРОВАННЫЕ ПОДЗАПРОСЫ

Предыдущий пример представляет собой *некоррелированный подзапрос*, то есть подзапрос не ссылается на внешний запрос. Подзапрос может выполняться самостоятельно, независимо от внешнего запроса.

Другой тип подзапроса называется *коррелированным*, и он действительно ссылается на значения во внешнем запросе. Как следствие, время обработки часто значительно замедляется, поэтому лучше переписать запрос, используя вместо него JOIN. Ниже приведен пример коррелированного подзапроса, а также более эффективный код.

Проблемы производительности при использовании коррелированных подзапросов

Запрос, показанный ниже, возвращает количество водопадов для каждого владельца. Обратите внимание: шаг `o.id = w.owner_id` в подзапросе ссылается на таблицу `owner` во внешнем запросе, что делает его коррелированным подзапросом.

```
SELECT o.id, o.name,
       (SELECT COUNT(*) FROM waterfall w
       WHERE o.id = w.owner_id) AS num_waterfalls
FROM owner o;

id     name              num_waterfalls
-----  ----------------  --------------
    1  Pictured Rocks                 3
    2  Michigan Nature                3
    3  AF LLC                         1
    4  MI DNR                         1
    5  Horseshoe Falls                0
```

Более правильным подходом было бы переписать запрос с помощью JOIN. В этом случае сначала соединяются таблицы,

а затем выполняется остальная часть запроса, что гораздо быстрее, чем повторное выполнение подзапроса для каждой строки данных. Дополнительную информацию о соединениях можно найти в разделе «Соединение таблиц» главы 9.

```
SELECT   o.id, o.name,
         COUNT(w.id) AS num_waterfalls
FROM     owner o LEFT JOIN waterfalls w
         ON o.id = w.owner_id
GROUP BY o.id, o.name

id     name              num_waterfalls
-----  ----------------  ---------------
    1  Pictured Rocks                  3
    2  Michigan Nature                 3
    3  AF LLC                          1
    4  MI DNR                          1
    5  Horseshoe Falls                 0
```

Ключевое слово DISTINCT

Если в предложении `SELECT` указан столбец, то по умолчанию возвращаются все строки. Для большей ясности можно добавить ключевое слово `ALL`, но это не обязательно. Эти запросы возвращают каждую комбинацию `type/open_to_public`:

```
SELECT o.type, w.open_to_public
FROM owner o
JOIN waterfall w ON o.id = w.owner_id;
```

или:

```
SELECT ALL o.type, w.open_to_public
FROM owner o
JOIN waterfall w ON o.id = w.owner_id;

type      open_to_public
--------  ---------------
public    y
public    y
public    y
private   y
private   y
```

```
private   y
private   y
public    y
```

Если необходимо удалить из результатов повторяющиеся строки, то можно воспользоваться ключевым словом `DISTINCT`. Этот запрос возвращает список уникальных комбинаций `type/open_to_public`:

```
SELECT DISTINCT o.type, w.open_to_public
FROM owner o
JOIN waterfall w ON o.id = w.owner_id;

type      open_to_public
--------  ---------------
public    y
private   y
```

COUNT и DISTINCT

Чтобы подсчитать количество уникальных значений в *одном столбце*, можно объединить ключевые слова `COUNT` и `DISTINCT` в предложении `SELECT`. Этот запрос возвращает количество уникальных значений столбца `type`:

```
SELECT COUNT(DISTINCT type) AS unique
FROM owner;

unique
-------
      2
```

Чтобы подсчитать количество уникальных комбинаций нескольких столбцов, можно обернуть запрос `DISTINCT` в подзапрос, а затем выполнить `COUNT` для подзапроса. Этот запрос возвращает количество уникальных комбинаций `type/open_to_public`:

```
SELECT COUNT(*) AS num_unique
FROM (SELECT DISTINCT o.type, w.open_to_public
     FROM owner o JOIN waterfall w
     ON o.id = w.owner_id) my_subquery;
```

```
num_unique
-----------
          2
```

