Базы данных

Лекция 05 – Основы SQL

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

2022.09.29

Оглавление

Язык SQL....3
Синтаксис....5
Идентификаторы и ключевые слова....6
Выделенные идентификаторы или идентификаторы в кавычках....7
Константы....8
Числовые константы....11
Операторы....13
Специальные знаки....15
Комментарии....16
Приоритеты операторов....17

Язык SQL

В языке SQL вместо терминов «отношение» и «переменная отношения» используются термины «таблица», а вместо терминов «кортеж» и «атрибут» — «строка» и «столбец».

Эти термины используются в стандарте языка SQL и в поддерживающих его продуктах.

SQL — язык очень большого объема. Его спецификация[1] содержит несколько тысяч страниц.

SQL не является в полной мере реляционным языком.

В языке SQL имеются операции как определения данных, так и манипулирования ими.

SQL (Structured Query Language — «язык структурированных запросов») — ***декларативный язык программирования***, применяемый для создания, модификации и управления данными в реляционной базе данных под управлением СУБД (DBMS).

Предназначен для описания, изменения и извлечения данных, хранимых в реляционных базах данных. «Чистый» SQL без современных расширений считается языком программирования не полным по Тьюрингу, но в стандарте языка есть спецификация SQL/PSM, которая предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

- создание в БД новой таблицы;
- добавление в таблицу новых записей;
- изменение записей;
- удаление записей;
- выборка записей из одной или нескольких таблиц в соответствии с заданным условием;
- изменение структур таблиц.

---

1) International Organization for Standardization (ISO): Information Technology — Database Languages — SQL, Document ISO/IEC 9075:2023

Современный SQL существенно усложнился — появились возможности описания и управления такими хранимыми объектами, как индексы, представления, триггеры и хранимые процедуры.

В результате стал приобретать черты, свойственные языкам программирования.

В данный момент SQL представляет собой совокупность операторов, инструкций и вычисляемых функций. Операторы SQL делятся на:

операторы определения данных (Data Definition Language, DDL):

CREATE создаёт объект базы данных (саму базу, таблицу, представление, пользователя и т.п.);
ALTER изменяет объект;
DROP удаляет объект.

операторы манипуляции данными (Data Manipulation Language, DML):

SELECT выбирает данные, удовлетворяющие заданным условиям;
INSERT добавляет новые данные;
UPDATE изменяет существующие данные;
DELETE удаляет данные.

операторы определения доступа к данным (Data Control Language, DCL):

GRANT предоставляет пользователю (группе) разрешения на определённые операции с объектом;
REVOKE отзывает ранее выданные разрешения;
DENY задаёт запрет, имеющий приоритет над разрешением.

операторы управления транзакциями (Transaction Control Language, TCL):

COMMIT регистрирует транзакцию;
ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции;
SAVEPOINT делит транзакцию на более мелкие участки.

Синтаксис

SQL-программа состоит из последовательности команд.

Команда — последовательность компонентов, оканчивающуюся точкой с запятой '**;**'.

Конец входного потока также считается концом команды.

Какие именно компоненты допустимы для конкретной команды, зависит от её синтаксиса.

Компонентом команды может быть:

- ключевое слово;
- идентификатор;
- идентификатор в кавычках;
- строка (или константа);
- специальный символ.

Компоненты обычно разделяются пробельными символами (пробел, табуляция, перевод строки), но это не требуется, если нет неоднозначности, например, когда спецсимвол оказывается рядом с компонентом другого типа.

Пример

```
SELECT * FROM MY_TABLE;
UPDATE MY_TABLE SET A = 5;
INSERT INTO MY_TABLE VALUES (3, 'hi there');
```

SQL-программы могут содержать ***комментарии*** – они не являются компонентами команд и пропускаются при интерпретации.

Идентификаторы – идентифицируют имена таблиц, столбцов или других объектов БД, в зависимости от того, где они используются. Иногда их называют «именами».

Ключевые слова и идентификаторы имеют одинаковую лексическую структуру, то есть, не зная языка, нельзя определить, является ли некоторый компонент ключевым словом или идентификатором. Ключевых слов около тысячи.

Идентификаторы и ключевые слова

Идентификаторы и ключевые слова SQL должны начинаться с буквы (**a-z**). Допускаются также не латинские буквы и буквы с диакритическими знаками или подчёркивания (_). Последующими символами в идентификаторе или ключевом слове могут быть буквы, цифры (**0-9**), знаки доллара (**$**) или подчёркивания.

