Microsoft's new database modeling tool: Part 5

Terry Halpin
Microsoft Corporation

Abstract: This is the fifth 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 discussed how to add basic subtyping details to an ORM model and map them to a database schema. Part 5 discusses mapping subtypes to separate tables, and occurrence frequency constraints.

Introduction

This is the fifth 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 map functional details of subtypes to separate tables, and how to add occurrence frequency constraints to 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] and [6].

Mapping subtypes to separate tables

Figure 1 shows a simple ORM model about hospital patients. The subtypes MalePatient and FemalePatient are introduced to declare that prostate status may be recorded only for male patients, and pregnancy counts and pap smear results are recorded only for female patients.
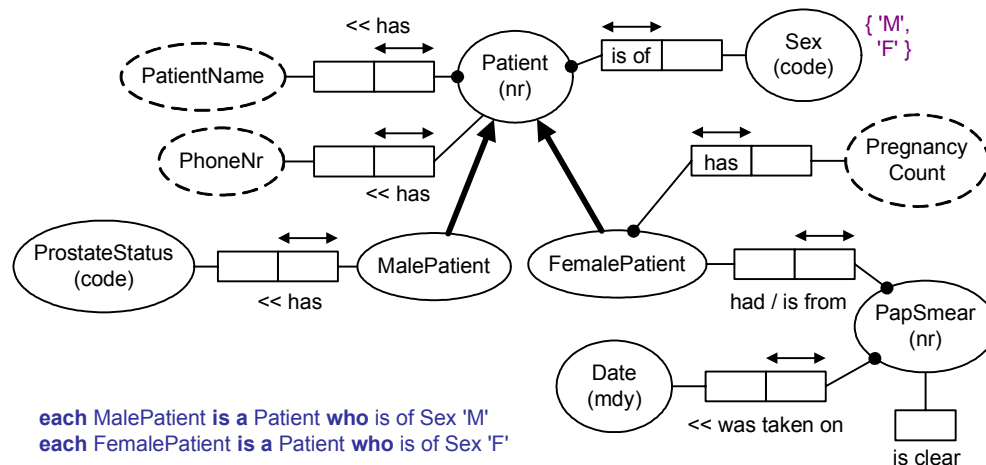

Figure 1   MalePatient and FemalePatient are subtypes of Patient

The previous article [6] discussed how this model may be mapped to the relational schema shown in Figure 2, using the default mapping procedure, where subtypes roles that are functional (with a simple uniqueness constraint) are effectively absorbed back to the supertype before mapping. 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. The actual subtype constraints (indicating the conditions under which subtype facts may be recorded) are now captured by qualifications on the optional prostateStatus and pregancyCount columns, and on the subset constraint depicted as a foreign key relationship from PapSmear.patientNr to Patient.patientNr. These qualifications need to be coded as check clauses or triggers.
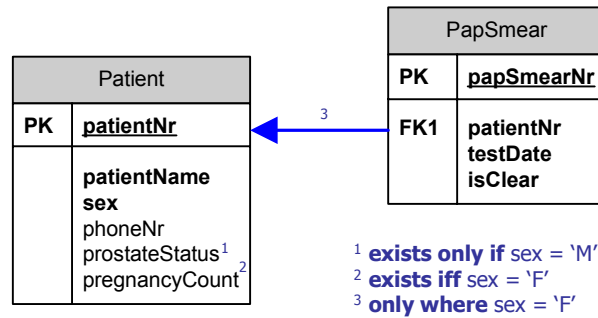

Figure 2   By default, functional subtype details are absorbed into the supertype table

