Microsoft's new database modeling tool: Part 4

Terry Halpin
Microsoft Corporation

Abstract: This is the fourth in a series of articles introducing the Visio-based database modeling component of Microsoft Visual Studio .NET Enterprise Architect. Part 1 discussed how to create a basic ORM source model, map it to a logical database model, and generate a DDL script for the physical database schema. Part 2 discussed how to use the verbalizer, mark an object type as independent, objectify an association, and add some other ORM constraints to an ORM source model. Part 3 showed how to add set-comparison constraints (subset, equality and exclusion) and how exclusive-or constraints are obtained by combining exclusion and disjunctive mandatory constraints. Part 4 discusses how to add basic subtyping details to an ORM model and map them to a database schema.

Introduction

This is the fourth 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 specify that an object type is a subtype of another. It also examines some of the basic issues in mapping subtype information to a relational database. 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] and [5].

Basic subtyping

If the population of an object type *A* must always be a subset of the population of another object type *B*, then *A* is said to be a *subtype* of *B*, and *B* is said to be a *supertype* of *A*. In ORM, this is depicted visually by a solid arrow running from the subtype to the supertype. For example, consider Figure 1, which shows a fragment of a model used for a hospital information system. In this model, MalePatient and FemalePatient are each subtypes of Patient. The main reason for introducing a subtype is to declare that specific roles are played only by that type. For example, prostate status may be recorded only for male patients, and pregnancy counts and pap smear results are recorded only for female patients.

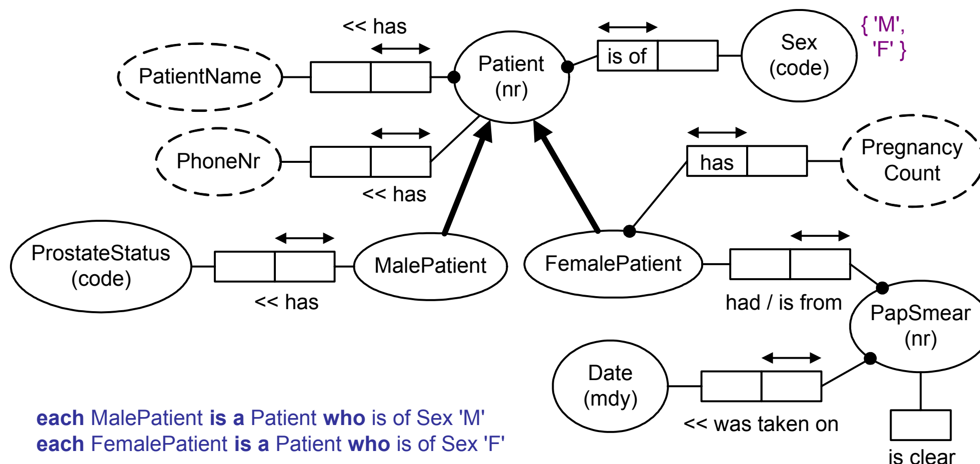

Figure 1 MalePatient and FemalePatient are subtypes of Patient

Other reasons for subtyping are to encourage reuse of supertypes, and to display a type taxonomy. Currently, the Visio ORM source model solution does not allow subtypes to be introduced purely for taxonomy reasons (i.e. merely to show a classification scheme). So each subtype must play at least one specific role. We plan to remove this restriction in a later release.

In strict ORM, each subtype must be supplied with a formal subtype definition, that enables membership in the subtype to be determined by reference to properties of its supertype(s). For example, Figure 1 includes the following definitions for the subtypes: **each** MalePatient **is a** Patient **who** is of Sex 'M'; **each** FemalePatient **is a** Patient **who** is of Sex 'F'. Given this definition, and the value constraint {'M', 'F'} on Sex, it follows that the subtypes are collectively exhaustive (their union is equal to their supertype). Given the subtype definition, and the uniqueness constraint (each Patient is of **at most one** Sex), it follows that the subtypes are mutually exclusive. So the subtypes form a partition of the supertype. In ORM, this implied partition constraint may be displayed as an xor-constraint between the supertype ends of the subtype connections (see circled X and mandatory dot in Figure 2). This notation is used because the xor constraint applies to roles in virtual predicates underlying the subtype connections.

