SQL语句MSSQL.MySQL.Oracle.PostgreSQL不同用法

作者: lovelong1 http://lovelong1.javaeye.com

SQL语句MSSQL.MySQL.Oracle.PostgreSQL不同用法

目 录

1.1 SQL语句MSSQL.MySQL.Oracle.PostgreSQL不同用法

发表时间: 2009-09-20 关键字: 数据库

The database vendor implementations shown in examples below (Microsoft SQL Server, MySQL, Oracle, and PostgreSQL) are discussed in our upcoming book SQL in a Nutshell. There are a great many function calls that are universally supported by the ANSI (American National Standards Institute) standard and all the database vendors. For example, most vendors support the commonly used aggregate functions of SUM, AVG, MIN, and MAX. These functions extract summary value, average value, and minimum or maximum value from a column or an expression, respectively. There are also a whole variety of functions that are not universally supported, such as RPAD and LPAD or SUBSTRING versus SUBSTR.

Although this article discusses database implementations by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL, this information represents just the tip of the iceberg on how business operations accomplish common, everyday SQL coding tasks using functions calls. As you will see, these functions can vary widely.

Date Operations
This first set of examples show how to query the database for common date-processing operations using functions. To get the current date and time:

```
Microsoft SQL Server


 SELECT GETDATE()
 GO
MySQL [retrieving the date but not time]
 SELECT CURDATE();
MySQL [retrieving date and time]
 SELECT NOW();
Oracle
 SELECT SYSDATE
 FROM dual;
PostgreSQL
 SELECT CURRENT_DATE;
```

As the examples illustrate, each vendor retrieves the current date and time differently using its own proprietary function calls. Microsoft SQL Server uses a SELECT statement calling the GETDATE() function. MySQL has two different function calls: CURDATE() and NOW(). The former retrieves the

date without time; the latter retrieves date and time. Oracle uses the SYSDATE function call. And PostgreSQL uses the SQL99 CURRENT_DATE function call. Note that for all of these function calls, no passed parameters are needed.

These next examples show how to find out what day a given date falls on:

```
Microsoft SQL Server

 SELECT DATEPART(dw, GETDATE())
 GO
MySQL
 SELECT DAYNAME(CURDATE());
Oracle
 SELECT TO_CHAR(SYSDATE,'Day')
 FROM dual;
PostgreSQL
 SELECT DATE_PART('dow', date 'now');
```

Microsoft SQL Server uses the DATEPART function call using the syntax DATEPART(datetype, date_expression). This function requires the type of date (month, day, week, day of week, and so on), as the first argument, and the date expression (either a column containing a date or an actual date value), as the second part. MySQL offers the DAYNAME(date_expression) as its function of choice for finding the day of the week for a given date value. Oracle requires that the date be converted into a character value using TO_CHAR, but allows the application of a format mask that returns the data of the week value. Conversions of this type in Oracle follow the syntax TO_CHAR(conversion_expression, 'datetype'). In this case, TO_CHAR can be used to convert any other datatype to character datatype, including INT and DATE datatypes. PostgreSQL accomplishes date conversion using the DATE_PART function to extract the day of the week from the date expression. The syntax is DATE_PART('text', timestamp), where text defines how the date is returned (in our example, as a day of the week), and timestamp defines the date expression.

Sometimes an application needs to know how far two dates are from one another. To determine how far away a date is from the current date (or any other date for that matter), either in the future or in the past, use these examples:

```
Microsoft SQL Server
```

```
 SELECT DATEDIFF(dd, '1/1/01', GETDATE())
 GO
MySQL
 SELECT FROM_DAYS(TO_DAYS(CURDATE()) -
 TO_DAYS('2001-11-25'));
Oracle
 SELECT TO_DATE('25-Nov-2000','dd-mon-yyyy') -
 TO_DATE('25-Aug-1969','dd-mon-yyyy')
 FROM dual;
PostgreSQL
 SELECT AGE(CURRENT_DATE, '25-Aug-1969');
```