MySQL и PostgreSQL поддерживают использование синтаксиса `COUNT(DISTINCT)` для нескольких столбцов. Два этих запроса эквивалентны предыдущему запросу и не требуют подзапроса:

```
-- эквивалент MySQL
SELECT COUNT(DISTINCT o.type, w.open_to_public)
       AS num_unique
       FROM owner o JOIN waterfall w
            ON o.id = w.owner_id;

-- эквивалент PostgreSQL
SELECT COUNT(DISTINCT (o.type, w.open_to_public))
       AS num_unique
       FROM owner o JOIN waterfall w
            ON o.id = w.owner_id;

num_unique
-----------
          2
```

Предложение FROM

С помощью предложения `FROM` вы можете указать источник данных, которые хотите получить. В простейшем случае в предложении `FROM` запроса указывается одна таблица или представление.

```
SELECT name
FROM waterfall;
```

Вы можете уточнить таблицу или представление, указав имя базы данных или схемы с помощью точечной нотации. Этот запрос извлекает данные конкретно из таблицы `waterfall` в схеме `sqlbook`:

```
SELECT name
FROM sqlbook.waterfall;
```

Получение данных из нескольких таблиц

Вместо того чтобы получать данные из одной таблицы, часто требуется объединить данные из нескольких. Самый распространенный способ сделать это — использовать предложение `JOIN` внутри предложения `FROM`. Этот запрос извлекает данные из таблиц `Waterfall` и `Tour` и выводит одну таблицу результатов:

```
SELECT *
FROM waterfall w JOIN tour t
     ON w.id = t.stop;

id     name             ... name       stop  ...
-----  ---------------      ---------  -----
    1  Munising Falls       M-28           1
    1  Munising Falls       Munising       1
    2  Tannery Falls        Munising       2
    3  Alger Falls          M-28           3
    3  Alger Falls          Munising       3
...
```

Разберем каждую часть блока кода.

Псевдонимы таблиц

```
waterfall w JOIN tour t
```

Таблицам `Waterfall` и `Tour` присвоены псевдонимы `w` и `t`, которые являются временными именами таблиц в запросе. Псевдонимы таблиц необязательны в предложении `JOIN`, но очень помогают сокращать имена таблиц, на которые необходимо ссылаться в предложениях `ON` и `SELECT`.

JOIN ... ON ...

```
waterfall w JOIN tour t
ON w.id = t.stop
```

Эти две таблицы соединяются с помощью ключевого слова `JOIN`. За предложением `JOIN` всегда следует предложение `ON`,

которое определяет, как таблицы должны быть связаны друг с другом. В данном случае идентификатор водопада в таблице `Waterfall` должен совпадать со значением остановки с водопадом в таблице `Tour`.

Вы можете увидеть, что предложения FROM, JOIN и ON расположены на разных строках или с отступом. Это не обязательно, но улучшает удобочитаемость, особенно при соединении большого количества таблиц.

Таблица результатов

Результатом запроса всегда является одна таблица. В таблице `waterfall` 12 столбцов, а в таблице `tour` — три. После соединения таблица результатов содержит 15 столбцов.

```
id     name            ... name       stop ...
-----  --------------      ---------  -----
    1 Munising Falls       M-28           1
    1 Munising Falls       Munising       1
    2 Tannery Falls        Munising       2
    3 Alger Falls          M-28           3
    3 Alger Falls          Munising       3
...
```

Вы заметите, что в таблице результатов есть два столбца с именем `name`. Первый — из таблицы `Waterfall`, а второй — из таблицы `Tour`. Чтобы сослаться на них в предложении `SELECT`, необходимо уточнить имена столбцов.

```
SELECT w.name, t.name
FROM waterfall w JOIN tour t
     ON w.id = t.stop;

name             name
---------------  ---------
Munising Falls   M-28
Munising Falls   Munising
Tannery Falls    Munising
...
```

Чтобы различать эти два столбца, необходимо присвоить псевдонимы именам столбцов.

```
SELECT w.name AS waterfall_name,
       t.name AS tour_name
FROM waterfall w JOIN tour t
     ON w.id = t.stop;

waterfall_name   tour_name
---------------  ----------
Munising Falls   M-28
Munising Falls   Munising
Tannery Falls    Munising
Alger Falls      M-28
Alger Falls      Munising
...
```

Варианты JOIN

В предыдущем примере, если `waterfall` не указана ни в одном туре, то не появится в таблице результатов. Если вы хотите видеть в результатах данные обо всех водопадах, то необходимо использовать другой тип соединения.