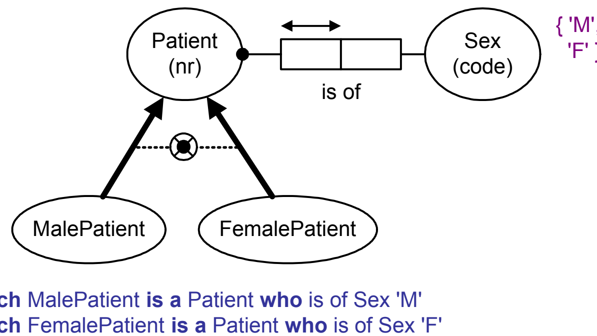

Figure 2 In principle, the subtype partition constraint is implied

This implied subtyping constraint approach enables far richer subtyping support than other approaches provide. Subtype definitions of arbitrary complexity may be expressed as queries in a formal ORM language such as ConQuer, enabling automatic generation of DDL code to enforce the constraints at the database level. This goes far beyond simple foreign key clauses and completeness and disjointness checks. For more details on this, see sections 6.5, 10.3 and 10.4 of [2].

In practice however, the Visio ORM tool does not yet support formal subtype definitions, or the automated display of implied exhaustion and exclusion constraints between subtype connections, or the full DDL code generation capabilities to map formal subtype definitions. It does however let you add subtype definitions as comments in the Subtype pane of the Database Properties sheet for the subtype. For example, to add a subtype definition for MalePatient, double-click that object type to bring up its properties sheet, select the Subtype category to bring up the Subtype pane, and then enter the definition in the Subtype definition field, as shown in Figure 3.

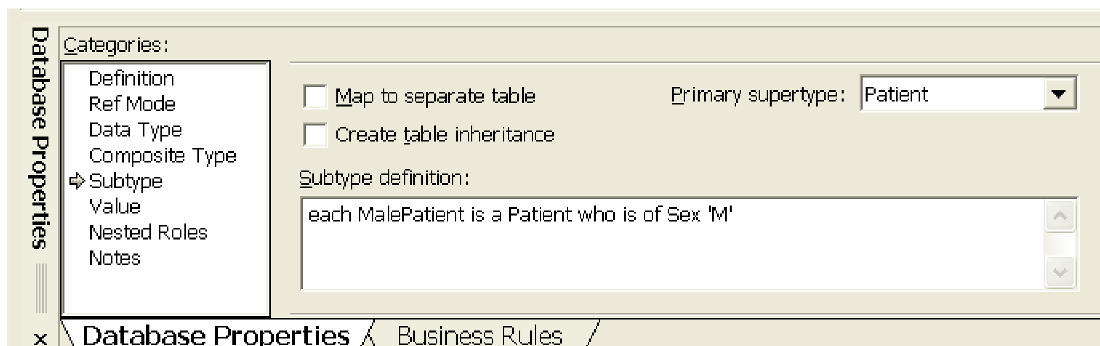

Figure 3 Adding a subtype definition

You can phrase the subtype definition in any way you like, because it is treated simply as a comment. If you do not enter a definition for each subtype, the tool issues a warning about this when a model error check is invoked (either directly or during a build). These subtype definitions are included in the automated verbalization, but do not appear on the diagram (unless you insert them in a text box, as I did to include them in the figures above).

If you want to display exhaustion and exclusion constraints between subtype connections, you need to do this manually using other Visio shapes and connectors (as I did to produce Figure 2). You can use the rich diagramming power of Visio to add as many adornments as you to like to the model diagrams. This can be very useful for documentation purposes. However any such annotations are ignored when a relational model is built from the underlying ORM source model, or when physical DDL code is generated.

In addition to the subtype definition field, the Subtype properties pane includes check boxes for table mapping and inheritance, and a list box for selecting the primary supertype (see Figure 3). The check box titled "Map to separate table" does not relate to the conceptual level at all. Instead it is used to control how subtype specific details are mapped to a relational database schema. If this box is unchecked (the default), any functional roles attached to the subtype will be absorbed back into the supertype when the model is mapped to a relational database schema. A functional role is a role with a simple uniqueness constraint (and hence it functionally determines the other role).

