Базы данных

Лекция 09 – Основы SQL. Запросы

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2023

2024.04.02

Оглавление

Запросы

Запросом называется процесс или команда получения данных из базы данных.
В SQL запросы формулируются с помощью команды **SELECT**.
В общем виде команда SELECT записывается так:

```
[WITH with_queries]                          -- расширение Postgres
SELECT select_list                           -- список выбора
FROM table_expression [sort_specification]   -- откуда и условия
```

Самый простой запрос выглядит так:

```
SELECT * FROM table;
```

Если в базе данных есть таблица **table**, эта команда выдаст все строки с содержимым всех столбцов из **table**.

Метод выдачи результата определяет приложение-клиент, например, psql выведет на экране содержимое в текстовом виде.

Если список выборки задан как *, это означает, что запрос должен вернуть все столбцы табличного выражения.

WITH — способ указать выполнение дополнительных операторов в больших запросах.

Эти операторы называют *общими табличными выражениями* (Common Table Expressions, CTE). Обычно это определения временных таблиц в рамках только одного запроса.

Дополнительным оператором в предложении WITH может быть SELECT, INSERT, UPDATE, DELETE, а само предложение WITH присоединяется к основному оператору, которым также может быть SELECT, INSERT, UPDATE или DELETE.

В списке выбора можно также указать подмножество доступных столбцов или составить выражения с этими столбцами — если в **table** есть столбцы **a**, **b** и **c**, а **b** и **c** имеют числовой тип данных, можно выполнить следующий запрос:

```
SELECT a, b + c FROM table;
```

FROM table — это простейший тип табличного выражения, в котором читается одна таблица.

Табличные выражения могут быть сложными конструкциями из базовых таблиц, соединений и подзапросов.

Также можно опустить табличное выражение и использовать команду SELECT как калькулятор:

```
SELECT 3 * 4;
```

Бывает полезно, когда выражения в списке выборки возвращают меняющиеся результаты:

```
SELECT random();
```

Табличные выражения

```
[WITH with_queries]                              -- расширение Postgres
SELECT select_list                               -- список выборки
FROM table_expression [sort_specification]  -- откуда и условия
```

Табличное выражение вычисляет таблицу.

Это выражение содержит предложение FROM, за которым *могут следовать* необязательные предложения:

WHERE;

GROUP BY;

HAVING.

Простые табличные выражения ссылаются на «физическую» таблицу, ее иногда называют базовой, но в более сложных выражениях такие таблицы можно преобразовывать и комбинировать самыми разными способами.

Необязательные предложения WHERE, GROUP BY и HAVING в табличном выражении определяют последовательность преобразований, осуществляемых с данными таблицы, полученной в предложении FROM.

В результате этих преобразований образуется виртуальная таблица, строки которой передаются списку выборки (`select_list`), который вычисляет выходные строки запроса.

Предложение FROM

Предложение FROM образует таблицу из одной или нескольких ссылок на таблицы, разделённых запятыми.

```
FROM table_reference [, table_reference [, ...]]
```

Здесь ссылкой на таблицу **table_reference** может быть:
- имя таблицы (как вариант, с именем схемы);
- производная таблица, например подзапрос;
- соединение таблиц;
- произвольная комбинация вышеупомянутых вариантов.

Если в предложении FROM перечисляются несколько ссылок, для них применяется перекрёстное соединение (CROSS JOIN — декартово произведение их строк).

Список FROM преобразуется в промежуточную виртуальную таблицу, которая может пройти через преобразования WHERE, GROUP BY и HAVING, и в итоге определит результат табличного выражения.

Если в ссылке на таблицу указывается таблица, являющаяся родительской в иерархии наследования, в результате будут получены строки не только этой таблицы, но и всех её дочерних таблиц.

Чтобы выбрать строки только одной родительской таблицы, перед её именем нужно добавить ключевое слово ONLY. При этом будут получены только столбцы указанной таблицы — дополнительные столбцы дочерних таблиц в результат не попадут.

Чтобы явно указать, обработку всех дочерних таблиц, следует дописать после родительской *.

