**Тема 4. Язык запросов SQL. Эффективное выполнения запросов . Подзапросы. Оператор JOIN..**

**Табличные выражения**

Табличное выражение вычисляет таблицу. Это выражение содержит предложение FROM, за которым могут следовать предложения WHERE, GROUP BY и HAVING. Тривиальные табличные выражения просто ссылаются на физическую таблицу, её называют также 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, вычисляющему выходные строки запроса.

**Предложение FROM**

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

FROM табличная\_ссылка [, табличная\_ссылка [, ...]]

Здесь табличной ссылкой может быть имя таблицы (возможно, с именем схемы), производная таблица, например подзапрос, соединение таблиц или сложная комбинация этих вариантов. Если в предложении FROM перечисляются несколько ссылок, для них применяется перекрёстное соединение (то есть декартово произведение их строк; см. ниже). Список FROM преобразуется в промежуточную виртуальную таблицу, которая может пройти через преобразования WHERE, GROUP BY и HAVING, и в итоге определит результат табличного выражения.

**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 запроса. Такое имя называется псевдонимом таблицы.

Определить псевдоним таблицы можно, написав

FROM табличная\_ссылка AS псевдоним

Ключевое слово AS является необязательным. Вместо псевдоним здесь может быть любой идентификатор. Псевдонимы часто применяются для назначения коротких идентификаторов длинным именам таблиц с целью улучшения читаемости запросов. Например:

SELECT \* FROM "очень\_длинное\_имя\_таблицы" s JOIN "другое\_длинное\_имя" a ON s.id = a.num;

Псевдоним становится новым именем таблицы в рамках текущего запроса, т. е. 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;

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

В другой форме назначения псевдонима временные имена даются не только таблицам, но и её столбцам:

FROM табличная\_ссылка [AS] псевдоним ( столбец1 [, столбец2 [, ...]] )

Если псевдонимов столбцов оказывается меньше, чем фактически столбцов в таблице, остальные столбцы сохраняют свои исходные имена. Эта запись особенно полезна для замкнутых соединений или подзапросов.

**Подзапросы**

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

FROM (SELECT \* FROM table1) AS псевдоним

Этот пример равносилен записи FROM table1 AS псевдоним. Более интересные ситуации, которые нельзя свести к простому соединению, возникают, когда в подзапросе используются агрегирующие функции или группировка.

Примеры подзапросов:

SELECT \* FROM t1 WHERE num IN (select num from t2 WHERE name IN ('b', 'c'))

SELECT num FROM t1 WHERE EXISTS (select \* from t2 WHERE t1.name = t2.name)

Предикат EXISTS принимает значение TRUE, если подзапрос содержит любое количество строк, иначе его значение равно FALSE. Для NOT EXISTS все наоборот. Этот предикат никогда не принимает значение UNKNOWN.

Обычно предикат EXISTS используется в зависимых (коррелирующих) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе. Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому предикат EXISTS может иметь разные значения для разных строк основного запроса.

**Предложение WHERE**

Предложение WHERE записывается так:

WHERE условие\_ограничения

где условие\_ограничения — любое выражение значения, выдающее результат типа boolean.

После обработки предложения FROM каждая строка полученной виртуальной таблицы проходит проверку по условию ограничения. Если результат условия равен true, эта строка остаётся в выходной таблице, а иначе (если результат равен false или NULL) отбрасывается. В условии ограничения, как правило, задействуется минимум один столбец из таблицы, полученной на выходе FROM. Хотя строго говоря, это не требуется, но в противном случае предложение WHERE будет бессмысленным.

Несколько примеров запросов с WHERE:

SELECT ... FROM fdt WHERE c1 > 5

SELECT ... FROM fdt WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > fdt.c1)

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

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

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

Оператор SQL IN позволяет определить, совпадает ли значение объекта со значением в списке. Оператор SQL IN имеет следующий синтаксис:

expression [ NOT ] IN ( expression, [...] )

Оператор SQL BETWEEN задает диапазон, в котором будет осуществляться проверка условия.

SELECT ... FROM fdt WHERE c1 BETWEEN

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

Оператор SQL BETWEEN имеет следующий синтаксис:

test\_expression [NOT] BETWEEN begin\_expression AND end\_expression

test\_expression — задает объект для проверки по диапазону;

start\_expression — начальное значение диапазона;

end\_expression — конечное значение диапазона;

fdt — название таблицы, порождённой в предложении FROM. Строки, которые не соответствуют условию WHERE, исключаются из fdt. Обратите внимание, как в качестве выражений значения используются скалярные подзапросы. Как и любые другие запросы, подзапросы могут содержать сложные табличные выражения. Заметьте также, что fdt используется в подзапросах. Дополнение имени c1 в виде fdt.c1 необходимо только, если в порождённой таблице в подзапросе также оказывается столбец c1. Полное имя придаёт ясность даже там, где без него можно обойтись. Этот пример показывает, как область именования столбцов внешнего запроса распространяется на все вложенные в него внутренние запросы.

