Функции в SQL, их виды и применение

— К доске пойдет… к доске пойдет… — Арина Леопольдовна пробежалась глазами по списку в школьном журнале. Антон вжался в парту и зажмурил глаза — ему совершенно не хотелось отвечать по теме *Функции*.

— Круглов! — учительница наконец назвала фамилию жертвы.

Антон вздохнул, поднялся, пошел к доске, и тут… прозвенел будильник. «Ну и приснится же, — подумал Антон. — Все-таки хорошо быть взрослым: не надо отвечать у доски».

В офисе его уже дожидался аналитик Ваня, который сразу обозначил план на сегодня:

- разобраться с функциями в SQL;
- с помощью изученных функций почистить столбец телефонных номеров клиентов в таблице contacts, которую недавно загрузили в СУБД.

«А сон, похоже, был вещим», — мелькнуло в голове у Антона, и он открыл документацию по функциям…

Давайте вместе с Антоном разберем понятие функций в SQL и их основные виды.

Со школьной скамьи известно, что функция — это некоторое правило, по которому одно число ставится в соответствие другому. К примеру, для функции $y = x^2$ каждому xсоответствует некоторое число, равное x в квадрате.

Для программистов это звучит несколько иначе: на вход передается аргумент x, функция воздействует на него, и на выходе получается новое число — квадрат старого. В отличие от школьных учебников, в программировании используются не только функции, работающие с числами, но и многие другие.

Ниже приведены основные виды функций в SQL:

- математические и агрегатные функции — работают с числовыми значениями;

Примечание: может показаться, что математические и агрегатные функции — одно и то же. Но это не так. Математические функции используются для обработки чисел в рамках отдельной строки: к примеру, при использовании функции округления числа вы получите одно значение для каждой строки. Агрегатные же используются для объединения чисел по столбцу — например, суммирования стоимостей всех покупок.

- строчные — работают со строчными значениями;
- функции для обработки даты и времени — необходимы для обработки разных временных форматов;
- другие функции — помогают приводить столбец к определенному типу данных (например, переводить из числа в строку).

В этом разделе речь пойдет о строчных функциях: зачем они нужны и где используются.

Ранее в курсе вы работали с готовыми таблицами, куда предварительно были загружены данные. При этом не возникал вопрос, откуда эти данные взялись. В базу данные загружаются из различных источников: это могут быть логирующие системы, в которых происходит отслеживание действий пользователей и их автоматическая запись в БД (к примеру, отслеживание статуса заказа в магазине), или фронтальные системы, куда пользователь вручную вносит информацию. В качестве примера можно привести сайт какой-либо компании, где пользователь оставляет свои данные при регистрации.

Из системы-источника данные в БД могут прийти с ошибками. К ним относятся как простые опечатки, так и ошибки технического характера. К примеру, из-за неправильной обработки данных в системе-источнике все значения могут быть загружены в БД без разделяющих символов. Пользоваться такими данными порой трудно, а иногда невозможно.

Однако данные — это новая нефть, поэтому нельзя просто удалять «испорченную» информацию. Надо научиться ею пользоваться. Самое простое решение в этом случае — очистить данные от ошибок. Для этого как раз и нужны строчные функции.

Любая строчная функция имеет следующий синтаксис:

```
FUNCTION_NAME(arg1, arg2, …).
```

Как вы уже догадались, FUNCTION_NAME — это наименование конкретной функции, а arg1, arg2, … — список аргументов, которые принимает функция на вход. Каждая функция может иметь разное количество аргументов, причем некоторые из них могут быть необязательными.