Соединение таблиц

Соединенная таблица — это таблица, полученная из двух других (реальных или производных от них) таблиц в соответствии с правилами соединения конкретного типа.

Общий синтаксис описания соединённой таблицы:

```
T1 join_type T2 [ join_condition ]
```

Соединения любых типов могут вкладываются друг в друга или объединяться — T1, и T2 могут быть результатами соединения.

Для однозначного определения порядка соединений предложения JOIN можно заключать в скобки. Если скобки отсутствуют, предложения JOIN обрабатываются слева направо.

Типы соединений

Перекрёстное соединение CROSS JOIN

```
T1 CROSS JOIN T2
```

Результирующую таблицу образуют все возможные сочетания строк из T1 и T2 (декартово произведение), а набор ее столбцов объединяет столбцы T1 со следующими за ними столбцами T2.

Если таблицы содержат N и M строк, соединённая таблица будет содержать N * M строк.

Запись

```
FROM T1 CROSS JOIN T2
```

эквивалентна

```
FROM T1 INNER JOIN T2 ON TRUE — все без исключения
```

Соединения с сопоставлениями строк

Внутреннее INNER и внешнее OUTER.

Внутреннее соединение

```
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
      ON логическое_выражение

T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
      USING ( список столбцов соединения )

T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
```

Ключевые слова INNER (внутреннее соединение) и OUTER (внешнее соединение) не являются обязательными во всех формах.

По умолчанию подразумевается INNER, а при указании LEFT, RIGHT и FULL — внешнее соединение.

Условие соединения указывается в предложении ON или USING, либо неявно задаётся ключевым словом NATURAL.

Условие соединения определяет, какие строки двух исходных таблиц считаются «соответствующими» друг другу.

Возможные типы соединений с сопоставлениями строк:

INNER JOIN
LEFT OUTER JOIN
RIGHT OUTER JOIN
FULL OUTER JOIN

INNER JOIN — для каждой строки R1 из T1 в результирующей таблице содержится строка для каждой строки из T2, удовлетворяющей условию соединения с R1.

```
SELECT * FROM (table-1 JOIN table-2)     --
  ON table-1.parameter=table-2.parameter -- условие соединения
  WHERE table-1.parameter IS ‘SomeData’  -- фильтрация нужного нам из соединения
```

Можно, например, запрашивать гитары конкретного бренда, при этом еще и выбирая дополнительное условие в духе количества струн.

```
Guitars (
  id         integer PRIMARY KEY,
  Name       text,
  brand_id   integer REFERENCES Brand, -- внешний ключ
  strings    intrger                   -- кол-во струн
)
Brand (
  id         integer PRIMARY KEY,
  Name       text
)
```

```
SELECT * FROM
Guitars JOIN Brand
ON Guitars.brand_id=Brand.id
WHERE Guitars.brand_id IS 'Cort' AND strings = 7
```

Внешние соединения

LEFT OUTER JOIN — сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из T1, которым не соответствуют никакие строки в T2, а вместо значений столбцов T2 вставляются NULL. В результирующей таблице всегда будет минимум одна строка для каждой строки из T1.

RIGHT OUTER JOIN — сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из T2, которым не соответствуют никакие строки в T1, а вместо значений столбцов T1 вставляются NULL. Это соединение является обратным к левому (LEFT JOIN): в результирующей таблице всегда будет минимум одна строка для каждой строки из T2.

FULL OUTER JOIN — сначала выполняется внутреннее соединение.

Затем в результат добавляются все строки из T1, которым не соответствуют никакие строки в T2, а вместо значений столбцов T2 вставляются NULL.

Затем в результат включаются все строки из T2, которым не соответствуют никакие строки в T1, а вместо значений столбцов T1 вставляются NULL.

Предложение ON

Предложение ON определяет наиболее общую форму условия соединения — в нём указываются выражения логического типа (предикат) — пара строк из T1 и T2 соответствуют друг другу, если выражение ON возвращает для них **`true`**.

