Microsoft's new database modeling tool: Part 8

Terry Halpin   Pat Hallock
North Face Learning   InConcept

Abstract: This is the eighth in a series of articles introducing the Visio-based database modeling component of Microsoft Visual Studio .NET Enterprise Architect. Part 1 showed how to create a basic ORM source model, map it to a logical database model, and generate a DDL script for the database schema. Part 2 discussed the verbalizer, independent object types, objectified associations, and some other ORM constraints. Part 3 showed how to add set-comparison constraints (subset, equality and exclusion) and how exclusive-or constraints combine exclusion and disjunctive mandatory constraints. Part 4 discussed the basics of modeling and mapping subtypes. Part 5 discussed mapping subtypes to separate tables, and occurrence frequency constraints. Part 6 discussed ring constraints. Part 7 discussed indexes, constraint layers, and data types. Part 8 examines options for controlling how table, column, and other model element names are generated when mapping an ORM model to a relational model.

Introduction

This is the eighth in a series of articles introducing the database modeling solution in Microsoft Visio for Enterprise Architects, which is included in the Enterprise Architect edition of Visual Studio. NET. This article discusses how to edit and control the table, column, and other relational model element names generated by mapping from an ORM model. Familiarity with ORM and relational database modeling is assumed. For an overview of ORM, see [1]. For a thorough treatment of ORM and database modeling, see [2]. For previous articles in this series, see [3], [4], [5], [6], [7], [8], and [9].

Previous articles discussed how to automatically map ORM source models to logical (ER or relational) models. The next section reviews how to manually refine a generated logical model by reordering columns and renaming relational model elements, and how to migrate such changes back to the ORM source model. The following section then indicates why it is generally best to avoid renaming at the logical level. The rest of the article discusses how to avoid the need to rename relational model elements, by specifying options on the ORM source model to automatically generate the desired logical names when mapping.

Refining the logical model: reordering columns, and renaming

The ORM schema in Figure 1(a) was introduced in [9]. By default, the tool maps this conceptual schema to the relational schema shown in Figure 1(b).
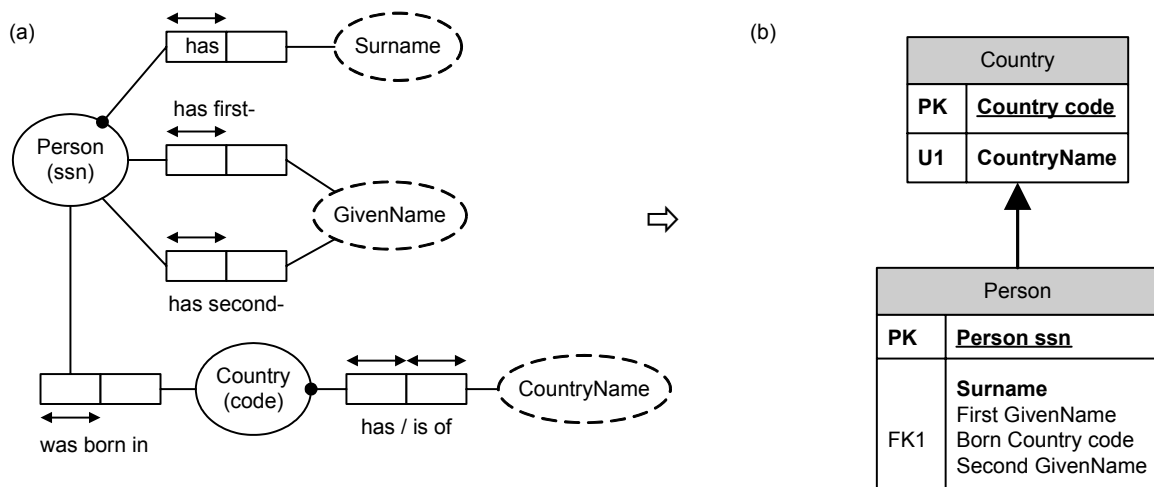

Figure 1   Default mapping of a simple ORM schema

The order in which columns are listed in the Person table may vary from that shown in Figure 1(b), depending on the order in which the fact types were entered in the ORM schema. In this particular mapping, the column order in the Person table is awkward, because the birth country column is displayed between the two given name columns. Also, some of the column names are not ideal. For example, "Born Country Code" would be better called "birthCountryCode".