In Figure 1, MalePatient plays only one role, and this is functional and optional: **each** MalePatient has **at most one** ProstateStatus. For example, one male patient might have his prostate status recorded as 'BE' (benign enlargement), while another male patient has never had his prostate checked. FemalePatient plays two roles. One of these is functional (**each** FemalePatient has **at most one** PregnancyCount) and mandatory (**each** FemalePatient has **some** PregnancyCount). If a female patient has never been pregnant, this is recorded as a pregnancy count of 0. The other role played by FemalePatient is non-functional (it does not have a simple uniqueness constraint on it): **it is possible that the same** FemalePatient had **more than one** papSmear. The other role in this fact type is functional (**each** PapSmear is from **at most one** FemalePatient) and mandatory (**each** PapSmear is from **some** FemalePatient).

The first article in this series [3] discussed how to use the tool to map a conceptual ORM model to a logical database model (a relational database schema). By default, the ORM schema in Figure 1 maps to the relational schema shown in Figure 4. I've tidied up the names and order of the columns a little (how to control name generation and column order is discussed in a later article), but this is the structure you get by default if you leave the Map-to-separate-table options unchecked for the two subtypes. The prostate and pregnancy fact types are functionally dependent on their subtype, so are absorbed into the supertype table Patient. Hence the Patient table includes prostateStatus and pregnancyCount as optional columns. Recall that mandatory (not null) columns are displayed in bold, unlike optional (nullable) columns, and that "PK" denotes "primary key" and "FK" denotes "foreign key".

The pap smear fact type in Figure 1 is functionally dependent on the object type PapSmear, so maps to a table for that object type. The arrow between the tables is a foreign key reference or subset constraint (each patient number in the PapSmear table must also occur within the primary key of the Patient table). Although the ORM {'M', 'F'} value constraint on Sexcode is not displayed on the relational diagram, it is preserved in the underlying relational model—you can access this constraint by viewing the column properties sheet for the sex column, and it will appear as a check clause in the generated DDL .

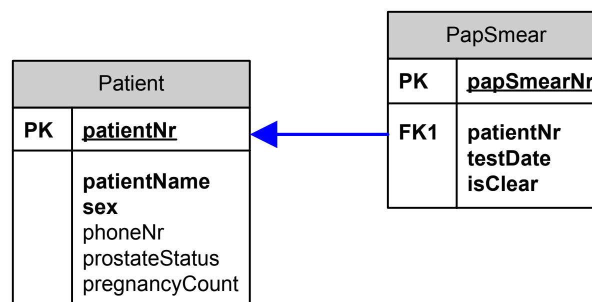

Figure 4 By default, functional subtype roles are absorbed into the supertype table

If you compare the ORM model in Figure 1 with the relational model in Figure 4, it is obvious that much of the subtyping semantics have been lost in the translation. The Patient table has three optional columns: phoneNr, prostateStatus and pregnancyCount. The phoneNr column is simply optional (there is no formal way of deciding which patients have their phone number recorded). But the prostateStatus and preganancyCount columns are not simply optional. Nor is the foreign key constraint a simple subset constraint.

If we are to preserve the additional semantics in the ORM source model, we need to add *qualifications* to any optional columns or subset constraints that result from subtyping. We could denote these qualifications by annotating the relational diagram as shown in Figure 5. Here the annotations have been added in simple Visio text boxes, using the relational constraint syntax discussed in [2]. Qualification 1 means that prostateStatus is recorded only if the patient is male (sex = 'M'). Qualification 2 means that pregnancyCount is recorded iff (if and only if) the patient is female (sex = 'F'). Qualification 3 means that each patient number in the PapSmear table must equal the patient number of a female patient recorded in the Patient table (sex = 'F').

.

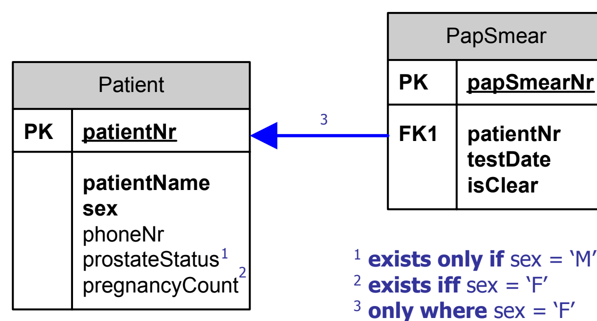

Figure 5 Subtyping leads to qualifications on optional columns or subset constraints

Because the ORM tool does not yet support formal subtype definitions, it cannot generate the code to enforce these qualifications. So for now, you need to write this code for yourself. You can do this by editing the table properties of the relational model before generation (or less preferably, by editing the DDL that is generated from the relational model). For example, qualification 1 may be implemented by the following check clause on the Patient table: **check**(prostateStatus **is null or** sex = 'M').