Стандарт SQL не включает использование знака доллара в идентификаторах, поэтому их использование влияет на переносимость приложений.

В стандарте SQL гарантированно не будет ключевых слов с цифрами и начинающихся или заканчивающихся подчёркиванием.

Система выделяет для идентификатора не более **NAMEDATALEN-1** байт, а более длинные имена усекаются.

По умолчанию **NAMEDATALEN** равно 64, так что максимальная длина идентификатора равна 63 байта.

Если этого недостаточно, этот предел можно увеличить, изменив константу **NAMEDATALEN** в файле **src/include/pg_config_manual.h**.

Ключевые слова и идентификаторы без кавычек воспринимаются системой без учёта регистра.

```
UPDATE MY_TABLE SET A = 5;
update my_table set a = 5;
```

Обычно используется неформальное соглашение записывать ключевые слова заглавными буквами, а имена строчными:

```
UPDATE my_table SET a = 5;
```

Выделенные идентификаторы или идентификаторы в кавычках

Обычный набор символов, заключенный в двойные кавычки (").

Такие идентификаторы всегда будут считаться идентификаторами, но не ключевыми словами.

Таким образом **"update"** можно использовать для обозначения столбца или таблицы, в то время как **upadte** без кавычек будет воспринят как ключевое слово.

```
UPDATE "my_table" SET "a" = 5;
```

Идентификаторы в кавычках могут содержать любые символы, за исключением символа **'\0'**.

Чтобы включить в такой идентификатор кавычки, их нужно продублировать.

Кавычки позволяют создавать таблицы и столбцы с именами, которые иначе были бы невозможны, например, с пробелами или амперсандами. Ограничение длины при этом сохраняется.

Идентификатор, заключённый в кавычки, становится зависимым от регистра.

Идентификаторы без кавычек в Postgres всегда переводятся в нижний регистр, что несовместимо со стандартом SQL, где говорится о том, что имена должны приводиться к верхнему регистру.

То есть, согласно стандарту **foo** должно быть эквивалентно **"FOO"**, но не **"foo"**. Поэтому при создании переносимых приложений рекомендуется либо всегда заключать определённое имя в кавычки, либо не заключать никогда.

Вариант идентификаторов в кавычках позволяет использовать символы Unicode по их кодам.

Такой идентификатор начинается с **U&**, а затем сразу без пробелов идёт двойная кавычка, например **U&"foo"**.

В кавычках можно записывать символы Unicode двумя способами — обратная косая черта, а за ней код символа из четырёх шестнадцатеричных цифр, либо обратная косая черта, знак плюс, а затем код из шести шестнадцатеричных цифр. Например, идентификатор "data" можно записать так:

```
U&"d\0061t\+000061"
```

Константы

В PostgreSQL есть три типа констант подразумеваемых типов — строки, битовые строки и числа. Константы можно также записывать, указывая типы явно.

Строковые константы

Строковая константа в SQL — это обычная последовательность символов, заключённая в апострофы (**'**), например:

```
'Это строка'
```

Чтобы включить апостроф в строку, нужно использовать два апострофа рядом:

```
'Жанна д''Арк'
```

Две строковые константы, разделённые пробельными символами и *минимум* одним переводом строки, объединяются в одну и обрабатываются, как если бы строка была записана в одной константе:

```
SELECT 'foo'
'bar';
```

эквивалентно:

```
SELECT 'foobar';
```

но будет синтаксической ошибкой:

```
SELECT 'foo'      'bar';
```

Строковые константы со спецпоследовательностями в стиле C

Postgres принимает «спецпоследовательности», что является расширением стандарта SQL.

Строка со спецпоследовательностями начинается с буквы **E** (заглавной или строчной), стоящей непосредственно перед апострофом:

```
E'foo'
```

Внутри таких строк символ '\' начинает C-подобные спецпоследовательности:

Спецпоследовательность	Интерпретация
\b	символ «забой»
\f	подача формы
\n	новая строка
\r	возврат каретки
\t	табуляция
\o, \oo, \ooo (o = 0–7)	восьмеричное значение байта
\xh, \xhh (h = 0–9, A–F)	шестнадцатеричное значение байта
\uxxxx, \Uxxxxxxxx (x = 0–9, A–F)	16- или 32-битный шестнадцатеричный код символа Unicode

Любой другой символ, идущий после обратной косой черты, воспринимается буквально.