As discussed in [8], you can finesse the logical model by moving and renaming. To refine a given table, open its Database Properties window, and make the desired changes. To move a column to a new position, choose the Columns category, select the column to be moved, and then press the Move Up or Move Down button to move it up or down respectively. To rename a column, simply edit the name displayed in the Physical Name field of the dialog. For example, Figure 2 shows a refined schema, with columns renamed in both tables, and a column moved in the Person table. In this figure, the Person table is selected, and the column entries for its Database Properties window are displayed.
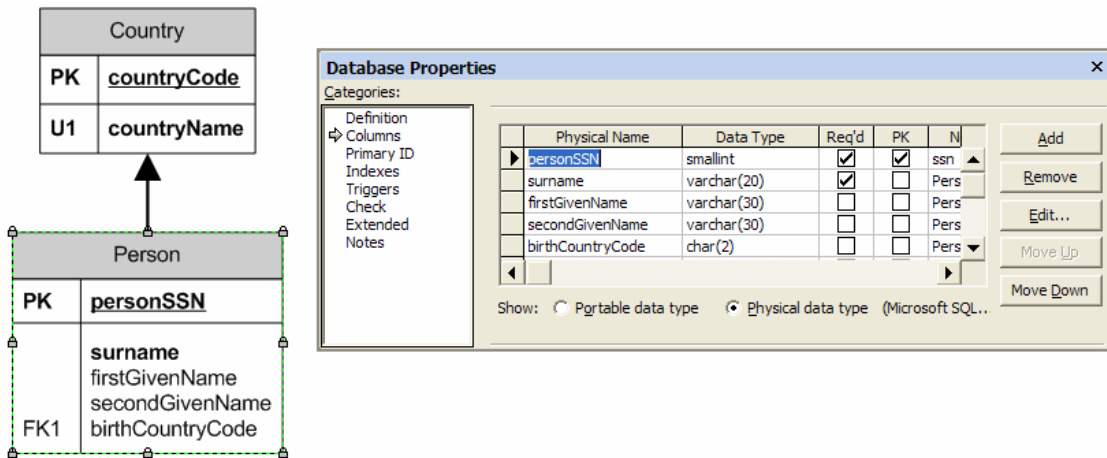

Figure 2   The Database Properties window may be used to move and rename columns

In addition to columns, you may rename other relational model elements such as tables, primary key constraints, indexes and foreign key constraints. To rename a table, choose the Definition category in its Database Properties window, and edit the Physical Name field. To rename a primary key constraint (e.g. Person_PK), select the Primary ID category and edit the Physical Name field. To rename indexes, choose the Indexes category, and edit the index name. To rename a foreign key constraint, select the foreign key arrow on the diagram, choose the Name category, and edit the Physical Name field of its Database properties window (see Figure 3).
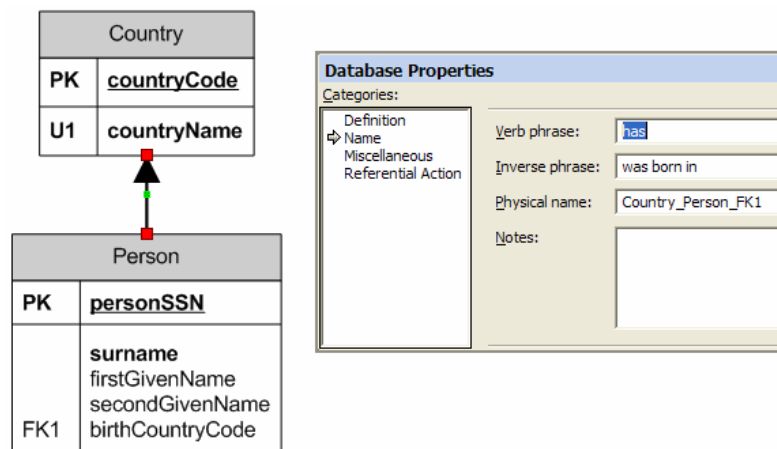

Figure 3   Foreign key constraints may be renamed

On saving the changes to the logical model, you are prompted whether to *migrate* these changes to the ORM source model used to generate the logical model. If you intend to reuse the same ORM model to build a different logical model, or if you have made any structural changes to the logical model (e.g. adding or deleting tables, columns or constraints, or changing the primary key) you should answer Yes to migrate. Otherwise these changes will be lost the next time you build or rebuild a logical model from the ORM source model.

If the only changes you made are non-structural (renaming columns, or reordering columns other than primary key columns), and you do not intend to reuse the ORM source model to build a separate logical model, you may answer No to migrate, since the non-structural modifications will be stored with the current project. Since migration can take a long time with large models, migrating changes only when needed can save you some time. That said, the bugs associated with migration in early versions of the tool have now been essentially eliminated, and you might discover later that you want to reuse an ORM source model in another project, so in most cases, answering Yes to migrate is normally advisable.

