Surviving Client/Server: Code Tables

by Steve Troxell

Almost all database systems of any size rely on code tables to translate data values into meaningful descriptions. For example, suppose the system tracks employee performance reviews. When printing or displaying the review information, you would like the word 'Excellent' to appear, but it is impractical to store the full text for every review for every employee in the database. So instead we store a shorter code value like 'EXC'. We might have a table called `Employees` with a field called `LastReviewRating` that contains the review code for each employee. The description of the review rating might be kept in a separate table called `Ratings` with fields `RatingCode` and `RatingDesc`, like that shown in Figure 1. We could then join the two tables as shown in Figure 2 to make a report of reviews for all employees.

Developers often refer to tables such as `Ratings` as code tables, reference tables, or lookup tables. Their main purpose is to provide a meaningful description to a database code value for screen or report display. This is all simple database design. But, as the complexity of the software increases, generally so do the number of code tables required to support it.

The Human Resources / Payroll system developed here at Ultimate Software Group sports nearly 300 tables, not counting code tables. Adding the individual code tables would bring the total closer to 500. That's a lot of tables to keep track of. The engineers at US Group, primarily Mike Barrera, have developed a novel technique for managing code tables which you may find useful in your projects whether you have 20 or 200 code tables. From this point on the context of the word 'table' could get confusing, so I'm going to start referring to the collection of values and descriptions for a given entity as a 'code list'. Figure 1 is the code list for review ratings.

Generic Code Lists

In practice, the majority of code lists are structurally similar: a short code value character string and a somewhat longer code description character string. So why not consolidate them all into one large list? Instead of a physical table for review ratings and a separate physical table for, say, employee status, we have a single table called `Codes` containing all the code lists, each identified by a keyword. In Figure 3 we have two *logical* tables (`RevRating`, `EmpStatus`) represented within one *physical* table (`Codes`).

RatingCode	RatingDesc
EXC	Excellent
SAT	Satisfactory
UNS	Unsatisfactory
POR	Poor

➤ *Figure 1: A code table for review ratings*

Now to reproduce the query we had in Figure 2, we join to `Codes` and specify the code list we want, as shown in Figure 4.

Granted this makes the joining slightly more complex due to the need to match two fields rather than one, but the impact of this can be minimized by indexing the

➤ *Figure 2: Joining the main table to the code table*

```
SELECT LastName, FirstName, RatingDesc
  FROM Employees E, Ratings R
  WHERE E.LastReviewRating = R.RatingCode
```

➤ *Figure 3: The general-purpose Codes table*

CodeList	CodeValue	CodeDesc
REVRATING	EXC	Excellent
REVRATING	SAT	Satisfactory
REVRATING	UNS	Unsatisfactory
RAVRATING	POR	Poor
EMPSTATUS	A	Active
EMPSTATUS	O	On Strike
EMPSTATUS	B	Leave of Absence
EMPSTATUS	L	Laid Off
EMPSTATUS	T	Terminated
EMPSTATUS	S	Suspended

➤ *Figure 4: Joining the main table to the Codes table*

```
SELECT LastName, FirstName, CodeDesc AS RatingDesc
  FROM Employees E, Codes C
  WHERE E.LastReviewRating = C.CodeValue AND C.CodeList = 'REVRATING'
```

CodeList	Description	TableName	CodeField	DescField
REVRATING	Review Rating	Codes	CodeValue	CodeDesc
EMPSTATUS	Employee Status	Codes	CodeValue	CodeDesc
DEDCODE	Deduction Code	Deductions	DedCode	DedDesc

➤ *Figure 5: The CodeDrivers table*

`CodeList` and `CodeValue` fields. On the other hand, it is often the case that the individual code lists contain a small number of values. So by consolidating them into one physical table we may actually be maximizing the total number of code values cached on the server at any one time. We are only dealing with the cache for a single table instead of a separate cache for each table.

The real value of this technique is simplified project management, although it may not seem apparent from the two code lists shown in Figure 3. When multiplied by the 1,051 codes spread across the 153 code lists in US Group's HRMS system, code maintenance is greatly simplified by having everything in one tidy package.

On the data-entry side of the system, the code values are presented to the user in the form of 'pick lists', or dropdown combo boxes. The dropdown lists are populated from the corresponding code lists in the table `Codes`. Since we have a central repository for all our various code lists, it is fairly straightforward to build a custom combo box component that lets us select the code list to use for populating the dropdown list. For example, we could have a combo box property called `CodeList` which tells the combo box to load its `Items` property from a particular code list in the table `Codes`. Designing a form containing a pick list becomes a simple matter of dropping down a combo box component and setting a single property.

Code Lists From Main Tables

However, the generic approach illustrated by the table shown in Figure 3 only handles simple code values and their descriptions. What if the system contains pick lists derived from more complicated data structures? For example, a screen for setting up employee payroll deductions would have a pick list for all the possible deductions that might appear on a payroll check. Employees might elect to have deductions for medical programs, life insurance, retirement plans, and so forth. The table defining all the possible deductions would certainly be much more complicated than the simple structure shown in Figure 3. The deduction table would include fields for plan account numbers, effective dates, deduction amounts and frequencies, and a host of other data necessary to manage the deductions. However, the pick list for deductions is still a simple code value and code description, even if those two simple fields are embedded within a more complicated table structure.