> **ПО УМОЛЧАНИЮ JOIN ОЗНАЧАЕТ INNER JOIN**
>
> В этом примере используется простое ключевое слово `JOIN` для объединения данных из двух таблиц, хотя лучше всего явно указывать тип используемого соединения. Само по себе `JOIN` по умолчанию является `INNER JOIN`; это означает, что в результатах будут возвращены только те записи, которые содержатся в обеих таблицах.

В SQL используются различные типы соединений; о них мы более подробно поговорим в разделе «Соединение таблиц» главы 9.

Получение данных из подзапросов

Подзапрос — это запрос, вложенный в другой запрос. Подзапросы в предложении `FROM` должны быть автономными операторами `SELECT`, то есть они не ссылаются на внешний запрос и могут выполняться самостоятельно.

Подзапрос в предложении FROM также известен как производная таблица, поскольку в конечном итоге подзапрос фактически действует как таблица в течение всего времени выполнения запроса.

В этом запросе перечислены все водопады, находящиеся в публичной собственности, причем часть, содержащая подзапрос, выделена жирным шрифтом:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM (SELECT * FROM owner WHERE type = 'public') o
     JOIN waterfall w
     ON o.id = w.owner_id;

waterfall_name   owner_name
--------------   --------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

Важно понимать, в каком порядке выполняется запрос.

Этап 1. Выполнение подзапроса

Сначала выполняется содержимое подзапроса. Видно, что в результате получается таблица, содержащая только информацию о типе владельцев `public`:

```
SELECT * FROM owner WHERE type = 'public';

id     name             phone          type
-----  ---------------  -------------  -------
    1  Pictured Rocks   906.387.2607   public
    4  MI DNR           906.228.6561   public
```

Если вернуться к исходному запросу, то можно заметить, что за подзапросом сразу следует буква `o`. Это временное имя (или псевдоним), которое присваивается результатам подзапроса.

Псевдонимы необходимы для подзапросов в предложении FROM в MySQL, PostgreSQL и SQL Server, но не в Oracle и SQLite.

Этап 2. Выполнение всего запроса

Далее можно представить, что на месте подзапроса стоит буква o. Теперь запрос выполняется как обычно:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM o JOIN waterfall w
     ON o.id = w.owner_id;

waterfall_name   owner_name
--------------   --------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

ПОДЗАПРОСЫ В СРАВНЕНИИ С ПРЕДЛОЖЕНИЕМ WITH

В качестве альтернативы написанию подзапроса можно использовать написание обобщенного табличного выражения (common table expression, CTE) с помощью предложения WITH. Преимущество предложения WITH заключается в том, что подзапрос именуется заранее, в результате чего код получается более чистым, а также появляется возможность ссылаться на подзапрос несколько раз.

```
WITH o AS (SELECT * FROM owner
           WHERE type = 'public')

SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM o JOIN waterfall w
     ON o.id = w.owner_id;
```

Предложение WITH поддерживается в MySQL 8.0+ (2018 и более поздние версии), PostgreSQL, Oracle, SQL Server и SQLite. В разделе «Обобщенные табличные выражения» главы 9 приведены дополнительные примеры использования этой техники.

Зачем использовать подзапрос в предложении FROM

Основное преимущество использования подзапросов заключается в том, что можно превратить большую задачу в более мелкие. Приведу два примера.

- *Пример 1. Несколько этапов для достижения результатов.* Допустим, вы хотите найти среднее количество остановок в туре. Сначала нужно будет найти количество остановок в каждом туре, а затем усреднить результаты.