Посмотрите, как работает простейшая строчная функция **LENGTH(string)**. На вход она принимает единственный аргумент — некоторую строчную переменную; этот аргумент является обязательным. Задача функции **LENGTH** — посчитать количество символов, из которых состоит переданная строка. То есть после выполнения функции **LENGTH(string)** вместо нее будет подставлено число, равное количеству символов в строке **string**. Далее с этим числом можно делать все, что угодно, например вывести на экран с помощью SELECT:

(Язык — SQL)
```
SELECT LENGTH('Арина Леопольдовна');
```
Результат: 18

Примечание: в качестве аргумента можно передавать как некоторое константное значение, так и наименование атрибута таблицы. Если передать константу, функция вернет одно значение — длину этой константы. Если передать наименование поля — столбец чисел, в котором будет указана длина значения в каждой ячейке таблицы соответствующего атрибута.

Посмотрите на таблицу contacts, которую предстоит обработать Антону. В ней:

- CLIENT_NM — фамилия и имя клиента;
- CLENT_MOBILE_NUM — номер мобильного телефона клиента.

CLIENT_NM	CLENT_MOBILE_NUM
Иванова Екатерина	89294564564 /ekaterina_ivanova@ya.ru///
Сидорян Валентин	89169872346///
Орехова Ольга	89264278945 /orechova_olga@rambler.ru///
Гуров Сергей	89196832985

С помощью функции **LENGTH()** Антон может вывести длину каждой ячейки в этой таблице. Он заметил, что поле CLIENT_MOBILE_NUM сформировано с ошибками. Если в нем хранятся номера телефонов, длина каждой строки должна быть равна 11. Похоже, иногда сюда дополнительно загружаются имейлы клиентов, символы слеш и пробелы (но это неточно). Чтобы проверить гипотезу с пробелами, Антон решил вывести длину каждой ячейки в столбце CLIENT_MOBILE_NUM и написал следующий запрос:

```
(Язык — SQL)
SELECT
    client_nm,
    client_mobile_num,
    LENGTH(client_mobile_num) AS client_mobile_len
FROM contacts;
```

При этом на экран выводятся следующие значения:

CLIENT_NM	CLENT_MOBILE_NUM	CLIENT_MOBILE_LEN
Иванова Екатерина	89294564564 /ekaterina_ivanova@ya.ru///	41
Сидорян Валентин	89169872346///	16
Орехова Ольга	89264278945 /orechova_olga@rambler.ru//	41
Гуров Сергей	89196832985	13

Даже в последней строке, где, на первый взгляд, все в порядке, Антон заметил проблему: номер состоит из 11 символов, а длина равна 13. Значит, где-то стоят пробелы. «Одной функцией **LENGTH()** тут не обойтись», — подумал Антон и продолжил изучать документацию.

Строчные функции в SQL

До этого мы разобрали использование строчных функций на примере функции **LENGTH()**, а теперь рассмотрим остальные функции, которые используют в SQL для работы со строками.

- **REPLACE(string, find_str, change_str)** — заменяет в строке все вхождения искомых символов/подстрок на новое значение:
    - string — строка или поле, в котором необходимо произвести поиск и замену, обязательный аргумент;
    - find_str — подстрока, которую необходимо найти, обязательный аргумент;
    - change_str — подстрока, на которую необходимо заменить найденные вхождения, обязательный аргумент;

  (Язык — SQL)
  SELECT REPLACE('Арина Леопольдовна', 'на', 'ZZ');
  Результат: АриZZ ЛеопольдовZZ
- **string1||string 2** — объединяет несколько строк в одну:
    - string1, string2 — строки или поля, которые необходимо объединить, обязательные аргументы;

  (Язык — SQL)
  SELECT 'Арина'||'Леопольдовна';
  Результат: 'АринаЛеопольдовна'

  Знак конкатенации может быть использован несколько раз подряд:
  (Язык — SQL)
  SELECT 'Учитель'||'математики'||' '||'Арина'||'Леопольдовна'||'Кулакова';
  Результат: 'Учительматематики АринаЛеопольдовнаКулакова'