Avoid renaming at the logical level

As discussed in the previous section, you can refine a generated logical model by changing it directly. Although reordering columns is fine, you should *try to avoid making name changes directly to the logical model*. Instead, configure the ORM model to automatically generate the required names in the logical model. There are good reasons for controlling all naming decisions from the conceptual model. The conceptual model is easier to validate with the domain expert, so changes made at this level are typically best understood. Annotating the conceptual model with required naming options for target implementations facilitates the desirable approach of driving the execution directly from the conceptual model.

Another pragmatic reason is that the Visio tool currently exhibits the following undesirable behavior: *if name changes made at the logical level are migrated to ORM, any changes made later to names of the corresponding ORM model elements are ignored in subsequent builds of the logical model*. Hence renaming at the logical level can much it much harder to synchronize conceptual and logical models. To illustrate this problem, consider the following scenario. Suppose we create the ORM model in Figure 4(a), map it to the logical model in Figure 4(b), and then rename the columns as shown in Figure 4(c).
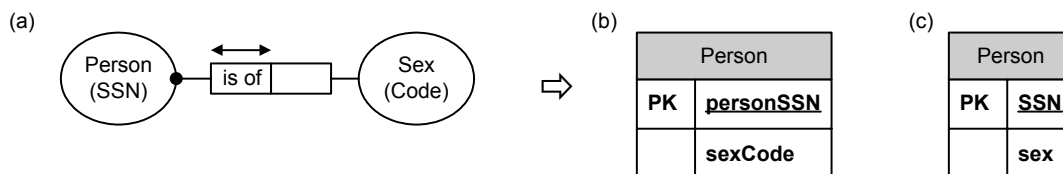

Figure 4   Mapping an ORM schema to a relational schema, then renaming columns

Suppose we now save the changes to the logical model, and answer Yes when prompted to migrate those changes to the ORM source model. This is fine if we never want to make changes to the relevant aspects of the ORM model. But suppose we later decide that social security numbers are not a good way to identify all persons in our universe of discourse, so we change the reference scheme to Person(Id). Also we decide that Sex is better renamed as Gender. The ORM model now appears as shown in Figure 5(a). On rebuilding the logical model, we would expect those changes to be reflected in the column names generated. Instead these changes are ignored, and the logical model still appears as shown in Figure 5(b).
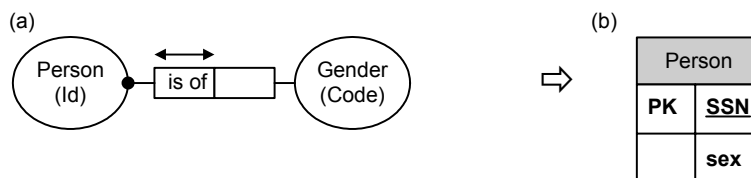

Figure 5   Later name changes to the ORM source are overridden by the migrated name changes

To avoid this undesirable behavior, you need to either say No when prompted to migrate logical name changes, or better still, configure the mapping options on the ORM model to automatically obtain the desired logical names so you can avoid renaming at the logical level. The rest of this article explores the second alternative.

Configuring ORM models to control name generation for logical models

Visio for Enterprise Architects includes several options for controlling how the names of logical model elements are generated from an ORM source model. In the pure ORM model itself, the names of model elements have a large bearing on the logical names generated. Most important in this regard are the names of object types, predicates, reference modes and roles. In addition, various mapping settings may be stored with the ORM model to control name generation. Settings made at the document level apply to the whole model unless overridden by local settings on individual object types and predicates. Most of the document level options are specified in the ORM Document Options dialog, while preferences for keys, indexes, and foreign key constraint names may be set in the Database Modeling Preferences dialog. We discuss these two dialogs first before moving onto the use of local settings and roles names to control name generation.