  Количество остановок в каждом туре можно найти с помощью этого запроса:

```
SELECT name, MAX(stop) as num_stops
FROM tour
GROUP BY name;

name       num_stops
---------  ----------
M-28               11
Munising            6
US-2               14
```

  Затем можно превратить этот запрос в подзапрос и написать вокруг него еще один запрос для нахождения среднего значения:

```
SELECT AVG(num_stops) FROM
(SELECT name, MAX(stop) as num_stops
FROM tour
GROUP BY name) tour_stops;

AVG(num_stops)
-----------------
10.3333333333333
```

- *Пример 2. Таблица в предложении FROM слишком велика.* Первоначальная цель состояла в том, чтобы перечислить все водопады, находящиеся в государственной собствен-

ности. На самом деле это можно сделать без подзапроса и вместо этого использовать `JOIN`:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM   owner o
JOIN   waterfall w ON o.id = w.owner_id
WHERE  o.type = 'public';

waterfall_name   owner_name
--------------   --------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

Предположим, что выполнение запроса занимает очень много времени. Это может произойти при соединении больших таблиц (например, десятки миллионов строк). Есть несколько способов переписать запрос, чтобы ускорить его выполнение, и одним из них является использование подзапроса.

Поскольку нас интересуют только владельцы типа `public`, мы можем сначала написать подзапрос, который отфильтрует всех частных владельцев. Затем уменьшенная таблица владельцев будет объединена с таблицей `waterfall`, что займет меньше времени и даст те же результаты:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM   (SELECT * FROM owner
       WHERE type = 'public') o
       JOIN waterfall w ON o.id = w.owner_id;

waterfall_name   owner_name
--------------   --------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

Это лишь два из множества примеров того, как с помощью подзапросов разбивать большой запрос на более мелкие шаги.

Предложение WHERE

Предложение `WHERE` позволяет ограничить результаты запроса только интересующими нас строками, или, проще говоря, служит для фильтрации данных. В редких случаях требуется отобразить все строки таблицы, чаще речь о строках, соответствующих определенным критериям.

При исследовании таблицы, содержащей миллионы строк, никогда не следует выполнять команду SELECT * FROM my_table;, поскольку ее выполнение займет неоправданно много времени.

Вместо этого целесообразно отфильтровать данные. Сделать это можно с помощью двух распространенных способов.

- Фильтрация по столбцу в предложении WHERE.

  Еще лучше отфильтровать по уже проиндексированному столбцу, чтобы еще больше ускорить поиск.

  ```
  SELECT *
  FROM my_table
  WHERE year_id = 2021;
  ```

- Покажите несколько первых строк данных с помощью предложения LIMIT (или WHERE ROWNUM <= 10 в Oracle или SELECT TOP 10 * в SQL Server).

  ```
  SELECT *
  FROM my_table
  LIMIT 10;
  ```

Показанный ниже запрос находит информацию обо всех водопадах, в названии которых нет слова `Falls`. Более подробно о ключевом слове `LIKE` можно узнать в главе 7.

```
SELECT id, name
FROM waterfall
WHERE name NOT LIKE '%Falls%';

id     name
-----  ----------------
    7  Little Miners
   14  Rapid River Fls
```

Выделенный жирным шрифтом раздел часто называют условным оператором или предикатом. Предикат выполняет логическое сравнение для каждой строки данных, результатом которого является TRUE/FALSE/UNKNOWN.

Таблица waterfall содержит 16 строк. Для каждой строки проверяется, содержит ли она название водопада *Falls*. Если нет, то предикат name NOT LIKE '%Falls%' имеет значение TRUE и строка возвращается в результаты, как это было для двух предыдущих строк.

Множественные предикаты

Можно также объединять несколько предикатов с помощью операции типа AND или OR. В примере ниже показана информация о водопадах, в названии которых отсутствует слово *Falls* и которые к тому же не имеют владельца:

```
SELECT id, name
FROM waterfall
WHERE name NOT LIKE '%Falls%'
      AND owner_id IS NULL;

id     name
-----  ----------------
   14  Rapid River Fls
```

Более подробную информацию об операциях можно найти в разделе «Операции» главы 7.