Таким образом, чтобы включить в строку обратную косую черту, нужно написать две косых черты (\\). Так же можно включить в строку апостроф, написав \', в дополнение к ''.

Строковые константы, заключённые в доллары

Стандартный синтаксис для строковых констант может плохо читаться, когда строка содержит много апострофов или обратных косых черт, поскольку каждый такой символ приходится дублировать.

Чтобы и в таких случаях запросы оставались читаемыми, Postgres предлагает способ записи строковых констант — «заключение строк в доллары».

Строковая константа, заключённая в доллары, начинается со знака доллара '**$**', необязательного «*тега*» из нескольких символов и ещё одного знака доллара, затем содержит обычную последовательность символов, составляющую строку, и оканчивается знаком доллара, тем же тегом и замыкающим знаком доллара.

Например, строку «Жанна д'Арк» можно записать в долларах двумя способами:

```
$$Жанна д'Арк$$
$SomeTag$Жанна д'Арк$SomeTag$
```

Битовые строковые константы

Битовые строковые константы похожи на обычные с дополнительной буквой **B** (заглавной или строчной), добавленной непосредственно перед открывающим апострофом без пробелов:

```
B'1001'.
```

В битовых строковых константах допускаются лишь символы 0 и 1.

Битовые константы могут быть записаны в шестнадцатеричном виде, с начальной буквой **X** (заглавной или строчной):

```
X'1FF'
```

Числовые константы

<цифры>
<цифры>**.**[<цифры>][**e**[**+-**]<цифры>]
[<цифры>]**.**<цифры>[**e**[**+-**]<цифры>]
<цифры>**e**[**+-**]<цифры>

где <цифры> — это одна или несколько десятичных цифр (**0..9**).

До или после десятичной точки (при её наличии) должна быть минимум одна цифра.

Как минимум одна цифра должна следовать за обозначением экспоненты (**e**), если она присутствует. В числовой константе не может быть пробелов или других символов. Любой знак минус или плюс в начале строки не считается частью числа — это оператор, применённый к константе.

Примеры

42
3.5
4.
.001
5e2
1.925e-3

Числовая константа, не содержащая точки и экспоненты, изначально рассматривается как константа типа **integer**, если её значение умещается в 32-битный тип **integer**, затем как константа типа **bigint**, если её значение умещается в 64-битный **bigint**, в противном случае она принимает тип **numeric**.

Константы, содержащие десятичные точки и/или экспоненты, всегда считаются константами типа **numeric**.

Константы других типов

Константу обычного типа можно ввести одним из следующих способов:

```
type 'string'               -- REAL '123.45'
'string'::type              -- '123.45'::REAL
CAST ( 'string' AS type ) -- CAST ( '123.45' AS REAL )
```

Явное приведение типа можно опустить, если нужный тип константы определяется однозначно (например, когда она присваивается непосредственно столбцу таблицы), так как в этом случае приведение происходит автоматически.

Операторы

Имя оператора представляет собой последовательность не более чем **NAMEDATALEN-1** (по умолчанию 63) символов из следующего списка:

+ - * / < > = ~ ! @ # % ^ & | ` ?

Однако для имён операторов есть ещё несколько ограничений:

- Сочетания символов **--** и **/*** не могут присутствовать в имени оператора (начало комментария).
- Многосимвольное имя оператора не может заканчиваться знаком **+** или **-**, если только оно не содержит также один из этих символов:

~ ! @ # % ^ & | ` ?

@- — допустимое имя оператора

***-** — недопустимое имя оператора.

Благодаря этому ограничению, Postgres может разбирать корректные SQL-запросы без пробелов между компонентами.

```
CREATE OPERATOR имя (
    {FUNCTION|PROCEDURE} = имя_функции
    [, LEFTARG = тип_слева ] [, RIGHTARG = тип_справа ]
    [, COMMUTATOR = коммут_оператор ] [, NEGATOR = обратный_оператор ]
    [, RESTRICT = процедура_ограничения ] [, JOIN = процедура_соединения ]
    [, HASHES ] [, MERGES ]
)
```

Пользовательские операторы

Любой оператор представляет собой просто «синтаксический сахар» для вызова нижележащей функции, которая выполняет реальную работу.