If you double-click a subtype to bring up its properties sheet, select the Subtype category to bring up the Subtype pane, and then check the "Map to separate table" option, this will ensure that fact types that functionally depend on the subtype map to a separate table, with the primary identifier of the subtype as the primary key. For example, to specify a separate table for the MalePatient subtype, mark the check box as shown in Figure 3.
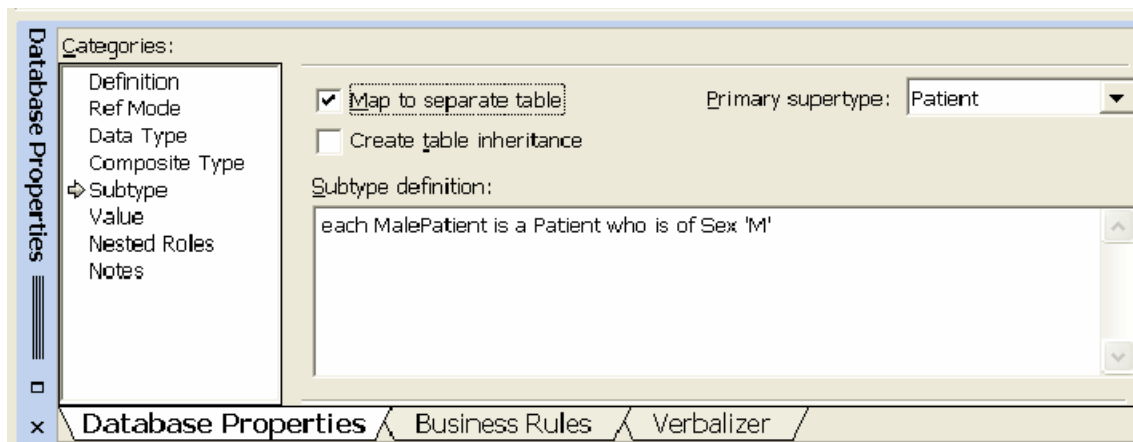

Figure 3   Declaring that functional details of MalePatient are to map to a separate table

If we chose the separate table mapping option for both the MalePatient and FemalePatient subtypes, we obtain the relational schema shown in Figure 4. Prostate status is now stored in the MalePatient subtable and pregancyCount is now stored in the FemalePatient subtable. Pap smear facts are still stored in a separate PapSmear table, because these are a function of PapSmear rather than FemalePatient (a patient may have many pap smears). When you choose the separate subtable mapping option, the tool uses a different notation for displaying foreign key relationships from subtables to supertable. Instead of arrows, a circle-bar notation is used, as shown. The circle is connected by a line to the supertable and has one or two bars underneath, connected by lines to the subtables.
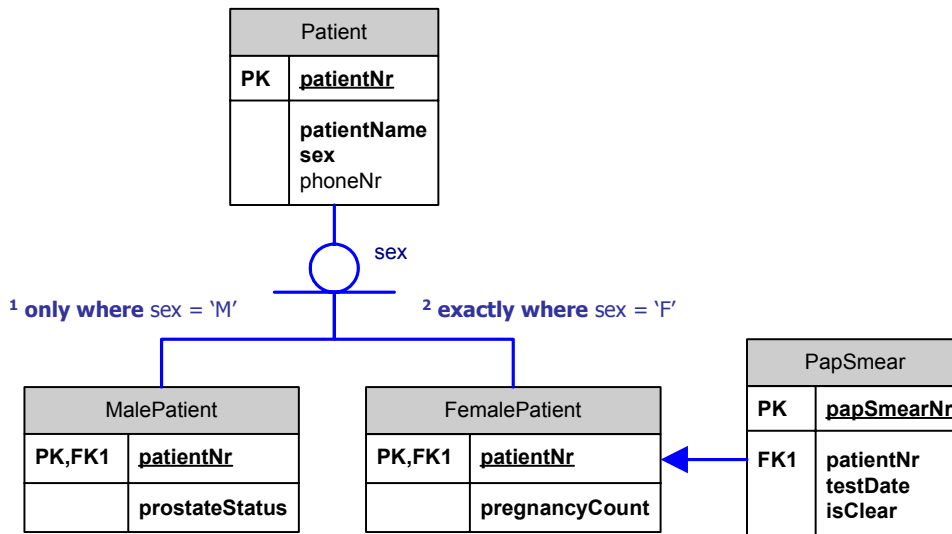
Figure 4   Functional subtype details are now mapped to separate tables

If you wish, you may specify a supertable attribute as a discriminator for the subtable hierarchy, by clicking the circle-bar icon to bring up its database properties window, and selecting the discriminator from the attribute list displayed, as shown in Figure 5. The tool displays the discriminator (in this case "sex") besides the circle-bar icon. If desired, you can drag the control handle for this shape to reposition the discriminator on the diagram. A single bar beneath the circle indicates that the categorization of a supertable into subtables is incomplete. In other words, it is possible that the union of the subtable primary key populations is a proper subset of the supertable primary key population. This is true for our example, because recording of prostate status is optional for males, so there may be male patients recorded in the Patient table that are absent from both the subtables. Constraints are on populations, rather than types. If we had made it mandatory for males to have their prostate status recorded, the categorization would be complete, and this could be declared by checking the "Category is complete" check-box in the properties sheet. The tool displays completeness of categorization as a double bar instead of a single bar.