Фильтрация по подзапросам

Подзапрос — это запрос, вложенный в другой запрос, и обычно он находится в предложении WHERE. В примере ниже извлекается информация об общедоступных водопадах, расположенных в округе Alger:

```
SELECT w.name
FROM   waterfall w
```

```
WHERE  w.open_to_public = 'y'
       AND w.county_id IN (
           SELECT c.id FROM county c
           WHERE c.name = 'Alger');

name
---------------
Munising Falls
Tannery Falls
Alger Falls
...
```

В отличие от подзапросов в предложениях SELECT или FROM, подзапросы в предложении WHERE не требуют псевдонима. Более того, при добавлении псевдонима будет выдана ошибка.

Зачем использовать подзапрос в предложении WHERE

Первоначальной целью запроса был поиск информации об общедоступных водопадах, расположенных в округе Alger. Если бы вы писали этот запрос с нуля, то, скорее всего, начали бы со следующего:

```
SELECT w.name
FROM   waterfall w
WHERE  w.open_to_public = 'y';
```

На текущий момент у вас есть данные обо всех общедоступных водопадах. Осталось найти информацию о тех, которые находятся конкретно в округе Alger. Вы знаете, что в таблице `waterfall` нет столбца с названием округа, но в таблице округов он есть.

У вас есть два варианта добавления названия округа в результаты. Можно либо написать подзапрос в предложении `WHERE`, который специально извлекает информацию об округе Alger, либо соединить таблицы `waterfall` и `county`:

```
-- Подзапрос в предложении WHERE
SELECT w.name
FROM   waterfall w
```

```
WHERE  w.open_to_public = 'y'
       AND w.county_id IN (
           SELECT c.id FROM county c
           WHERE c.name = 'Alger');
```

или:

```
-- в предложении JOIN
SELECT w.name
FROM   waterfall w INNER JOIN county c
       ON w.county_id = c.id
WHERE  w.open_to_public = 'y'
       AND c.name = 'Alger';

name
---------------
Munising Falls
Tannery Falls
Alger Falls
...
```

Эти два запроса дают одинаковые результаты. Преимущество первого подхода заключается в том, что подзапросы зачастую проще понять, чем соединение. Преимущество второго подхода состоит в том, что соединение обычно выполняется быстрее, чем подзапросы.

РАБОТА ▸ ОПТИМИЗАЦИЯ

При написании SQL-кода выполнить одно и то же действие часто можно несколькими способами.

Вашим главным приоритетом должно быть написание *работающего* кода. Пусть он долго выполняется или выглядит некрасиво. Это неважно, поскольку он работает!

На следующем этапе, если у вас есть время, вы можете *оптимизировать* код, улучшив производительность. Возможно, вы перепишете его с помощью JOIN, сделаете более читабельным с помощью отступов и заглавных букв и т. д.

Не зацикливайтесь на создании наиболее оптимизированного кода заранее, а лучше напишите код, который работает. Написание элегантного кода приходит с опытом.

Другие способы фильтрации данных

Предложение `WHERE` — не единственное место в операторе `SELECT`, где можно фильтровать строки данных.

- Предложение `FROM`: при соединении таблиц в предложении `ON` указывается, как они должны быть связаны между собой. Здесь можно внести условия по ограничению строк данных, возвращаемых запросом. Более подробную информацию см. в разделе «Соединение таблиц» главы 9.
- Предложение `HAVING`: если в операторе `SELECT` есть агрегаты, то в предложении `HAVING` указывается, как их фильтровать. Более подробную информацию см. в разделе «Предложение HAVING» далее в текущей главе.
- Предложение `LIMIT`: чтобы вывести на экран определенное количество строк, можно использовать предложение `LIMIT`. В Oracle это делается с помощью `WHERE ROWNUM`, а в SQL Server — с помощью `SELECT TOP`. Более подробную информацию см. в разделе «Предложение LIMIT» далее в текущей главе.