USING — это сокращённая запись условия, полезная в ситуации, если с обеих сторон соединения столбцы имеют одинаковые имена. Она принимает список общих имён столбцов через запятую и формирует условие соединения с равенством этих столбцов.

Например, запись соединения **T1** и **T2** с **USING (a, b)** формирует условие:

```
ON T1.a = T2.a AND T1.b = T2.b.
```

Из вывода JOIN USING исключаются избыточные столбцы, поскольку оба сопоставленных столбца выводить не нужно — они содержат одинаковые значения.

Т.е. когда JOIN ON выдаёт все столбцы из T1, а за ними все столбцы из T2, JOIN USING выводит только один столбец для каждой пары, в указанном порядке, за ними все оставшиеся столбцы из T1 и, наконец, все оставшиеся столбцы T2.

NATURAL — сокращённая форма USING. Она образует список USING из всех имён столбцов, существующих в обеих входных таблицах. Как и с USING, эти столбцы оказываются в выходной таблице в единственном экземпляре.

Псевдонимы таблиц и столбцов

Таблицам и ссылкам на сложные таблицы в запросе можно дать временное имя, по которому к ним можно будет обращаться в рамках данного сложного запроса. Такое имя называется псевдонимом таблицы.

Псевдоним таблицы:

```
FROM table_reference [AS] alias
```

или

```
FROM table_reference alias
```

Ключевое слово **AS** является необязательным. Вместо **alias** может быть любой идентификатор.

Псевдонимы часто применяются для назначения коротких идентификаторов длинным именам таблиц с целью улучшения читаемости запросов:

```
SELECT * FROM some_very_long_table_name AS s
    JOIN another_fairly_long_name a
    ON s.id = a.num;
```

Псевдоним становится новым именем таблицы в рамках текущего запроса. После назначения псевдонима использовать исходное имя таблицы в другом месте запроса нельзя.

Таким образом, следующий запрос недопустим:

```
SELECT * FROM my_table AS m WHERE my_table.a > 5;    -- неправильно
```

Псевдонимы бывают необходимы, если таблица соединяется сама с собой:

```
SELECT * FROM people AS parent JOIN people AS child
    ON parent.id = child.parent_id;
```

Псевдонимы обязательно нужно назначать подзапросам

В случае неоднозначности определения псевдонимов можно использовать скобки.
В следующем примере первый оператор назначает псевдоним **b** второму экземпляру **my_table**, а второй оператор назначает псевдоним результату соединения:

```
SELECT * FROM my_table AS a CROSS JOIN my_table AS b ...
SELECT * FROM (my_table AS a CROSS JOIN my_table) AS b ...
```

Временные имена могут даваться не только таблицам, но и ее столбцам:

```
FROM table_reference [AS] alias ( column1 [, column2 [, ...]] )
```

Если псевдонимов столбцов оказывается меньше, чем фактически столбцов в таблице, остальные столбцы сохраняют свои исходные имена. Эта запись особенно полезна для «замкнутых» соединений (self-join) или подзапросов.

Запросы типа Self Join

Используются для объединения таблицы с ней самой таким образом, как будто это две разные таблицы. Для этой цели временно переименовываем одну из них через алиас.

```
CREATE TABLE Customers (
  id      PRIMARY KEY,
  Name    text, -- наименование заказчика
  Address text, -- адрес
  City    text, -- город
)
-----
1  AA   Addr_A  City_A
2  BB   Addr_B  City_A
3  CC   Addr_C  City_A
4  DD   Addr_D  City_B
5  EE   Addr_E  City_B
```

Задача — выбрать заказчиков из одного города в порядке названия городов.

```
SELECT A.Name AS Name1, B.Name AS Name2, A.City
FROM Customers A, Customers B
WHERE A.id <> B.id AND A.City = B.City ORDER BY A.City;
---------
AA BB City_A
AA CC City_A
BB CC City_A
DD EE City_B
```

Используется для запросов к иерархии, когда, например, требуется получить информацию о сотрудниках вместе с именами их менеджеров

```
CREATE TABLE employees (
    id         integer PRIMARY KEY,
    manager_id integer REFERENCES employees (id),
    name       text
)
 id | manager_id |   name
----+------------+-----------
  1 |            | big_boss
  2 |          1 | boss1
  3 |          1 | boss2
  4 |          2 | emp1
  5 |          2 | emp3
  6 |          3 | emp2
---------------

SELECT e1.name, e2.name
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.id;
 name  |   name
-------+-----------
 boss1 | big_boss
 boss2 | big_boss
 emp1  | boss1
 emp3  | boss1
 emp2  | boss2
```

Здесь используется SELF JOIN для объединения работников по общим **`manager_id`** и **`id`**.