Document level options for name generation

To set mapping options that apply by default to the whole ORM model, open the *ORM Document Options dialog* by choosing Database > Options > Document from the main menu of the ORM model. This dialog has six panes: General, Abbreviation, Prefix, Suffix, Capitalization and Miscellaneous. The General pane has a check box for showing physical constraints: you should normally leave this checked (the default). The Prefix and Suffix panes allow you to automatically insert prefixes or append suffixes to column and/or table names in the logical model. You will normally want to accept the default settings (No prefix, No suffix), as shown in Figure 6. Some companies enforce naming standards that require tables or columns to be prefixed or suffixed. For example, to prefix each table name by "DEMO_", choose the Custom Prefix option for Table Prefix and enter "DEMO_" in the field. Custom column prefixes and suffixes are sometimes used help delineate context or sub-areas, but in the version used at the time of writing, there appear to be bugs with the generation of custom column prefixes/suffixes.
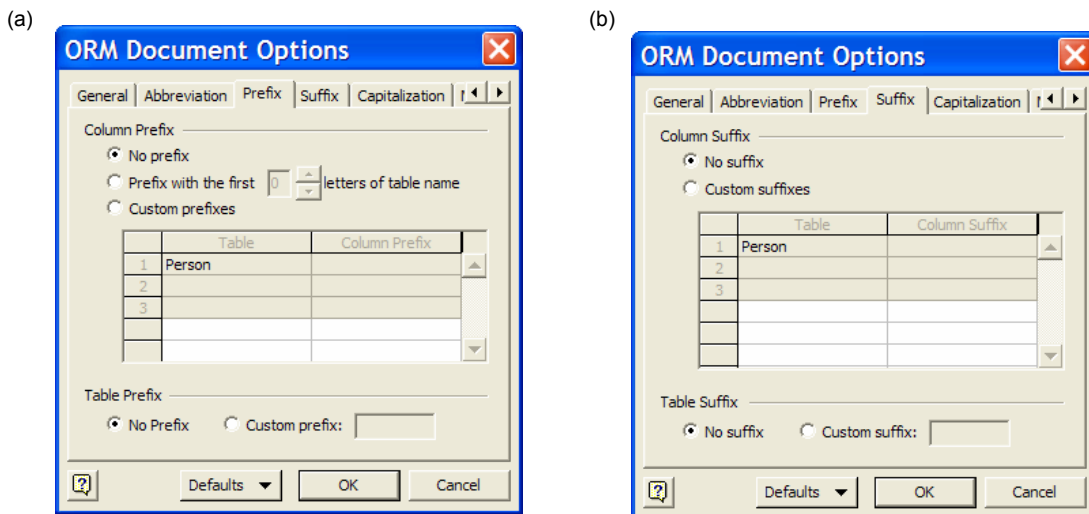
(a)   (b)

Figure 6   Setting prefix and suffix options at the ORM document level

The Capitalization pane allows you to control upper and lower case settings for column and table names. Different companies adopt different standards in this regard. The settings shown in Figure 7 are recommended if you prefer camel casing for column names (e.g. countryCode) and Pascal casing for table

names (e.g. LineItem). Here the case used in the conceptual names is preserved, except for the first letter, which is forced lower case for columns, and forced upper case for tables. To generate table names in all upper case (e.g. LINEITEM) choose the "Force upper" setting for Table Names.
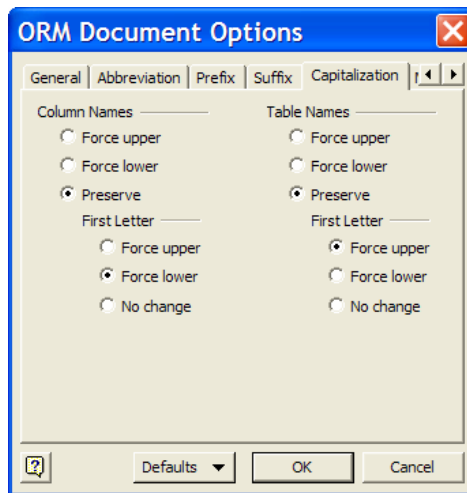

Figure 7   This setting starts column names in lowercase, and starts table names in uppercase

The Miscellaneous pane is even more useful. Although conceptual names may contain spaces, embedded spaces are often undesirable or even illegal at the logical level, unless delimited identifiers are used. In Figure 8, setting the Spacing Character to None removes all spaces from generated names for relational elements. For example, "First GivenName" becomes "FirstGivenName". To replace a space by an underscore (e.g. First_GivenName), choose the Underscore setting. The Other setting allows you to specify a different replacement character.

The default setting for Reference Mode is Add to object type name. For example, the reference scheme Country(Code) generates the name "CountryCode". This is usually best as a document wide setting. You can override this for individual object types, as discussed earlier. The Do not use for naming setting ignores the reference mode, instead using just the object type name, e.g. "Country". The Use as column name setting ignores the object type name and uses just the reference mode name, e.g. "Code".
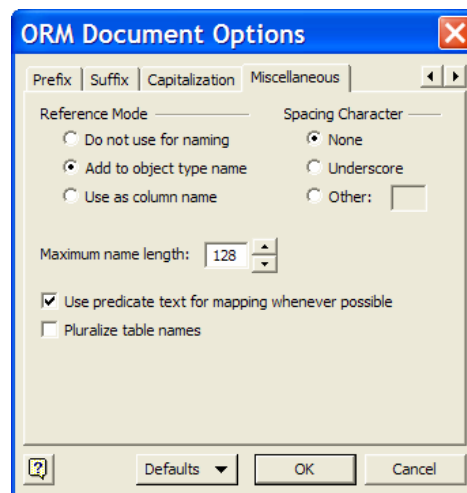