Why did I bring this up? This isn't any particular problem, we simply query the deduction code value and description from the deduction table rather than our nice, tidy generic table `Codes`. So we have to forego the convenience of our fancy custom combo box component which does the work of populating the dropdown list automatically. We just write a simple SQL query and populate it the old fashioned way. It's a special case. Or is it?

The CodeDrivers Table

The engineers at US Group addressed this by creating another table called a 'driver' table, which contains one row for every possible code list in the system. Some of those code lists would be defined in the table `Codes` and some in other dedicated tables. Figure 5 shows the general structure of the `CodeDrivers` table.

In the driver table, the `CodeList` field identifies the list of values we want. This is not necessarily the name of a physical table in the database, but simply an identifier by which we reference the code list. The `Description` field is a simple description used by our custom combo box component to help identify code lists. The `TableName` field refers to the physical table in which the code list can be found. Most of the time, the code list will be within the `Codes` table. In the case of the deduction codes, the values are found in the main table `Deductions`, which presumably contains additional fields to support deduction processing. The `CodeField` field contains the name of the field in the physical table in which we can find the code value (to store in the data tables). For example, the review rating codes are found in `Codes.CodeValue`. The `DescField` field contains the name of the field in the physical table in which we can find the text description of the code (to display in the pick list).

With this information we can easily write code to generate the appropriate SQL statement to retrieve the code list no matter where it's located. The logic breaks down to two cases: either the list is among the majority of code lists stored in the `Codes` table, or it's in a separate table specifically set up for it. Figure 6 shows

➤ *Figure 6: Dynamic SQL derived from CodeDrivers*

```
If we select the EMPSTATUS code table, we generate:
SELECT CodeValue AS Value, CodeDesc AS Descr
  FROM Codes
  WHERE CodeTable = 'EMPSTATUS'
  ORDER BY Descr
If we select the DEDCODE code table, we generate:
SELECT DedCode as Value, DedDesc AS Descr
  FROM Deductions
  ORDER BY Descr
```

CodeList	Description	TableName	CodeField	DescField	CustomFilter
REVRATING	Review Rating	Codes	CodeValue	CodeDesc	<null>
EMPSTATUS	Employee Status	Codes	CodeValue	CodeDesc	<null>
DEDCODE	Deduction Code	Deductions	DedCode	DedDesc	<null>
DEDCODE_FT	Deduction Code: Full Time	Deductions	DedCode	DedDesc	DedAvailability <> 'P'
DEDCODE_PT	Deduction Code: Part Time	Deductions	DedCode	DedDesc	DedAvailability <> 'F'

➤ *Figure 7: CodeDrivers for custom code lists*

examples of the SQL needed for these two cases.

As you can see, all of our code lists can be accounted for in the `CodeDrivers` table. Our pick list combo box component can examine the fields `TableName`, `CodeField` and `DescField` to generate a dynamic SQL statement to retrieve the dropdown list values. Our logic recognizes the table name `Codes` as a special case and knows to filter the rows based on the `Codes.CodeTable` field. Notice, also, that no matter what the names of the source fields are, we alias them to the consistent names `Value` and `Descr` in the result set.

Customizing The Code Lists

Sometimes the values available in any given code list may vary depending on the circumstances. For example, there may be more deductions available to full-time employees than part-time, retirement benefit deductions, for example.

Let's say that the `Deductions` table contains a field called `DedAvailability` which is set to one of three values: `F` for deductions available only to full-time employees, `P` for deductions available only to part-time employees, or `A` for deductions available to all employees regardless of status. Presumably, when we have a pick list for employee deductions on a screen, we want the values available in the pick list to vary depending on whether we are examining a full-time or part-time employee. We can support this by adding a field to `CodeDrivers` called `CustomFilter`. This field will optionally contain a filter statement to be used in constructing a `WHERE` clause for populating the code list. We then create two new entries in `CodeDrivers`: one for full-time employee deductions and one for part-time employee deductions, as shown in Figure 7. We leave the original code list 'Deduction Code' as a third entry to be used in cases where we don't care about the availability restriction. For example, we might need this for general purpose joining in a report of all employees.

Notice that all three deductions code lists derive from the same table: `Deductions`. However, the full-time and part-time lists are handled as special cases by the presence of a nonnull value in the `CustomFilter` field. If the custom combo box component detects a non-null value in this field, it adds a `WHERE` clause to the SQL statement it generated to filter the code list accordingly. Figure 8 shows the resulting SQL.

➤ *Figure 8: Custom SQL for deduction codes*

```
For DEDCODE:
SELECT DedCode as Value, DedDesc AS Descr
  FROM Deductions
  ORDER BY Descr
For DEDCODE_FT:
SELECT DedCode as Value, DedDesc AS Descr
  FROM Deductions
  WHERE DedAvailability <> 'P'
  ORDER BY Descr
For DEDCODE_PT:
SELECT DedCode as Value, DedDesc AS Descr
  FROM Deductions
  WHERE DedAvailability <> 'F'
  ORDER BY Descr
```

Conclusion

Code tables are a basic element of almost all database projects. The larger and more complex the project, the more useful a technique for simplifying the management and maintenance of those code tables becomes. The approach shown here has simplified project management for US Group and hopefully you will find elements of it useful in your projects as well.

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