A ListFind( ) function for SQL

Posted At : August 5, 2005 12:59 PM | Posted By : Mark Kruger
Related Categories: SQL tips, MS SQL Server, Coldfusion & Databases

First let me say that it is usually not a good idea to store lists in Database tables. Many novice web developers falls into the trap of treating the data in the database the way they treat it in their web application. They are used to using lists and list functions in logic loops to qualify users or statements, and they try to do the same thing in the Database. But SQL doesn't have any native list functionality. The equivalent of a "list" to a database is a table with a single row.

A while back I saw a security scheme implemented by storing a list of group ids in a character column of the users table. The code grabbed the groupIds column from the users table on login, then it selected from the "groups" table where group_id IN (1,2,4,8...). This actually works fine when dealing with individual users. But what if you wanted to grab all the users from a particular group? You end up with code that looks like "where groupIds LIKE '%,2,%' Or groupIds like '2,%' or groupIds LIKE '%,2' or groupIs = '2'. Why the 3 statements? Because you have to account for a group id of 20 *and* a case where 2 is alone or at the beginning or the end. There are other ways around this - none of them pretty (like storing beginning and ending comma's for example). Not to mention this sort of design makes DBA's pull out what little hair they have.

Now, having warned you against being "listy" in your database design, I'm going to show you one way of working with a list that might help in a case like that above. I know, I know, a little knowledge is a dangerous thing. Let's just assume you are forced to deal with a legacy schema that you have no power to change (ha).

>Creating a UDF in SQL

The trick is to figure out a way to loop through the list and compare individual values in it against the value in question. You are probably familiar with how to do this in Coldfusion. We use the function "listfind()". It takes a list, a value and an optional delimiter - as in listFind('fred,george,ron','ron',','). What You may not know is that SQL allows you to create functions with arguments and use them in your queries. So you can duplicate this behavior in your SQL code. I'm using MS SQL syntax here so be forewarned. If you are a postgres or Pl/SQL guru then feel free to modify the code sample and post it to this thread. It's always great to have extra samples.

A UDF in MS SQL has 3 parts - a declaration, a return and a body. The declaration contains your function name and any arguments. The return type is any of the SQL data types. Since we already know what our function looks like and what it returns, this part is easy.

```
CREATE FUNCTION listFind(
@list VARCHAR(8000),
@value VARCHAR(255),
@delimiter VARCHAR(10))

RETURNS int
```

As you can see we are going to mimic exactly the Coldfusion function syntax. We have 3 variables to pass in - a list, a value to compare and a delimiter. One important thing to note. The delimiter cannot be made optional. You *must pass in all arguments* and

you cannot *default* any arguments. You can probably find some ways around this - like hard coding the delimiter and making separate functions for each delimiter you need, but it's probably easier to just make sure and include it.

The "listFind()" function

Ok, we've got a good start, now lets add the meat of the UDF. The function body starts with "AS":

```
AS
BEGIN

-- 2 declarations
-- @pos will be the location in the list
DECLARE @pos int
-- @item will contain items from the list
DECLARE @item VARCHAR(255)
-- set @pos to 0 by default
SELECT @pos = 0

   ....logic and looping...
END
```

The first thing I do in the body is to declare and initialize any variables I need. I'm going to need 2. First, an "@item" variable is a placeholder for the individual items in the list. Second, the "@pos" variable will be a counter that represents the position in the list of the item.

The next step is to create a loop that handles each value in the list. To do this we will use the "DATALENGTH()" function. It returns the length of the string that is our list. Note, it does *not* return the length of the list - it's the length of the string that *contains* the list - so "1,2,3" is a datalength of 5, not 3.

```
WHILE (DATALENGTH(@list) > 0)
BEGIN
-- this is the current position of @item in the list
SELECT @pos = @pos + 1
-- IF the list length is greater than 1
IF CHARINDEX(@delimiter,@list) > 0
BEGIN
        -- item contains the value to compare
SELECT @item =
SUBSTRING(@list,1,(CHARINDEX(@delimiter, @list)-1))
        -- remove that particular item from the list
SELECT @list =
SUBSTRING(@list,(CHARINDEX(@delimiter, @list)
               + DATALENGTH(@delimiter)),DATALENGTH(@list))

       -- if the 2 items are the same return the position
IF @item = @value
         RETURN @pos
END

ELSE
      -- there is only 1 item in the list (or perhaps a zero length string)
      -- so we just compare them directly
BEGIN
       SELECT @item = @list
```

```
    SELECT @list = null
    IF @item = @value
       RETURN @pos
END
END
-- if we get this far the item is not in the list
SELECT @pos = 0
RETURN @pos
END
```