Figure 8   Setting miscellaneous naming options

The maximum name length setting specifies the maximum number of characters allowed in generated names. This defaults to 128, which is the maximum length for an identifier in the SQL-92 and SQL:1999 standards. Although a number of DBMSs, including Microsoft SQL Server, allow 128 character names,

some DBMSs do not (e.g. Oracle identifiers are restricted to 30 characters). It is not uncommon to generate names that are longer than 30 characters, especially for foreign key constraints or tables that correspond to an n-ary predicate. One way to avoid names that are too long for the target DBMS is to enter a smaller value (e.g. 30) for the maximum name length in this dialog. The tool then does its best to produce a shorter name that is still meaningful, mainly by eliminating vowels starting at the right hand end. For example, if you specify an independent object type with the conceptual name "Person_or_Organization_or_Company" and set a 30 character limit on logical names, then this object type maps to a relational table named "Person_or_Organization_r_Cmpny".

The Use predicate text for mapping whenever possible check box should normally be checked, as this helps to provide more meaningful names. The Pluralize table names check box should be unchecked, unless the naming standards in your company require plural table names. Singular table names are preferable, especially for referencing columns (e.g. "Person.surname" is more natural than "Persons.surname"). If you do specific plural names, the tool does a reasonable job of rendering these naturally. For example, "Country" is pluralized as "Countries", and "Mouse" is pluralized as "Mice".

The Abbreviation pane enables finer control over name generation, by enabling you to specify standard abbreviations for various words. This is especially helpful if your company naming standards require such standard abbreviations. The tool comes with a list of predefined abbreviations that are displayed when you open the Abbreviations pane. Each predefined entry in the Name column ("a", "an", …, "would") is abbreviated to the null string, as shown by the absence of an entry in the Abbreviation column (see Figure 9). This causes those words to be removed from any generated names. You can add your own entries to the abbreviation list, as shown in Figure 9(b). Here the words "country", "date" and "code" are to be abbreviated as "CNTRY", "DT" and "CD" respectively. Although such abbreviations like these may lose clarity, they may be imposed on you by company naming standards.

(a)   (b)
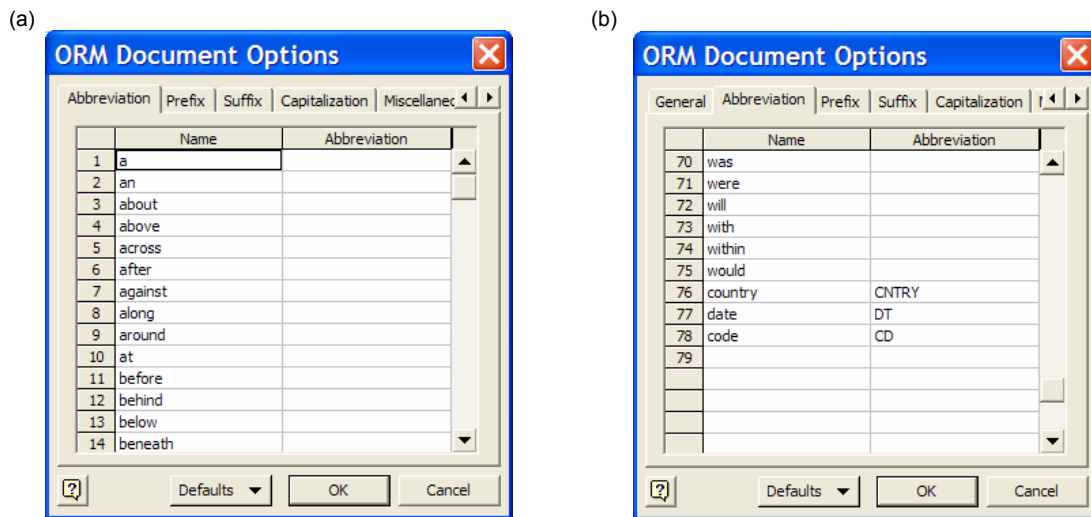

Figure 9   Predefined abbreviations (a) may be supplemented with user-defined abbreviations (b)