Предложение GROUP BY

Цель предложения `GROUP BY` — собрать строки в группы и обобщить строки внутри групп каким-либо образом, в конечном итоге возвращая только одну строку на группу. Иногда это называют разрезанием строк на группы и свертыванием строк в каждой группе.

Этот запрос подсчитывает количество водопадов на каждом из маршрутов:

```
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
FROM     waterfall w INNER JOIN tour t
         ON w.id = t.stop
GROUP BY t.name;
```

```
tour_name num_waterfalls
----------  --------------
M-28                     6
Munising                 6
US-2                     4
```

Здесь следует сосредоточиться на двух составляющих:

- *сбор строк*, который задается в предложении `GROUP BY`;
- *суммирование строк* внутри групп, которое задается в предложении `SELECT`.

Этап 1. Сбор строк

В предложении `GROUP BY`:

```
GROUP BY t.name
```

мы указываем, что хотели бы просмотреть все строки данных и объединить в одну группу водопады из тура `M-28`, в другую — все водопады из тура `Munising` и т. д. Скрытым образом данные группируются так:

```
tour_name   waterfall_name
----------  ---------------
M-28        Munising Falls
M-28        Alger Falls
M-28        Scott Falls
M-28        Canyon Falls
M-28        Agate Falls
M-28        Bond Falls

Munising    Munising Falls
Munising    Tannery Falls
Munising    Alger Falls
Munising    Wagner Falls
Munising    Horseshoe Falls
Munising    Miners Falls

US-2        Bond Falls
US-2        Fumee Falls
US-2        Kakabika Falls
US-2        Rapid River Fls
```

Этап 2. Суммирование строк

В предложении SELECT:

```
SELECT t.name AS tour_name,
       COUNT(*) AS num_waterfalls
```

мы указываем, что для каждой группы или каждого тура хотим подсчитать количество строк данных в группе. Поскольку каждая строка представляет собой водопад, наше действие приведет к подсчету общего количества водопадов в каждом туре.

Приведенная здесь функция COUNT() более формально известна как *агрегатная*, или функция, которая суммирует множество строк данных в одно значение. Дополнительные агрегатные функции можно найти в разделе «Агрегатные функции» главы 7.

В данном примере COUNT(*) возвращает количество водопадов в каждом туре. Однако это происходит только потому, что каждая строка данных в таблицах waterfall и tour представляет собой один водопад.

Если один водопад был указан в нескольких строках, то COUNT(*) даст большее значение, чем ожидалось. В этом случае для поиска уникальных водопадов можно использовать COUNT(DISTINCT waterfall_name). Более подробную информацию можно найти в пункте «COUNT и DISTINCT» выше в текущей главе.

Основной вывод заключается в том, что важно вручную дважды перепроверять результаты работы агрегатной функции, чтобы убедиться, что она суммирует данные именно так, как вы задумали.

Теперь, когда группы созданы с помощью предложения GROUP BY, агрегатная функция будет применена к каждой группе один раз:

```
tour_name   COUNT(*)
----------  ---------
```

```
M-28                6
M-28
M-28
M-28
M-28
M-28

Munising            6
Munising
Munising
Munising
Munising
Munising

US-2                4
US-2
US-2
US-2
```

Все столбцы, к которым не была применена агрегатная функция, а в данном случае это столбец `tour_name`, теперь сворачиваются в одно значение:

```
tour_name   COUNT(*)
----------  ---------
M-28                6
Munising            6
US-2                4
```

Такое сворачивание множества детальных строк в одну агрегированную строку означает, что при использовании предложения GROUP BY в предложении SELECT должны содержаться только:

- все столбцы, перечисленные в предложении GROUP BY: t.name
- агрегации: COUNT(*)

```
SELECT t.name AS tour_name,
       COUNT(*) AS num_waterfalls
...
GROUP BY t.name;
```

Невыполнение этого требования может привести к появлению сообщения об ошибке или возврату неверных значений.