This involves more than one column, so requires a table-check clause rather than a column-check clause. To add the check clause for prostate status, proceed as follows. Click the Patient table to bring up its database properties sheet, and select the Check category, as shown in Figure 6.

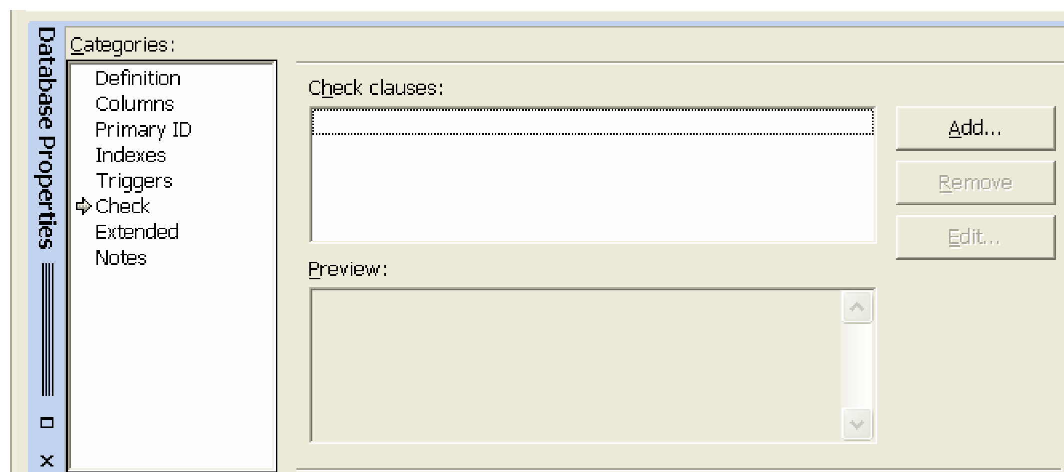

Figure 6 To add a table level check clause, hit the Add button

Now hit the Add button to bring up the code editor. Select its Properties pane and enter a meaningful name for the check constraint. In this case, I chose the name "ProstateStatusOnlyIfMale" (see Figure 7).

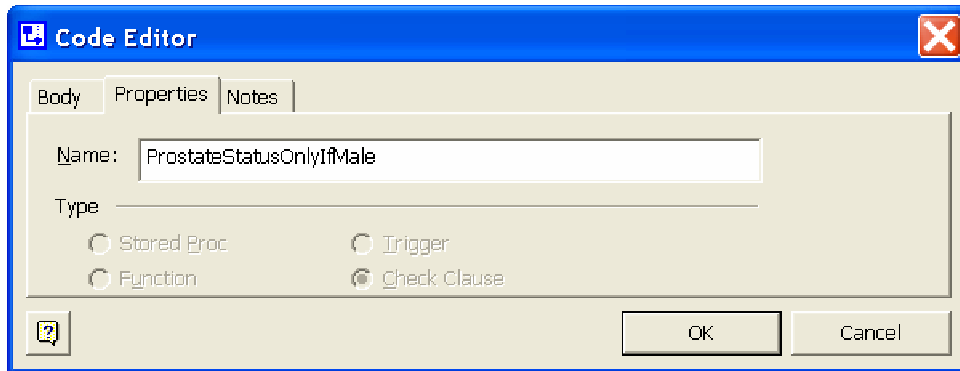

Figure 7 Naming the check constraint in the code editor

Now select the Body pane, and enter the body of the check-clause, as in Figure 8. Do *not* enter the "check ( )" wrapper for this code, since the tool does this automatically when generating the DDL. If you do include this wrapper, it will be treated as part of the code body, and hence generate an error.