For example, Figure 10 shows the result of mapping the ORM schema considered earlier, using the settings discussed and the abbreviations "CNTRY" and "CD" for "Country" and "Code". The Country table is now named "CNTRY", and its primary key is named "cNTRYCD" instead of countryCode (the earlier decision to start column names with lower case does not work well with words that are all upper case). Note that the column "countryName" is not named 'CTRYName". This is because abbreviations work only on whole words, not parts of words. However, you can automatically generate the name "CNTRYName" in this case simply by including a space to separate the words in the conceptual name (i.e. name the ORM object type "Country Name" instead of "CountryName").
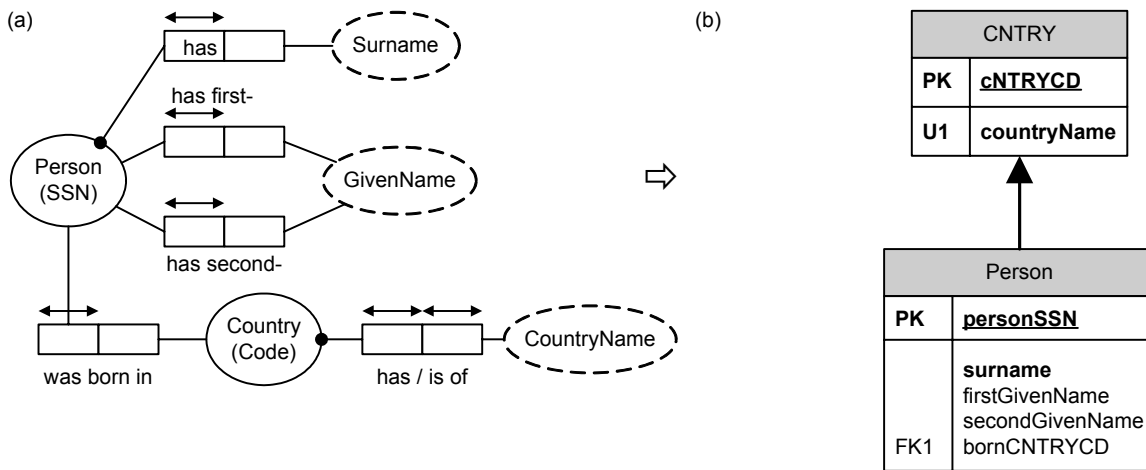
Figure 10   Using abbreviations for Country and Code

The foreign key of the Person table is now named "bornCNTRYCD". This uses both the user-defined abbreviations for Country and Code, as well as the predefined abbreviation of "was" to the empty string. The primary key of the Person table is named "personSSN", not personssn". This was achieved by changing the name of the reference mode to "SSN" instead of "ssn". Although the effect is not illustrated here, it is also best to use "Code" instead of "code" for the reference mode of Country, because the tool does not provide a capitalization option to start each successive word in a name with a capital letter.

The other document level dialog relevant to name generation is the *Database Modeling Preferences dialog*. This can be invoked from either an ORM source model or a logical database model using the main menu option: Database > Options > Modeling. Now select the miscellaneous pane (Logical Misc). For naming purposes, the relevant aspects of this pane are the Default name suffixes for keys and indexes, and the FK name generation option. Unless your company's naming standards differ, we suggest you use the predefined default suffixes (e.g. primary key constraint names are appended with "_PK"). The predefined default for foreign key constraint names appends the base suffix (e.g. "_FK") and a distinguishing numeral (e.g. "1" or "2" to cater for multiple foreign keys) to the concatenation of the names of the parent (referenced) table and child (referencing) tables. For example, as seen earlier in Figure 3, the foreign key constraint from Person to Country is named "Country_Person_FK1". If your target DBMS allows only short identifiers, you may wish to choose a shorter option from the drop-down list, as shown in Figure 11. You may also choose to start the constraint name with the name of the child table instead of the parent table.
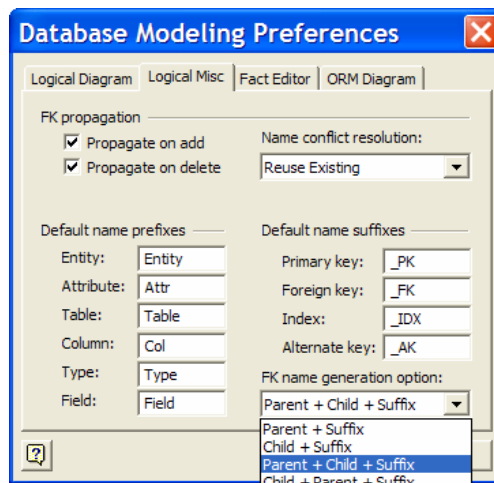

Figure 11   Setting suffix preferences for keys and indexes, and controlling FK constraint names