Measuring the time span between two dates is a procedure best left to procedure calls. But again, the syntax varies widely between the vendors. Microsoft uses the DATEDIFF function to measure the time span between two dates (in the example, between January 1, 2001 and today's date). The syntax is DATEDIFF(datetype, start_date, end_date), where datetype is a code representing how the time span should be represented (days, weeks, months, and so on), the start_date is the date to measure from, and the end_date is the date to measure to. MySQL must use the somewhat more cumbersome FROM_DAYS and TO_DAYS functions in a nested format to tell the time span between two dates. Oracle very neatly allows date addition and subtraction. The only reason the TO_DATE function is even needed is that the operation is being performed on character strings. If the operation were performed against two columns of DATE datatype, then no TO_DATE conversion would be necessary and the subtraction operation would act directly on the date expression. PostgreSQL has a cool function called AGE(start_date, end_date) that tells the time span between two passed dates as parameters.

It is common procedure to retrieve a date in a different format mask (Mon, DD, YYYY; mm/dd/yy; dd/mm/yy; etc.). Here are some examples:

```
Microsoft SQL Server

 SELECT CONVERT(VARCHAR(11), GETDATE(), 102)
 GO
MySQL
 SELECT DATE_formAT( "2001-11-25", "%M %e, %Y");
Oracle
```

```
 SELECT TO_CHAR(SYSDATE,'dd-Mon-yyyy hh:mi:ss PM')
 FROM dual;
PostgreSQL
 SELECT TO_CHAR (timestamp(CURRENT_DATE),
 'dd-Mon-yyyy hh:mi:ss PM');
```

In these examples, the vendors use specialized function calls to retrieve date expressions in a specific format. Microsoft SQL Server uses the CONVERT function (though CAST could also be used) in the syntax of CONVERT(convert_to_datatype, source_expression, date_format), where the convert_to_datatype is the datatype to return in the query, the source_expression is the source that will be converted, and the date_format is a set of codes that Microsoft has set aside for specific date format masks. MySQL uses the DATE_formAT function in the syntax of DATE_formAT(source_expression, date_format). Oracle uses TO_CHAR, as shown earlier, in the syntax of TO_CHAR(source_expression, date_format). PostgreSQL also uses TO_CHAR, though somewhat differently in that the source_expression must be enclosed within the time-stamp subfunction, as shown in the example above.

String Operations

Often, an application may need to find one string within another string. This is one way of performing this operation across the different vendors:

```
Microsoft SQL Server

 SELECT CHARINDEX('eat', 'great')
 GO
MySQL
 SELECT POSITION('eat' IN 'great');
Oracle
 SELECT INSTR('Great','eat') FROM dual;
PostgreSQL
 SELECT POSITION('eat' IN 'great');
```

Microsoft SQL Server uses its own function, CHARINDEX, to extract values from other strings. In this example, it will return the starting position of one string, 'eat,' within another, 'great.' The syntax is CHARINDEX(search_string, searched_string, [starting_position]). MySQL and PostgreSQL both

accomplish a similar operation using the POSITION function, showing where 'eat' occurs within 'great.' Oracle uses the INSTR function, although the order of the passed parameters are reversed. Unlike the other vendors, Oracle requires the searched_string first, then the search_string.

It is often necessary to trim trailing and leading spaces from an expression in an SQL operation:

```
Microsoft SQL Server


 SELECT LTRIM('  sql_in_a_nutshell'),
    SELECT RTRIM('sql_in_a_nutshell      '),
    SELECT LTRIM(RTRIM('     sql_in_a_nutshell     ')
 GO
MySQL
 SELECT LTRIM('  sql_in_a_nutshell'),
    SELECT RTRIM('sql_in_a_nutshell      '),
    SELECT TRIM('     sql_in_a_nutshell      '),
    SELECT TRIM(BOTH FROM '     sql_in_a_nutshell     ');
Oracle
 SELECT LTRIM('  sql_in_a_nutshell'),
    SELECT RTRIM('sql_in_a_nutshell      '),
    TRIM('     sql_in_a_nutshell      ')
 FROM dual;
PostgreSQL
 SELECT TRIM(LEADING FROM '     sql_in_a_nutshell'),
     TRIM(TRAILING FROM 'sql_in_a_nutshell     '),
     TRIM(BOTH FROM '     sql_in_a_nutshell     ');
```