GROUP BY НА ПРАКТИКЕ

При использовании GROUP BY вы должны определить:

1) какие столбцы хотите использовать для разделения или группировки данных (например, название тура);
2) как вы хотите обобщить данные в каждой группе (например, подсчитать количество водопадов в каждом туре).

После этого:

1) в предложении SELECT перечислите столбцы, по которым нужно сгруппировать данные (например, название тура), и агрегат (-ы), который (-е) нужно вычислить в каждой группе (например, количество водопадов);
2) в предложении GROUP BY перечислите все столбцы, которые не являются агрегатами (например, название тура).

Более сложные ситуации группировки, включая ROLLUP, CUBE и GROUPING SETS, рассматриваются в разделе «Группировка и агрегирование» главы 8.

Предложение HAVING

Предложение HAVING накладывает ограничения на строки, возвращаемые в результате выполнения запроса GROUP BY. Другими словами, оно позволяет фильтровать результаты после применения GROUP BY.

Предложение HAVING всегда следует непосредственно за предложением GROUP BY. Без GROUP BY не может быть HAVING.

Это запрос, который перечисляет количество водопадов в каждом туре, используя предложение GROUP BY:

```
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
```

```
FROM     waterfall w INNER JOIN tour t
         ON w.id = t.stop
GROUP BY t.name;

tour_name   num_waterfalls
----------  --------------
M-28                     6
Munising                 6
US-2                     4
```

Допустим, мы хотим перечислить только те туры, в которых ровно шесть остановок. Для этого необходимо добавить предложение HAVING после предложения GROUP BY:

```
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
FROM     waterfall w INNER JOIN tour t
         ON w.id = t.stop
GROUP BY t.name
HAVING COUNT(*) = 6;


tour_name   num_waterfalls
----------  --------------
M-28                     6
Munising                 6
```

СРАВНЕНИЕ WHERE С HAVING

Целью обоих предложений является фильтрация данных. Если вы пытаетесь:

- фильтровать по определенным столбцам, то пропишите условия в предложении WHERE;
- фильтровать по агрегатам, то записывайте условия в предложение HAVING.

Содержимое предложений WHERE и HAVING нельзя поменять местами:

- никогда не помещайте условие с агрегированием в предложение WHERE. Это приведет к ошибке;
- никогда не помещайте в предложение HAVING условия, не связанные с агрегированием. Такие условия гораздо эффективнее вычисляются в предложении WHERE.

Вы заметите, что предложение HAVING относится к агрегированию COUNT(*):

```
SELECT COUNT(*) AS num_waterfalls
...
HAVING COUNT(*) = 6;
```

а не к псевдониму:

```
# код не будет выполняться
SELECT COUNT(*) AS num_waterfalls
...
HAVING num_waterfalls = 6;
```

Причина этого заключается в порядке выполнения предложений. Предложение SELECT записывается перед предложением HAVING. Однако на самом деле предложение SELECT выполняется *после* предложения HAVING.

Это означает, что псевдоним num_waterfalls в предложении SELECT не существует на момент выполнения предложения HAVING. Вместо этого в предложении HAVING должна быть ссылка на необработанную агрегацию COUNT(*).

Исключение составляют MySQL и SQLite, которые допускают использование псевдонимов (num_waterfalls) в предложении HAVING.

Предложение ORDER BY

Предложение ORDER BY используется для указания порядка сортировки результатов запроса.

Этот запрос возвращает список владельцев и водопадов без какой-либо сортировки:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,
       w.name AS waterfall_name
FROM   waterfall w
       LEFT JOIN owner o ON w.owner_id = o.id;
```

```
owner             waterfall_name
----------------  ---------------
Pictured Rocks    Munising Falls
Michigan Nature   Tannery Falls
AF LLC            Alger Falls
MI DNR            Wagner Falls
Unknown           Horseshoe Falls
...
```

ФУНКЦИЯ COALESCE

Функция COALESCE заменяет все NULL-значения в столбце на другое значение. В данном случае она превратила NULL-значения в столбце `o.name` в текст `'Unknown'`.

Если бы здесь не использовалась функция COALESCE, то все водопады, не имеющие владельцев, были бы исключены из результатов. Вместо этого они теперь они помечены как имеющие владельца `'Unknown'` и могут быть отсортированы и добавлены в результаты.

Более подробную информацию можно найти в главе 7.

Этот запрос возвращает тот же список, но сначала отсортированный в алфавитном порядке по владельцу, а затем по водопаду:

```
SELECT   COALESCE(o.name, 'Unknown') AS owner,
         w.name AS waterfall_name
FROM     waterfall w
         LEFT JOIN owner o ON w.owner_id = o.id
ORDER BY owner, waterfall_name;
```

```
owner             waterfall_name
----------------  ---------------
AF LLC            Alger Falls
MI DNR            Wagner Falls
Michigan Nature   Tannery Falls
Michigan Nature   Twin Falls #1
Michigan Nature   Twin Falls #2
...
```

По умолчанию сортировка производится по возрастанию, то есть текст будет располагаться от A до Z, а числа — от меньшего к большему. Управлять сортировкой каждого столбца можно с помощью ключевых слов ASCENDING и DESCENDING (сокращенно ASC и DESC).

Ниже приведена предыдущая сортировка, но с изменениями, и на этот раз она сортирует имена владельцев в обратном порядке:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,
       w.name AS waterfall_name
...
ORDER BY owner DESC, waterfall_name ASC;

owner             waterfall_name
----------------  ---------------
Unknown           Agate Falls
Unknown           Bond Falls
Unknown           Canyon Falls
...
```

Вы можете сортировать по столбцам и выражениям, которые не входят в список SELECT:

```
SELECT   COALESCE(o.name, 'Unknown') AS owner,
         w.name AS waterfall_name
FROM     waterfall w
         LEFT JOIN owner o ON w.owner_id = o.id
ORDER BY o.id DESC, w.id;

owner             waterfall_name
----------------  ---------------
MI DNR            Wagner Falls
AF LLC            Alger Falls
Michigan Nature   Tannery Falls
...
```

Можно выполнять сортировку и по номеру позиции столбца:

```
SELECT COALESCE(o.name, 'Unknown') AS owner,
       w.name AS waterfall_name
...
ORDER BY 1 DESC, 2 ASC;
```

```
owner             waterfall_name
----------------  ---------------
Unknown           Agate Falls
Unknown           Bond Falls
Unknown           Canyon Falls
...
```

Поскольку строки таблицы SQL не упорядочены, если не включить в запрос предложение ORDER BY, то при каждом выполнении запроса результаты могут отображаться в разном порядке.

> **ORDER BY НЕЛЬЗЯ ИСПОЛЬЗОВАТЬ В ПОДЗАПРОСЕ**
>
> Из шести основных предложений только ORDER BY не может быть использовано в подзапросе. К сожалению, нельзя принудительно упорядочить строки подзапроса.
>
> Во избежание этой проблемы необходимо переписать запрос, используя другую логику, чтобы не использовать предложение ORDER BY в подзапросе, а добавить ORDER BY только во внешний запрос.

Предложение LIMIT

При быстром просмотре таблицы лучше всего возвращать не всю таблицу, а ограниченное количество строк.

MySQL, PostgreSQL и SQLite поддерживают предложение LIMIT. Oracle и SQL Server используют другой синтаксис при той же функциональности:

```
-- MySQL, PostgreSQL и SQLite
SELECT *
FROM owner
LIMIT 3;

-- Oracle
SELECT *
FROM owner
WHERE ROWNUM <= 3;
```

```
-- SQL Server
SELECT TOP 3 *
FROM owner;

id   name              phone          type
---  ----------------  -------------  --------
  1  Pictured Rocks    906.387.2607   public
  2  Michigan Nature   517.655.5655   private
  3  AF LLC                           private
```

Другой способ ограничения количества возвращаемых строк — фильтрация по столбцу в предложении `WHERE`. Фильтрация будет выполняться еще быстрее, если столбец проиндексирован.