Local options for name generation

Some naming options set at the document level can be overridden on an individual basis for ORM model elements by choosing a different setting for them in their *Database Properties dialog*. For example, consider once more the ORM schema shown in Figure 1. By default, the name generated for the primary key of the Person table is "Person ssn" because the default document level setting for reference modes is to add them to the name of the object type (see Figure 8). To generate the column name "ssn" instead of "Person ssn", select the Person object type, and in the Ref Mode pane of its Database Properties dialog, change the radio button option from "Use document's setting" to "Use as column name" (see Figure 12). This overrides the reference mode document level setting for the Person object type only, so when you save this change and rebuild the logical model, the primary key of Person is named "ssn". If you choose the option Do not use for naming, the reference mode would be removed from the column name, so the primary key column would then be named "Person".
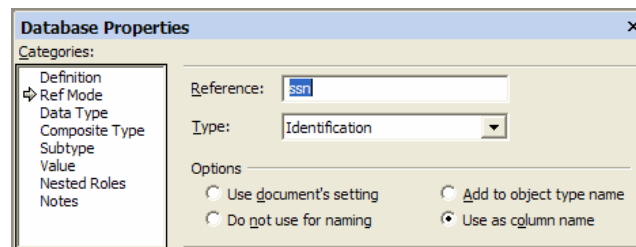

Figure 12   Choosing how the reference mode of an individual object type is used for column naming

The other document level naming option you can override on an individual basis is the setting to Use predicate text for mapping wherever possible (see Figure 8). For example, by default the column name generated from the Person was born in Country fact type in Figure 1 is "Born Country Code". The "was" and "in" from the predicate are omitted from the column name because their abbreviations are predefined to the null string. If for some reason you wanted to remove the whole predicate from the column name, you could open the Database Properties dialog for Person was born in Country and choose the Do not use option for Text use for naming (see Figure 13). If you then rebuild the logical model, the column name appears as "Country Code". This option is rarely used because it typically makes it harder to determine the precise meaning of the column names. If you apply this setting to more than one Country predicate that mapped to the same table, the tool appends numbers to the column names to distinguish them.
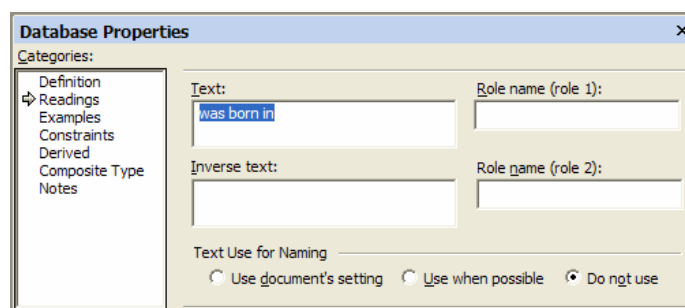

Figure 13   Choosing how the text of an individual predicate is used for name generation

Note that *naming option changes for an ORM object type or predicate are ignored if you previously migrated a logical name change back to that model element*. For example, suppose you changed "personssn" to "personSSN" at in the logical model, migrated that change back to the ORM model, then chose the "Use as column name" setting for Person's reference mode. If you now rebuild the logical model, you will still get "personSSN", not "ssn" as the column name. This is yet another reason to *avoid renaming at the logical level*.

Using role names to control column names

In an ORM model, each predicate must have at least one reading, but it is optional whether any role in a predicate is named. Although not displayed automatically on the diagram, role names are useful for controlling how column names are generated in logical models (and for specifying attribute-style derivation rules). For example, suppose we add the name "birthCountry" for the second role of Person was born in Country by entering it in the Readings pane of the predicate's Database Properties dialog, as shown in Figure 14. This role name will now be used in place of the predicate name when generating the relevant column name. For example, instead of "BornCountryCode" the column will be named "birthCountryCode". If you also set the reference mode naming option for Country to Do not use, the column will be named "birthCountry".
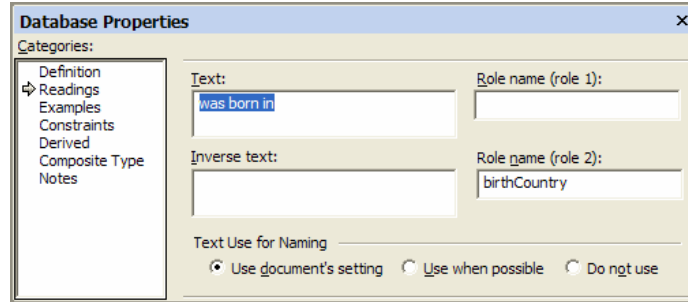