- **SUBSTR(string, start_position, symbol_len)** — «вырезает» из исходной строки подстроку с началом в start_position и длиной symbol_len:
    - string — строка или поле, в котором необходимо вырезать подстроку, обязательный аргумент;
    - start_position — номер позиции символа в строке, начиная с которого будет сформирована подстрока (нумерация символов начинается с 1), обязательный аргумент;
    - symbol_len — количество символов, которое необходимо «вырезать» из исходной строки, обязательный аргумент;

  (Язык — SQL)
  SELECT SUBSTR('Арина Леопольдовна', 3, 5);
  Результат: 'ина Л'

- **LOWER(string)/UPPER(string)** — приводит исходную строку к верхнему или нижнему регистру:
    - string — строка или поле, которое будет переведено в нижний/верхний регистр, обязательный аргумент;

  (Язык — SQL)
  SELECT
  UPPER('Арина Леопольдовна') AS big_letters,
  LOWER('Арина Леопольдовна') AS small_letters;

Результат:

BIG_LETTERS	SMALL_LETTERS
АРИНА ЛЕОПОЛЬДОВНА	арина леопольдовна

- **TRIM(string [, trim_string])** — удаляет все заданные символы с обеих сторон от строки:
    - string — строка или поле, в котором будут удалены символы/подстроки, обязательный аргумент;
    - trim_string — символ или набор символов, которые необходимо удалить, необязательный аргумент. В случае если аргумент не указан, по умолчанию будут удалены пробелы;

Можно использовать две разновидности этой функции: **LTRIM(string [, trim_string])** и **RTRIM(string [, trim_string])**, которые удаляют символы слева или справа от строки соответственно.

(Язык — SQL)

```
SELECT
    TRIM('№№Арина№№Леопольдовна№№', '№') AS trim_string,
    RTRIM('№№Арина№№Леопольдовна№№', '№') AS rtrim_string,
    LTRIM('№№Арина№№Леопольдовна№№', '№') AS ltrim_string;
```

TRIM_STRING	RTRIM_STRING	LTRIM_STRING
Арина№№Леопольдовна	№№Арина№№Леопольдовна	Арина№№Леопольдовна№№

В случае если **trim_string** состоит из нескольких символов, будут удалены только символы в основной строке.

(Язык — SQL)

```
SELECT
    TRIM('№№Арина№№Леопольдовна№№', '№А');
```

Результат: 'рина№№Леопольдовна'

Примечание: все функции, где необходимо найти подстроку, регистрозависимые. Это значит, что, к примеру, *У* и *у* — совершенно разные символы.

- **INSTR(string, find_substring)** — возвращает номер позиции первого вхождения подстроки в строку:
    - string — строка, в которой будет производиться поиск подстроки, обязательный аргумент;
    - find_substring — подстрока, которую необходимо найти, обязательный аргумент.

(Язык — SQL)

```
SELECT
    INSTR('Арина Леопольдовна', 'а');
```

Результат: 5

Примечание: синтаксис перечисленных функций приведен для СУБД SQLite, для других СУБД он может отличаться. К примеру, в СУБД Oracle вместо || может быть использована функция **CONCAT(string1, string2)**, а функция **INSTR()** имеет ряд дополнительных параметров. Поэтому прежде чем использовать функцию в новой для вас СУБД, стоит изучить соответствующую документацию.

Все описанные функции можно использовать и в операторе вывода SELECT, и в операторе фильтрации WHERE, и в операторе группировки, который будет разобран далее в курсе.

А что с Антоном? Он тем временем приводит в порядок таблицу с телефонами клиентов. То есть в новой таблице в поле CLIENT_MOBILE_NUM должны лежать только номера мобильных телефонов.

«Функции-то я знаю, — подумал Антон, — но, похоже, самое сложное — это понять, какие из них нужны и в каком порядке их применять». Для того чтобы в этом разобраться, Антон выписал на бумажку проблемы, обнаруженные в столбце с телефонами, и рядом написал возможные способы решения, а также их преимущества и недостатки.