Microsoft SQL Server uses the LTRIM and RTRIM functions to remove spaces from the left or right side of an expression, respectively. When trimming spaces on both sides of an expression in Microsoft SQL Server, the LTRIM function must encapsulate the RTRIM function (or vice versa). MySQL and Oracle both use LTRIM and RTRIM, but differ from SQL Server in that spaces can be trimmed from both sides of an expression with the TRIM function. MySQL also allows TRIM with the BOTH operator to indicate that both left and right sides of the expression should be trimmed. PostgreSQL uses only the TRIM function and controls whether the left, right, or both sides should be trimmed using the LEADING, TRAILING, and BOTH operators, as shown in the example above.

The opposite of trimming spaces is to pad them into an expression. To pad in x number of trailing or leading spaces with the various vendors:

```
Microsoft SQL Server


 Not supported
MySQL
 SELECT LPAD('sql_in_a_nutshell', 20, ' '),
 RPAD('sql_in_a_nutshell', 20, ' ');
Oracle
 SELECT LPAD(('sql_in_a_nutshell', 20, ' '),
 RPAD(('sql_in_a_nutshell', 20, ' ')
 FROM dual;
PostgreSQL
 SELECT LPAD('sql_in_a_nutshell', 20, ' '),
 RPAD('sql_in_a_nutshell', 20, ' ');
```

In this example, the supporting vendors all use LPAD to insert spaces (or a character expression) on the left side of a string expression and RPAD to put them on the right. The syntax for MySQL, Oracle, and PostgreSQL is xPAD('string_expression1', length, 'string_expression2'), where string_expression1 is the string to have characters padded, length is the total length of the string, and string_expression2 is the characters to pad out.

An operation similar to pad is to substitute characters within a string with other characters:

```
Microsoft SQL Server [returns 'wabbit_hunting_season']


 SELECT STUFF('wabbit_season', 7, 1, '_hunting_')
 GO
MySQL [returns 'wabbit_hunting_season']
 SELECT
    REPLACE('wabbit_season','it_','it_hunting_');
Oracle [returns 'wabbit_hunting_season']
 SELECT
    REPLACE('wabbit_season','it_','it_hunting_')
 FROM dual;
```

```
PostgreSQL
 SELECT TRANSLATE('wabbit_season','it_','it_hunting_');
```

Microsoft SQL Server uses the STUFF function to overwrite existing characters. Using this syntax, STUFF(string_expression, start, length, replacement_characters), string_expression is the string that will have characters substituted, start is the starting position, length is the number of characters in the string that are substituted, and replacement_characters are the new characters interjected into the string. MySQL and Oracle both use the function call REPLACE, using the syntax REPLACE(string_expression, search_string, replacement_string), where every incidence of search_string found in the string_expression will be replaced with replacement_string. PostgreSQL uses the TRANSLATE function as a synonym of REPLACE.

Many times, a SQL statement must retrieve only a portion of a string. The following examples show how to extract 'duck_season' from the string 'wabbit_duck_season' for each vendor:

```
Microsoft SQL Server

 SELECT SUBSTRING('wabbit_duck_season', 7, 11)
 GO
MySQL
 SELECT
    SUBSTRING('wabbit_duck_season', 7, 11);
Oracle
 SELECT SUBSTR('wabbit_duck_season', 7, 11)
 FROM dual;
PostgreSQL
 SELECT
    SUBSTR('wabbit_duck_season', 7, 11);
```

In each example, the syntax for SUBSTRING (or SUBSTR) is essentially the same: SUBSTRING(string_expression, start, length), where string_expression is the expression or column to be searched, start is an integer telling the starting position, and length is an integer telling the database how many characters to extract.