Figure 14   Using a role name to generate a column name

The tool uses role names to generate column names but not table names. If a predicate is either many:many or n-ary, it maps to a table all by itself, and role names are ignored in name generation. For example, consider the ORM schema in Figure 15(a). Here, bracketed role names are added in text boxes to the diagram for discussion purposes. The reference mode settings for Date and Country have also been set to Do not use, and the ORM document options have been set as shown in Figure 7 and Figure 8. The logical schema obtained from this mapping is shown in Figure 15(b). It would be nice to also control the table name "PersonvisitedCountry" from the ORM model, but the tool currently does not offer this option. However, the options discussed in this article will give you almost total control over how names are generated in the logical model. Remember, try to avoid as much as possible making any name changes directly to the logical model, because this makes it very awkward to make later changes to the corresponding elements in the conceptual model.
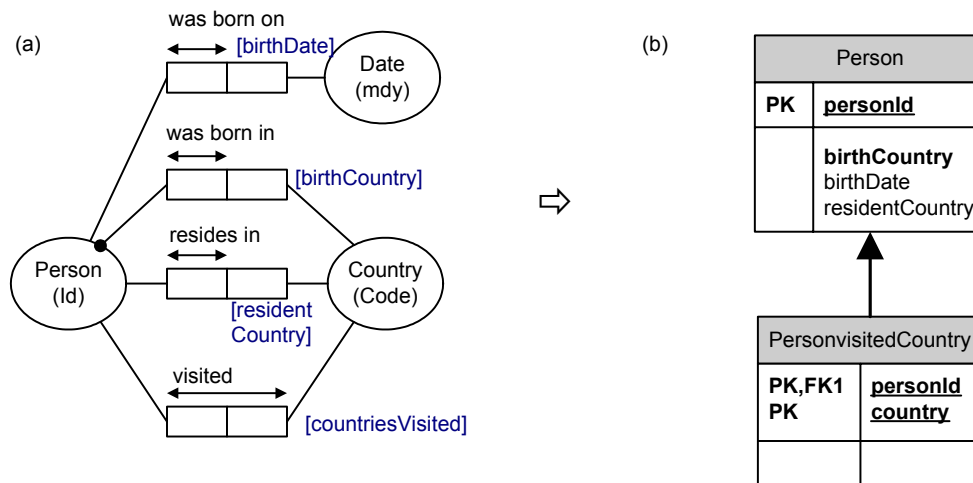

Figure 15   Using role names and local overrides to control name generation

Conclusion

This article provided a brief overview of how to edit and control the names of relational model elements that are generated by mapping an ORM model to a logical database model. If you have any constructive feedback on this article, please e-mail Terry Halpin at: thalpin@attbi.com.

References

1. Halpin, T. A. 1998 (revised 2001), 'Object Role Modeling: an overview', white paper, (online at www.orm.net).
2. Halpin, T.A. 2001a, *Information Modeling and relational Databases*, Morgan Kaufmann Publishers, San Francisco (www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672-6).
3. Halpin, T.A. 2001b, 'Microsoft's new database modeling tool: Part 1', *Journal of Conceptual Modeling*, June 2001 issue (online at www.InConcept.com and www.orm.net).
4. Halpin, T.A. 2001c, 'Microsoft's new database modeling tool: Part 2', *Journal of Conceptual Modeling*, August 2001 issue, (online at www.orm.net).
5. Halpin, T.A. 2001d, 'Microsoft's new database modeling tool: Part 3', *Journal of Conceptual Modeling*, October 2001 issue, (online at www.orm.net).
6. Halpin, T.A. 2002a, 'Microsoft's new database modeling tool: Part 4' (online at www.orm.net). This is a revised version of an earlier article of the same title in the January 2002 issue of *Journal of Conceptual Modeling*.
7. Halpin, T.A. 2002b, 'Microsoft's new database modeling tool: Part 5' (online at www.orm.net). This is a revised version of an earlier article of the same title in the March 2002 issue of *Journal of Conceptual Modeling*.
8. Halpin, T.A. 2002c, 'Microsoft's new database modeling tool: Part 6' (online at www.orm.net). This is a revised version of an earlier article of the same title in the May 2002 issue of *Journal of Conceptual Modeling*.
9. Halpin, T.A. 2002d, 'Microsoft's new database modeling tool: Part 7' (online at www.orm.net). This is a revised version of an earlier article of the same title in the July 2002 issue of *Journal of Conceptual Modeling*.

Note: Revised versions of many of the above references are also accessible online from the MSDN library (http://msdn.microsoft.com/library/default.asp). From the tree browser on the MSDN Library Home Page choose the following path to find these articles: Visual Tools and Languages > Visual Studio .NET > Visual Studio .NET (General) > Technical Articles.