Проблема	Решение
Лишние пробелы	Удалить их с помощью функции **TRIM()**: + удаляются все невидимые пробелы по краям - не нашел Удалить их с помощью функции **REPLACE()** + удаляются все пробелы - удалятся вообще все пробелы в строке, это приведет к потере разделителя между телефоном и email Итог: Буду лучше использовать **TRIM()**
Ненужные символы слеша	Удалить их с помощью функции **TRIM():** + удаляются все символы слеша по краям - удаляются только по краям Удалить их с помощью функции **REPLACE()** + удаляются все символы слеша - не нашел Итог: Да, тут лучше использовать **REPLACE()**
Ненужные почтовые адреса	Из полученной строки «вырезать» телефон с помощью функции **SUBSTR().** Альтернативы нет. Но какие аргументы нужно поставить? Поскольку каждый номер состоит из 11 символов, эта задача решается просто — нужно из строки выбрать одиннадцать

	символов, начиная с первого. То есть аргументы будут 1 и 11.

Поскольку Антон — еще новичок, он решил применять новый функционал поэтапно и только для одной записи из таблицы и выбрал следующую:

' 89294564564 /ekaterina_ivanova@ya.ru///'

Используя свои наработки, наш стажер реализовал вот такой алгоритм.
1) Удаляем пробелы с двух сторон от сроки.

(Язык — SQL)

```
SELECT TRIM('  89294564564 /ekaterina_ivanova@ya.ru///');
```

Результат: '89294564564 /ekaterina_ivanova@ya.ru///'

2) Заменяем знаки слеша на пустоту.

(Язык — SQL)

```
SELECT REPLACE(TRIM('  89294564564 /ekaterina_ivanova@ya.ru///), '/', '');
```

Результат: '89294564564 ekaterina_ivanova@ya.ru'

Примечание: для того чтобы использовать результаты предыдущего шага, Антон в новую функцию в качестве одного из аргументов «вкладывает» старую, то есть формирует *Вложенные функции*. В данном случае, вместо первого аргумента функции REPLACE() Антон вставил результат работы функции TRIM — очищенную от пробелов по краям исходную строку.

Стоит отметить, что чем больше вложенных функций, тем сложнее читать код, поэтому не стоит ими увлекаться.

3) Теперь необходимо отделить телефон от почты, для этого Антон решил использовать функцию **SUBSTR()**. Он знает, что номер телефона всегда состоит из 11 символов, поэтому, чтобы оставить только номер телефона, Антон из строки «вырезал» подстроку длиною 11 с началом в первом символе.

(Язык — SQL)

```
SELECT SUBSTR(REPLACE(TRIM('  89294564564 /ekaterina_ivanova@ya.ru///'), '/', ''), 1, 11);
```

Результат: '89294564564'

Примечание: если бы длина телефонного номера менялась от строки к строке, то прежде чем использовать функцию **SUBSTR(),** пришлось бы искать индекс вхождения пробела, который разделяет номер телефона и адрес электронной почты, с помощью функции **INSTR()**, после чего «вырезать» символы с первого индекса до найденного.

А весь запрос теперь выглядит вот так:

(Язык — SQL)

```
SELECT
        client_nm,
        client_mobile_num,
```

SUBSTR(REPLACE(TRIM(CLIENT_MOBILE_NUM), '/', ''), 1, 11) as new_client_mobile_num
FROM contacts;

И получается следующий результат:

CLIENT_NM	CLENT_MOBILE_NUM	CLIENT_MOBILE_LEN
Иванова Екатерина	89294564564 /ekaterina_ivanova@ya.ru///	89294564564
Сидорян Валентин	89169872346///	89169872346
Орехова Ольга	89264278945 /orechova_olga@rambler.ru//	89264278945
Гуров Сергей	89196832985	89196832985

— Ну вот и готово, Арина Леопольдовна, ставьте пятерку, — произнес Антон, собираясь домой.

Итоги:

- В этом блоке мы узнали что такое функция в SQL, какие виды функций бывают и как работают строчные функции.