We loop until nothing is left in the list. We are going from the first item to the last item. We use "CHARINDEX()" and "SUBSTRING()". CHARINDEX() returns the first instance of an item in a string, and SUBSTRING() extracts part of a string from the whole. So the line that says:

```
SELECT @item =
   SUBSTRING(@list,1,(CHARINDEX(@delimiter, @list)-1))"
```

is the equivelent of "time = listFirst(list)" in Coldfusion. The line that reads:

```
SELECT @list =
SUBSTRING(@list,(CHARINDEX(@delimiter, @list)
   + DATALENGTH(@delimiter)),DATALENGTH(@list))
```

would be the equivalent of "list = listDeleteat(list,1)" in Coldfusion. As you can see the list functions in Coldfusion are a thing of beauty. In any case, as we are looping we are extracting items in the list into the variable "@item" and comparing them against @value. If we have a match we return the variable @pos - which will contain the position in the list. If we make it through the loop with not values matching we return 0. Here's the complete function.

```
CREATE FUNCTION listFind(
@list VARCHAR(8000),
@value VARCHAR(255),
@delimiter VARCHAR(10))

RETURNS int

AS
BEGIN
-- 2 declarations
-- @pos will be the location in the list
DECLARE @pos int
-- @item will contain items from the list
DECLARE @item VARCHAR(255)
-- set @pos to 0 by default
SELECT @pos = 0

-- Loop over the commadelimited list
WHILE (DATALENGTH(@list) > 0)
BEGIN
-- this is the current position of @item in the list
SELECT @pos = @pos + 1
-- IF the list length is greater than 1
IF CHARINDEX(@delimiter,@list) > 0
BEGIN
    -- item contains the value to compare
SELECT @item =
SUBSTRING(@list,1,(CHARINDEX(@delimiter, @list)-1))
    -- remove that particular item from the list
SELECT @list =
```

```
SUBSTRING(@list,(CHARINDEX(@delimiter, @list) + DATALENGTH(@delimiter)),DATALENGTH(@list))

       -- if the 2 items are the same return the position
IF @item = @value
         RETURN @pos
END

ELSE
   -- there is only 1 item in the list (or perhaps a zero length string)
   -- so we just compare them directly
BEGIN
        SELECT @item = @list
       SELECT @list = null
        IF @item = @value
         return @pos
END

END
-- if we get this far the item is not in the list
SELECT @pos = 0
   RETURN @pos
END
```

How do you use it? Well there's 1 gotcha that relates to MS sql. If you are in the habit of not qualifying tables and objects it might make you a little frustrated. You must include the ownership qualifier for the function. In many cases this will be "dbo". So you would call the function like this: "select dbo.listFind('fred,george,ron,harry','sam',',')". Inside of a query it's the same thing - but you would have your columns to reference instead as in this example.

```
<cfquery name="getMyGroups" datasource="#dsn#">
SELECT
      dbo.listfind(groupids,'#groupId#',',') AS member
FROM     Users
</cfquery>
```

In the above query anyone with a "member" column greater than 0 is a member of the group. Or you could filter the query in the WHERE clause like this.

```
<cfquery name="getMyGroups" datasource="#dsn#">
SELECT    *
FROM       Users
WHERE
   dbo.listfind(base_columns,'#groupId#',',') > 0
</cfquery>
```

This would give you just the members of the group.

There you go. It's still not a great schema, but at least you have a work around.

Addendum: **UDF for list to table** - from Adam's comments below.