Подзапросы

Подзапросы, образующие таблицы, должны заключаться в скобки и им обязательно должны назначаться псевдонимы:

```
FROM (SELECT * FROM table1) AS alias_name
```

Этот пример эквивалентен записи

```
FROM table1 AS alias_name
```

Если в подзапросе используются агрегирующие функции или группировка, возникают более сложные ситуации, которые нельзя свести к простому соединению.

Подзапросом может также быть список VALUES, которая часто используется для создания «таблицы констант»:

```
FROM (VALUES ('anne', 'smith'), ('bob', 'jones'), ('joe', 'blow'))
     AS names(first, last)
```

Такому подзапросу тоже требуется псевдоним.

Предложение WHERE

После обработки предложения FROM каждая строка полученной виртуальной таблицы проходит проверку по ***условию ограничения*** через предложение WHERE:

```
WHERE search_condition
```

где **search_condition** — любое выражение, выдающее результат типа **boolean** (предикат).

Если результат условия равен **true**, эта строка остаётся в выходной таблице, а если результат равен false или NULL, отбрасывается.

В условии ограничения, как правило, используется минимум один столбец из таблицы, полученной на выходе FROM. Примеры:

```
SELECT ... FROM foo WHERE c1 > 5

SELECT ... FROM foo WHERE c1 IN (1, 2, 3)

SELECT ... FROM foo WHERE c1 IN (SELECT c1 FROM t2)

SELECT ... FROM foo WHERE c1 IN (SELECT c3 FROM t2 WHERE c2 = foo.c1 + 10)

SELECT ... FROM foo WHERE c1 BETWEEN
  (SELECT c3 FROM t2 WHERE c2 = foo.c1 + 10) AND 100

SELECT ... FROM foo WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > foo.c1)
```

foo — название таблицы, порождённой в предложении FROM. Строки, которые не соответствуют условию WHERE, исключаются из foo.

Сортировка строк (ORDER BY)

После того как запрос выдал таблицу результатов (после обработки списка выборки), ее можно отсортировать. Если сортировка не задана, строки возвращаются в неопределённом порядке.

Фактический порядок строк в этом случае будет зависеть от плана соединения и сканирования, а также от порядка данных на диске.

Определённый порядок выводимых строк гарантируется только в том случае, если явно задан этап сортировки.

Порядок сортировки определяет предложение **ORDER BY**:

```
SELECT список_выборки FROM табличное_выражение
    ORDER BY выражение_сортировки1 [ASC | DESC] [NULLS { FIRST | LAST }]
             [, выражение_сортировки2 [ASC | DESC] [NULLS { FIRST | LAST }] ...]
```

Выражениями сортировки могут быть любые выражения, допустимые в списке выборки запроса:

```
SELECT a, b FROM table1 ORDER BY a + b, c;
```

Когда указывается несколько выражений, последующие значения позволяют отсортировать строки, в которых совпали все предыдущие значения .

Каждое выражение можно дополнить ключевыми словами ASC или DESC, которые выбирают сортировку соответственно по возрастанию или убыванию. По умолчанию ASC.

При сортировке по возрастанию сначала идут меньшие значения, где понятие «меньше» определяется оператором <. Подобным образом, сортировка по возрастанию определяется оператором >.

Для определения места значений NULL можно использовать указания NULLS FIRST и NULLS LAST, которые помещают значения NULL соответственно до или после значений не NULL.

По умолчанию значения NULL считаются больше любых других, то есть подразумевается NULLS FIRST для порядка DESC и NULLS LAST в противном случае.