Ben Forta

FOURTH EDITION

FULL COLOR

Sams Teach Yourself

SQL in 10 Minutes

SAMS

Ben Forta

Sams Teach Yourself SQL in 10 Minutes

Fourth Edition

SAMS 800 East 96th Street, Indianapolis, Indiana 46240

Using Functions

Most SQL implementations support the following types of functions:

- Text functions are used to manipulate strings of text (for example, trimming or padding values and converting values to upper and lowercase).
- Numeric functions are used to perform mathematical operations on numeric data (for example, returning absolute numbers and performing algebraic calculations).
- Date and time functions are used to manipulate date and time values and to extract specific components from these values (for example, returning differences between dates, and checking date validity).
- System functions return information specific to the DBMS being used (for example, returning user login information).

In the last lesson, you saw a function used as part of a column list in a `SELECT` statement, but that's not all functions can do. You can use functions in other parts of the `SELECT` statement (for instance in the `WHERE` clause), as well as in other SQL statements (more on that in later lessons).

Text Manipulation Functions

You've already seen an example of text-manipulation functions in the last lesson—the `RTRIM()` function was used to trim white space from the end of a column value. Here is another example, this time using the `UPPER()` function:

Input ▼

```
SELECT vend_name, UPPER(vend_name) AS vend_name_upcase
FROM Vendors
ORDER BY vend_name;
```

Output ▼

```
vend_name                      vend_name_upcase
---------------------------    ----------------------------
Bear Emporium                    BEAR EMPORIUM
Bears R Us                       BEARS R US
Doll House Inc.                  DOLL HOUSE INC.
Fun and Games                    FUN AND GAMES
Furball Inc.                     FURBALL INC.
Jouets et ours                   JOUETS ET OURS
```

As you can see, `UPPER()` converts text to upper case and so in this example each vendor is listed twice, first exactly as stored in the `Vendors` table, and then converted to upper case as column `vend_name_upcase`.

Table 8.2 lists some commonly used text-manipulation functions.

TABLE 8.2 Commonly Used Text-Manipulation Functions

Function	**Description**
`LEFT()` (or use substring function)	Returns characters from left of string
`LENGTH()` (also `DATALENGTH()` or `LEN()`)	Returns the length of a string
`LOWER()`(`LCASE()` if using Access)	Converts string to lowercase
`LTRIM()`	Trims white space from left of string
`RIGHT()` (or use substring function)	Returns characters from right of string
`RTRIM()`	Trims white space from right of string
`SOUNDEX()`	Returns a strings `SOUNDEX` value
`UPPER()` (`UCASE()` if using Access)	Converts string to uppercase

One item in Table 8.2 requires further explanation. `SOUNDEX` is an algorithm that converts any string of text into an alphanumeric pattern describing the phonetic representation of that text. `SOUNDEX` takes into account similar sounding characters and syllables, enabling strings to be compared by how they sound rather than how they have been typed. Although SOUNDEX is not a SQL concept, most DBMSs do offer SOUNDEX support.

NOTE: **SOUNDEX Support**

SOUNDEX() is not supported by Microsoft Access or PostgreSQL, and so the following example will not work on those DBMSs.

In addition, it is only available in SQLite if the SQLITE_SOUNDEX compile-time option is used when SQLite is built, and as this is not the default compile option, most SQLite implementations won't support SOUNDEX().

Here's an example using the SOUNDEX() function. Customer Kids Place is in the Customers table and has a contact named Michelle Green. But what if that were a typo, and the contact actually was supposed to have been Michael Green? Obviously, searching by the correct contact name would return no data, as shown here:

Input ▼

```
SELECT cust_name, cust_contact
FROM Customers
WHERE cust_contact = 'Michael Green';
```

Output ▼

```
cust_name                     cust_contact
--------------------------    ----------------------------
```

Now try the same search using the SOUNDEX() function to match all contact names that sound similar to Michael Green:

Input ▼

```
SELECT cust_name, cust_contact
FROM Customers
WHERE SOUNDEX(cust_contact) = SOUNDEX('Michael Green');
```

Output ▼

```
cust_name                      cust_contact
--------------------------     ----------------------------
Kids Place                     Michelle Green
```

Analysis ▼

In this example, the `WHERE` clause uses the `SOUNDEX()` function to convert both the `cust_contact` column value and the search string to their `SOUNDEX` values. Because `Michael Green` and `Michelle Green` sound alike, their `SOUNDEX` values match, and so the `WHERE` clause correctly filtered the desired data.

Date and Time Manipulation Functions

Date and times are stored in tables using datatypes, and each DBMS uses its own special varieties. Date and time values are stored in special formats so that they may be sorted or filtered quickly and efficiently, as well as to save physical storage space.

The format used to store dates and times is usually of no use to your applications, and so date and time functions are almost always used to read, expand, and manipulate these values. Because of this, date and time manipulation functions are some of the most important functions in the SQL language. Unfortunately, they also tend to be the least consistent and least portable.

To demonstrate the use of date manipulation function, here is a simple example. The `Orders` table contains all orders along with an order date. To retrieve a list of all orders made in `2012` in SQL Server, do the following:

Input ▼

```
SELECT order_num
FROM Orders
WHERE DATEPART(yy, order_date) = 2012;
```

Output ▼

```
order_num
-----------
20005
20006
20007
20008
20009
```

In Access use this version:

Input ▼

```
SELECT order_num
FROM Orders
WHERE DATEPART('yyyy', order_date) = 2012;
```

Analysis ▼

This example (both the SQL Server and Sybase version, and the Access version) uses the `DATEPART()` function which, as its name suggests, returns a part of a date. `DATEPART()` takes two parameters, the part to return, and the date to return it from. In our example `DATEPART()` returns just the year from the `order_date` column. By comparing that to `2012`, the `WHERE` clause can filter just the orders for that year.

Here is the PostgreSQL version which uses a similar function named `DATE_PART()`:

Input ▼

```
SELECT order_num
FROM Orders
WHERE DATE_PART('year', order_date) = 2012;
```

Oracle has no `DATEPART()` function either, but there are several other date manipulation functions that can be used to accomplish the same retrieval. Here is an example:

Input ▼

```
SELECT order_num
FROM Orders
WHERE to_number(to_char(order_date, 'YYYY')) = 2012;
```

Analysis ▼

In this example, the `to_char()` function is used to extract part of the date, and `to_number()` is used to convert it to a numeric value so that it can be compared to `2012`.

Another way to accomplish this same task is to use the `BETWEEN` operator:

Input ▼

```
SELECT order_num
FROM Orders
WHERE order_date BETWEEN to_date('01-01-2012')
 AND to_date('12-31-2012');
```

Analysis ▼

In this example, Oracle's `to_date()` function is used to convert two strings to dates. One contains the date January 1, 2012, and the other contains the date December 31, 2012. A standard `BETWEEN` operator is used to find all orders between those two dates. It is worth noting that this same code would not work with SQL Server because it does not support the `to_date()` function. However, if you replaced `to_date()` with `DATEPART()`, you could indeed use this type of statement.

MySQL and MariaDB have all sorts of date manipulation functions, but not `DATEPART()`. MySQL and MariaDB users can use a function named `YEAR()` to extract the year from a date:

Input ▼

```
SELECT order_num
FROM Orders
WHERE YEAR(order_date) = 2012;
```