Предикат LIKE сравнивает строку, указанную в первом выражении, для вычисления значения строки, называемого проверяемым значением, с образцом, который определен во втором выражении для вычисления значения строки.

Синтаксис:

LIKE::=

<Выражение для вычисления значения строки>

[NOT] LIKE <Выражение для вычисления значения строки>

[ESCAPE <символ>]

В образце разрешается использовать два трафаретных символа:

* символ подчеркивания (\_), который можно применять вместо любого единичного символа в проверяемом значении;
* символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.

Если проверяемое значение соответствует образцу с учетом трафаретных символов, то значение предиката равно TRUE. Пример написания образцов:

SELECT \* FROM t1 WHERE name LIKE 'a\_c%'

Оператор SQL DISTINCT используется для указания на то, что следует работать только с уникальными значениями столбца.

Оператор SQL DISTINCT нашел широкое применение в операторе SQL SELECT, для выборки уникальных значений. Также используется в агрегатных функциях. Оператор SQL DISTINCT имеет следующий синтаксис:

SELECT DISTINCT column\_name FROM table\_name

Примеры:

SELECT DISTINCT num, name FROM t1

SELECT COUNT(DISTINCT num) FROM t1

**Агрегатные функции**

Агрегатные функции SQL действуют в отношении значений столбца с целью получения единого результирующего значения. Наиболее часто применяются агрегатные функции SQL SUM, MIN, MAX, AVG и COUNT. Следует различать два случая применения агрегатных функций. Первый: агрегатные функции используются сами по себе и возвращают одно результирующее значение. Второй: агрегатные функции используются с оператором SQL GROUP BY, то есть с группировкой по полям (столбцам) для получения результирующих значений в каждой группе.

Имеется список этих функций:

COUNT - производит номера строк или не-NULL значения полей которые выбрал запрос.

SUM - производит арифметическую сумму всех выбранных значений данного пол.

AVG - производит усреднение всех выбранных значений данного пол.

MAX - производит наибольшее из всех выбранных значений данного пол.

MIN - производит наименьшее из всех выбранных значений данного пол.

Примеры использования агрегатных функций без группировки:

SELECT SUM(num) FROM t2

SELECT MIN(num) FROM t2

SELECT MAX(num) FROM t2

SELECT AVG(num) FROM t2

Рассмотрим сначала случаи использования агрегатных функций без группировки.

**Предложения GROUP BY и HAVING**

Строки порождённой входной таблицы, прошедшие фильтр WHERE, можно сгруппировать с помощью предложения GROUP BY, а затем оставить в результате только нужные группы строк, используя предложение HAVING.

SELECT список\_выборки

FROM ...

[WHERE ...]

GROUP BY группирующий\_столбец [, группирующий\_столбец]...

Предложение GROUP BY группирует строки таблицы, объединяя их в одну группу при совпадении значений во всех перечисленных столбцах. Порядок, в котором указаны столбцы, не имеет значения. В результате наборы строк с одинаковыми значениями преобразуются в отдельные строки, представляющие все строки группы. Это может быть полезно для устранения избыточности выходных данных и/или для вычисления агрегатных функций, применённых к этим группам. Например:

SELECT \* FROM test1;

x | y

---+---

a | 3

c | 2

b | 5

a | 1

(4 rows)

=> SELECT x FROM test1 GROUP BY x;

x

---

a

b

c

(3 rows)

Во втором запросе мы не могли написать SELECT \* FROM test1 GROUP BY x, так как для столбца y нет единого значения, связанного с каждой группой. Однако столбцы, по которым выполняется группировка, можно использовать в списке выборки, так как они имеют единственное значение в каждой группе.

Вообще говоря, в группированной таблице столбцы, не включённые в список GROUP BY, можно использовать только в агрегатных выражениях. Пример такого агрегатного выражения:

=> SELECT x, sum(y) FROM test1 GROUP BY x;

x | sum

---+-----

a | 4

b | 5

c | 2

(3 rows)

Если таблица была сгруппирована с помощью GROUP BY, но интерес представляют только некоторые группы, отфильтровать их можно с помощью предложения HAVING, действующего подобно WHERE. Записывается это так:

SELECT список\_выборки FROM ... [WHERE ...] GROUP BY ... HAVING логическое\_выражение

В предложении HAVING могут использоваться и группирующие выражения, и выражения, не участвующие в группировке (в этом случае это должны быть агрегирующие функции).