Some vendors allow function calls that can structure an IF, THEN, ELSE result set within the query:

```
Microsoft SQL Server


 SELECT  CASE
     WHEN foo = 'hi'   THEN 'there'
     WHEN foo = 'good' THEN 'bye'
     ELSE 'default'
 END
 FROM t2
 GO
MySQL
 N/A
Oracle
 SELECT DECODE
    (payments_info,'CR','Credit','DB','Debit', null)
 FROM dual;
PostgreSQL
 SELECT CASE
     WHEN foo = 'hi'   THEN 'there'
     WHEN foo = 'good' THEN 'bye'
     ELSE 'default'
 END
 FROM t2;
```

Microsoft SQL Server and PostgreSQL support the powerful ANSI SQL command CASE. CASE has two usages: simple and searched. Simple CASE expression compares one value, the input_value, with a list of other values and returns a result associated with the first matching value. Searched CASE expressions allow the analysis of several logical conditions and returns a result associated with the first one that is true.

Simple comparison operation

```
 CASE input_value
 WHEN when_condition THEN resulting_value
```

```
 [...n]
 [ELSE else_result_value]
 END
Boolean searched operation
 CASE
 WHEN Boolean_condition THEN resulting_value
 [...n]
 [ELSE else_result_expression]
 END
```

In the simple CASE function, the input_value is evaluated against each WHEN clause. The resulting_value is returned for the first TRUE instance of input_value = when_condition. If no when_condition evaluates as TRUE, the else_result_value is returned. If no else_result_value is specified, then NULL is returned.

In the more elaborate Boolean searched operation, the structure is essentially the same as the simple comparison operation except that each WHEN clause has its own Boolean comparison operation. In either usage, multiple WHEN clauses are used, although only one ELSE clause is necessary.

Oracle supports its own extremely powerful IF, THEN, ELSE function call: DECODE. DECODE has a unique syntax along these lines, DECODE(search_expression, search1, replace1, search[,.n], replace,.n], default), where search_expression is the string to be searched; subsequently each search string is paired with a replacement string. If a search is successful, the corresponding result is returned. In our example, when returning a result set from the payments_info column, any incident of 'CR' will be replaced with 'Credit,' any instance of 'DB' will be replace with 'Debit,' and any other values will be replaced with a default value of Null.

Nulls Operations

Nulls are sometimes tricky business. Sometimes a company process, such as a query or other data manipulation statement, must explicitly handle NULLs. These examples show how to return a value specified when a field or result is null:

```
Microsoft SQL Server

 SELECT ISNULL(foo, 'value is Null')
 GO
```

```
MySQL
 N/A
Oracle
 SELECT NVL(foo,'value is Null')
 FROM dual;
PostgreSQL [allows you to write a user-defined function to handle this feature]
 N/A
```

Microsoft SQL Server uses the ISNULL function following the syntax ISNULL(string_expression, replacement_value), where string_expression is a string or column being searched and replacement value is the value returned if string_expression is NULL. Oracle uses a different function, NVL, but follows an almost identical syntax.

Alternately, there may be times when a NULL value is needed if a field contains a specific value:

```
Microsoft SQL Server [returns NULL when foo equates to 'Wabbits!']

 SELECT NULLIF(foo, 'Wabbits!')
 GO
MySQL
 N/A
Oracle
 SELECT DECODE(foo,'Wabbits!',NULL)
 FROM dual;
PostgreSQL
 SELECT NULLIF(foo, 'Wabbits!');
```

Aside from using CASE or DECODE to solve this problem, Microsoft and PostgreSQL allow use of the NULLIF function. The syntax is NULLIF(expression1, expression2), which tells the database that if expression1 equals expression2, then returns a NULL value.

Summary

There are a great many function calls that are universally supported by the ANSI standard and all the database vendors. This article has shown a variety of useful function calls available in database implementations by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL. For more details on ANSI standard functions, check out our book, SQL in a Nutshell.