Welcome to the first installment of *Surviving SQL*, a semi-regular supplement devoted to tips and tricks of the SQL language itself. Some of you have asked to see more coverage of SQL topics, and your Editor and I have come up with this periodic "injection" to help satisfy your cravings! If you have an SQL question you'd like to see covered here, or even a brief tip or trick of your own you'd like to share, then send me an email at the address listed at the end of the article.

Since we have been discussing phonetic matching algorithms, let's take a closer look at the Soundex algorithm from an SQL standpoint. Obviously, a Soundex code is going to be attached to a record containing the actual name and all new records will get a Soundex code generated for them. Further, if we change an existing name, we will want a new Soundex code for that name. Sounds like an ideal candidate for a database trigger. Many databases have built-in Soundex functions which can easily be incorporated into a trigger.

As an exercise, and for those developers without a built in Soundex function in their database, let's look at how we can implement Soundex in SQL. Obviously, how we do this is going to depend greatly on the SQL language provided by the vendor.

The listing at the end shows an SQL implementation for Soundex using Microsoft SQL Server's Transact-SQL. In practice we'd be better off from a performance standpoint to write a SQL Server extended stored procedure in a Delphi DLL. But today we're interested in SQL, so let's write a standard stored procedure.

There are a few peculiarities in SQL we need to concern ourselves with. The first is the possibility of null input. Although the algorithm will ultimately produce a null result, we explicitly test for null input at the start and avoid the expense of running the entire algorithm.

The output Soundex code will always be a fixed-length four character code, but I deliberately used a `Varchar` datatype rather than a `Char` datatype for the output parameter. Why? Because it made the algorithm easier to code by appending the digits onto the end of the return string. With a `Char` datatype, the variable always contains four characters, even if it has to pad with spaces, so you'd have to specifically replace each character in position rather than simply append on the end. I did try this approach to see if any speed benefit could be gained and found no measurable difference in performance.

Another point to consider is that in Transact-SQL there is no such thing as an empty string. Empty strings are automatically changed to a single space character, so you can never have a zero-length `Varchar` string. To account for this in our algorithm, we initially set our return string to a null character (ASCII 0) using the `Char` function. Similarly we test for an ASCII 0 character in the return string when we're deciding whether or not we've got the first character.

The `Char` function, not to be confused with the `Char` datatype, is similar to Delphi's `Chr` function and returns the character represented by the given ASCII code. On the other hand, the `ASCII` function returns the ASCII code for a given character. Beyond these issues, the SQL version of our Soundex algorithm is very much the same as our Delphi version.

A final comment concerns vendors' built-in Soundex functions. Transact-SQL includes a Soundex function which I compared to the algorithm presented here. I found some cases where the built-in function produced different Soundex codes. The built-in function stops processing the input string at the first non-alpha character it finds, so the name Hale-Shaw returns H400 from Transact-SQL's function and H420 from ours. Also, the built-in function does not discard duplicate digit codes.

All three of my sources for the Soundex algorithm state that duplicate digits in the returned code should be collapsed into a single digit. But for the name Buchanan, the Transact-SQL function returns B255, whereas ours returns B250. These strike me as limitations of the built-in function and serve as another reason why you might be tempted to code your own Soundex function even if one is provided by your vendor.

```
create procedure MakeSoundex(@Word varchar(255),
@Code varchar(4) output)
as
  declare @I Int
  declare @Ch Char
  declare @LastCh Char
  declare @MaxCodeLength SmallInt
  declare @LetterCodes char(26)
begin
  if @Word is null begin
    select @Code = null
    return(0)
  end
                        /*ABCDEFGHIJKLMNOPQRSTUVWXYZ*/
  select @LetterCodes = '01230120022455012623010202'
  select @MaxCodeLength = 4
  select @Code = Char(0), @LastCh = Char(0), @I = 1
  select @Word = Upper(@Word)
  while DataLength(@Code) <> @MaxCodeLength
    begin
      if @I > DataLength(@Word)
        select @Code = @Code + '0'
      else begin
        select @Ch = Substring(@Word, @I, 1)
        if (@Ch >= 'A') and (@Ch <= 'Z')
          if @Code = Char(0)
            select @Code = @Ch
          else begin
            select @Ch =
              Substring(@LetterCodes, Ascii(@Ch)-64, 1)
            if (@Ch <> '0') and (@Ch <> @LastCh)
              select @Code = @Code + @Ch, @LastCh = @Ch
          end
        select @I = @I + 1
      end
    end
end
```

Steve Troxell is a software engineer with Ultimate Software Group in the USA. He can be contacted via email at Steve_Troxell@USGroup.com