Пример:

=> SELECT x, sum(y) FROM test1 GROUP BY x HAVING sum(y) > 3;

x | sum

---+-----

a | 4

b | 5

(2 rows)

=> SELECT x, sum(y) FROM test1 GROUP BY x HAVING x < 'c';

x | sum

---+-----

a | 4

b | 5

(2 rows)

**Соединённые таблицы**

Соединённая 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 тип\_соединения T2 [ условие\_соединения ]

Соединения любых типов могут вкладываются друг в друга или объединяться: и 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**

1. Перекрёстное соединение

T1 CROSS JOIN T2

Соединённую таблицу образуют все возможные сочетания строк из T1 и T2 (т. е. их декартово произведение), а набор её столбцов объединяет в себе столбцы T1 со следующими за ними столбцами T2. Если таблицы содержат N и M строк, соединённая таблица будет содержать N \* M строк.

FROM T1 CROSS JOIN T2 эквивалентно FROM T1 INNER JOIN T2 ON TRUE (см. ниже). Эта запись также эквивалентна FROM T1, T2.

1. Соединения с сопоставлениями строк

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

Слова INNER и OUTER необязательны во всех формах. По умолчанию подразумевается INNER (внутреннее соединение), а при указании LEFT, RIGHT и FULL — внешнее соединение.

Условие соединения указывается в предложении ON или USING, либо неявно задаётся ключевым словом NATURAL. Это условие определяет, какие строки двух исходных таблиц считаются «соответствующими» друг другу (это подробно рассматривается ниже).

Возможные типы соединений с сопоставлениями строк:

INNER JOIN

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

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

Предложение ON определяет наиболее общую форму условия соединения: в нём указываются выражения логического типа, подобные тем, что используются в предложении WHERE. Пара строк из T1 и T2 соответствуют друг другу, если выражение ON возвращает для них true.

USING — это сокращённая запись условия, полезная в ситуации, когда 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.

Более того, при выводе JOIN USING исключаются избыточные столбцы: оба сопоставленных столбца выводить не нужно, так как они содержат одинаковые значения. Тогда как JOIN ON выдаёт все столбцы из T1, а за ними все столбцы из T2, JOIN USING выводит один столбец для каждой пары (в указанном порядке), за ними все оставшиеся столбцы из T1 и, наконец, все оставшиеся столбцы T2.

Наконец, 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. Если столбцов с одинаковыми именами не находится, NATURAL JOIN действует как JOIN ... ON TRUE и выдаёт декартово произведение строк.

Для наглядности предположим, что у нас есть таблицы t1:

num | name

-----+------

1 | a

2 | b

3 | c

и t2:

num | value

-----+-------

1 | xxx

3 | yyy

5 | zzz

С ними для разных типов соединений мы получим следующие результаты:

=> SELECT \* FROM t1 CROSS JOIN t2;

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

1 | a | 3 | yyy

1 | a | 5 | zzz

2 | b | 1 | xxx

2 | b | 3 | yyy

2 | b | 5 | zzz

3 | c | 1 | xxx

3 | c | 3 | yyy

3 | c | 5 | zzz

(9 rows)

=> SELECT \* FROM t1 INNER JOIN t2 ON t1.num = t2.num;

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

3 | c | 3 | yyy

(2 rows)

=> SELECT \* FROM t1 INNER JOIN t2 USING (num);

num | name | value

-----+------+-------

1 | a | xxx

3 | c | yyy

(2 rows)

=> SELECT \* FROM t1 NATURAL INNER JOIN t2;

num | name | value

-----+------+-------

1 | a | xxx

3 | c | yyy

(2 rows)

=> SELECT \* FROM t1 LEFT JOIN t2 ON t1.num = t2.num;

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

2 | b | |

3 | c | 3 | yyy

(3 rows)

=> SELECT \* FROM t1 LEFT JOIN t2 USING (num);

num | name | value

-----+------+-------

1 | a | xxx

2 | b |

3 | c | yyy

(3 rows)

=> SELECT \* FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

3 | c | 3 | yyy

| | 5 | zzz

(3 rows)

=> SELECT \* FROM t1 FULL JOIN t2 ON t1.num = t2.num;

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

2 | b | |

3 | c | 3 | yyy

| | 5 | zzz

(4 rows)

Условие соединения в предложении ON может также содержать выражения, не связанные непосредственно с соединением. Это может быть полезно в некоторых запросах, но не следует использовать это необдуманно. Рассмотрите следующий запрос:

=> SELECT \* FROM t1 LEFT JOIN t2 ON t1.num = t2.num AND t2.value = 'xxx';

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

2 | b | |

3 | c | |

(3 rows)

Заметьте, что если поместить ограничение в предложение WHERE, вы получите другой результат:

=> SELECT \* FROM t1 LEFT JOIN t2 ON t1.num = t2.num WHERE t2.value = 'xxx';

num | name | num | value

-----+------+-----+-------

1 | a | 1 | xxx

(1 row)

Это связано с тем, что ограничение, помещённое в предложение ON, обрабатывается до операции соединения, тогда как ограничение в WHERE — после. Это не имеет значения при внутренних соединениях, но важно при внешних.