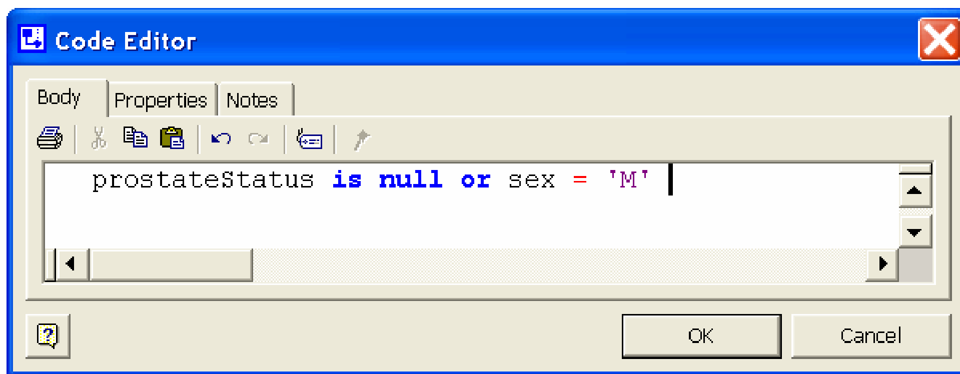

Figure 8 Adding the body of a check-clause

Hit OK to enter the check-clause. This returns you to the properties window, with the check-clause listed as shown in Figure 9.

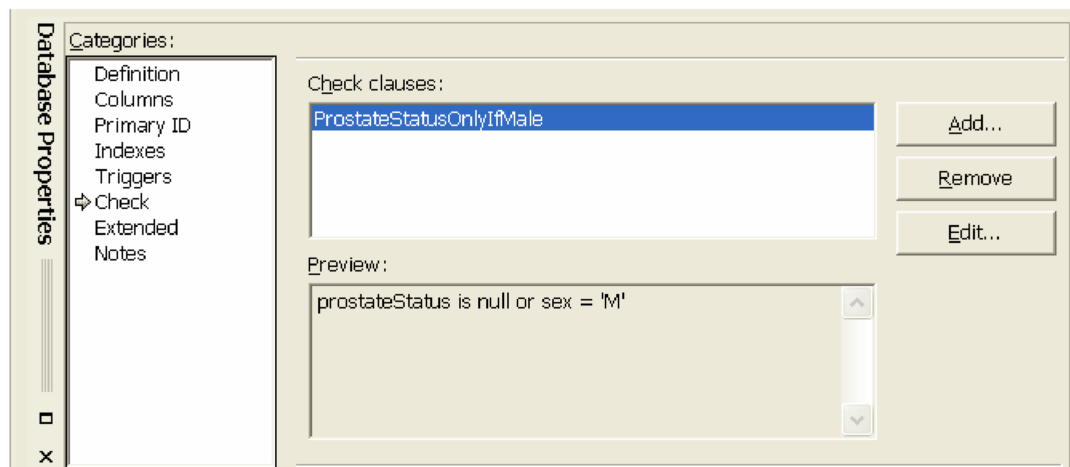

Figure 9 The check clause has been added

You may now remove or edit the check-clause, or add more check clauses in a similar way. To implement qualification 2, you should add the following two check clauses. Try this for yourself. Suggested names for these constraints are appended as comments.

```
check( pregnancyCount is null or sex = 'F' )          -- PregnancyCountOnlyIfFemale

check( sex <> 'F' or pregnancyCount is not null )     -- PregnancyCountIfFemale
```

If you add the three check-clauses as discussed, the following code is included in the generated DDL after the create-table clause for the Patient table:

```
/* Create table level checks for table Patient.   */

alter table "Patient" add constraint ProstateStatusOnlyIfMale check ( prostateStatus is null or sex = 'M' )

alter table "Patient" add constraint PregnancyCountOnlyIfFemale check ( pregnancyCount is null or sex = 'F' )

alter table "Patient" add constraint PregnancyCountIfFemale check (  sex <> 'F' or pregnancyCount is not null )
```

Qualification 3 (restricting the foreign key reference to female patients) can be implemented by using the database properties window to add appropriate triggers to the Patient and PapSmear tables. Triggers are a topic for a later article, so I'll omit details of that here.

Conclusion

This article discussed how to specify and map simple subtyping schemes using Microsoft's database modeling tool. The default mapping choice to absorb functional subtype facts into the supertype table is often advisable, since it avoids the need to perform a join when a query requires functional details of both the supertype and the subtype. However, sometimes it is better to map the functional details for a subtype to a separate table. This is what the Map-to-separate-table check box is for (Figure 3). This alternative mapping choice, and some advanced aspects of subtyping, will be explored in the next article. If you have any constructive feedback on this article, please e-mail me at: TerryHa@microsoft.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.InConcept.com and 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.InConcept.com and www.orm.net).

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.