Поэтому прежде чем можно создать оператор, необходимо создать нижележащую функцию.

Однако оператор не только синтаксический сахар — он несёт и дополнительную информацию, помогающую планировщику запросов оптимизировать запросы с этим оператором.

Postgres поддерживает префиксные и инфиксные операторы.

A + B — инфиксная запись (требует скобки для смены приоритетов) // A + B * C → A + (B * C)

+ A B — префиксная запись + A * B C

A B + — постфиксная запись B C * A + или A B C * + (стековые компьютеры)

Операторы могут быть перегружены — то есть одно имя оператора могут иметь различные операторы с разным количеством и типами операндов.

Когда выполняется запрос, система определяет, какой оператор вызвать, по количеству и типам предоставленных операндов.

Специальные знаки

Некоторые не алфавитно-цифровые символы имеют специальное значение, но при этом не являются операторами.

- Знак доллара '**$**', предваряющий число, используется для представления позиционного параметра в теле определения функции или подготовленного оператора. В других контекстах знак доллара может быть частью идентификатора или строковой константы, заключённой в доллары.

- Круглые скобки '**( )**' имеют обычное значение и применяются для группировки выражений и повышения приоритета операций. В некоторых случаях скобки — это необходимая часть синтаксиса определённых SQL-команд.

- Квадратные скобки '**[ ]**' применяются для выделения элементов массива.

- Запятые '**,**' используются в некоторых синтаксических конструкциях для разделения элементов списка.

- Точка с запятой '**;**' завершает команду SQL. Она не может находиться нигде внутри команды, за исключением строковых констант или идентификаторов в кавычках.

- Двоеточие '**:**' применяется для выборки «срезов» массивов (slice). В некоторых диалектах SQL, например, в Embedded SQL, двоеточие может быть префиксом в имени переменной.

- Звёздочка '*****' используется в некоторых контекстах как обозначение всех полей строки или составного значения. Она также имеет специальное значение, когда используется как аргумент некоторых агрегатных функций, а именно функций, которым не нужны явные параметры.

- Точка '**.**' используется в числовых константах, а также для отделения имён схемы, таблицы и столбца.

Комментарии

Комментарий — это последовательность символов, которая начинается с двух минусов и продолжается до конца строки:

```
-- Это стандартный комментарий SQL
```

Кроме этого, можно записывать блочные комментарии в стиле C:

```
/*
 * многострочный комментарий
 * с вложенностью: /* вложенный блок комментария */
 */
```

Здесь комментарий начинается с `/*` и продолжается до соответствующего вхождения `*/`.

Блочные комментарии можно вкладывать друг в друга — это разрешено по стандарту SQL и позволяет комментировать большие блоки кода, которые при этом уже могут содержать блоки комментариев.

Комментарий удаляется из входного потока в начале синтаксического анализа и фактически заменяется пробелом.

Приоритеты операторов

Большинство операторов имеют одинаковый приоритет и вычисляются слева направо.

Приоритет и очерёдность операторов жёстко фиксированы в синтаксическом анализаторе.

Если необходимо, чтобы выражение с несколькими операторами разбиралось не в том порядке, который диктуют приоритеты, следует использовать скобки.

Оператор/элемент	Очерёдность	Описание
`.`	слева-направо	разделитель имён таблицы и столбца
`::`	слева-направо	приведение типов в стиле PostgreSQL
`[ ]`	слева-направо	выбор элемента массива
`+ -`	справа-налево	унарный плюс, унарный минус
`^`	слева-направо	возведение в степень
`* / %`	слева-направо	умножение, деление, остаток от деления
`+ -`	слева-направо	сложение, вычитание
`(любой другой оператор)`	слева-направо	все другие встроенные и пользовательские операторы
`BETWEEN IN LIKE ILIKE SIMILAR`		проверка диапазона, проверка членства, сравнение строк
`< > = <= >= <>`		операторы сравнения
`IS ISNULL NOTNULL`		`IS TRUE, IS FALSE, IS NULL, IS DISTINCT FROM` и т. д.
`NOT`	справа-налево	логическое отрицание
`AND`	слева-направо	логическая конъюнкция
`OR`	слева-направо	логическая дизъюнкция

Правила приоритета операторов также применяются к операторам, определённым пользователем с теми же именами, что и вышеперечисленные встроенные операторы.