The use of discriminators and completeness indicators covers only a fragment of ORM's subtyping semantics, which allows subtype definitions of arbitrary complexity (e.g. involving multi-branched paths through ORM space). However since the tool does not yet support formal ORM subtype definitions, it will not generate a discriminator, or guarantee the correct completeness setting when you map to the logical level. So if you want these aspects displayed properly you need to look after them manually yourself.
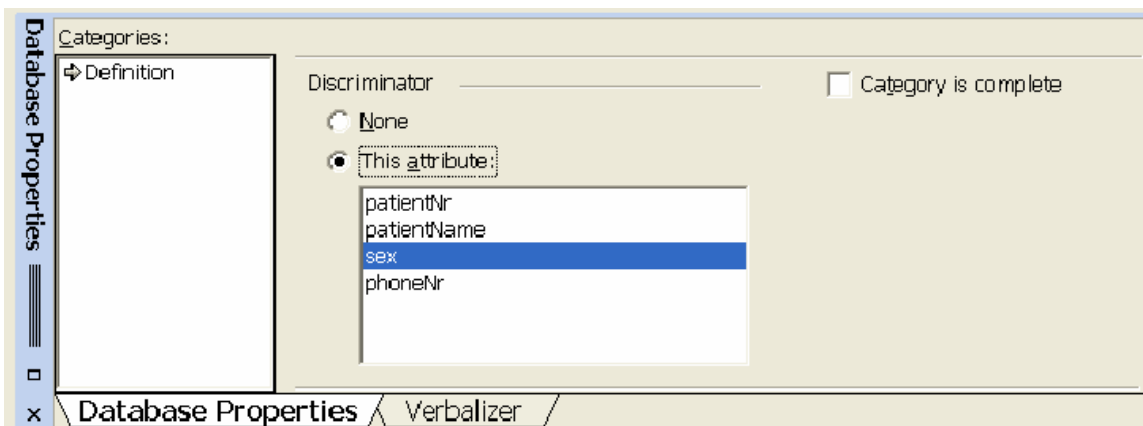

Figure 5   Specifying a subtyping discriminator and completeness status at the logical level

Even for the simple example in Figure 4, the discriminator and incompleteness settings do not convey the full ORM subtyping semantics captured in Figure 1. To preserve the additional ORM semantics, we need to qualify the subtable foreign key references as shown in Figure 4. Here the annotations have been manually added in simple Visio text boxes, using the relational constraint syntax discussed in [2]. Qualification 1 means that the set of patient numbers in the MalePatient table must be a subset of the set of patient numbers of male patients in the Patient table; so prostateStatus is recorded only where the patient is male (sex = 'M'). Qualification 2 means the set of patient numbers in the FemalePatient table must equal the set of patient numbers of female patients recorded in the Patient table; so pregnancyCount is recorded exactly where the patient is female (sex = 'F').

Since pap smear tests are optional for female patients, there is no need to qualify the foreign key reference from the PapSmear table to the FemalePatient table. This foreign key connection is just an unqualified subset constraint, so will be enforced as a simple foreign key declaration generated in the DDL.

Because the ORM tool does not yet support formal subtype definitions, it cannot generate the code to enforce the two qualifications on the subtable foreign key references. 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). As neither of these qualifications can be implemented as a check-clause, we postpone their discussion until our coverage of triggers and stored procedures in a later article.

Frequency constraints

In ORM, an occurrence frequency constraint may be used to declare how many times an entry may occur in the population of a role, or role combination. The number of times may be a simple integer (e.g. 2), a bounded range (e.g. 2..5) or an unbounded range (e.g. >=2). Consider the ORM model in Figure 6. Here each patient is allocated to at most one test group. The ">= 5" frequency constraint next to the role played by TestGroup means that any test group that does play this role must do so at least 5 times. If you populate this fact type, each entry in the TestGroup column must appear there at least 5 times. So, non-empty test groups must include at least five patients. Be careful in choosing the role(s) to which the constraint applies. If in doubt, populate the relevant fact types to clarify the meaning of the constraint.

The quaternary fact type in Figure 6 is used to maintain a history of patient's blood pressure readings. BPtype (Blood pressure type) is identified by a code (D = Diastolic, S = Systolic). The "2" frequency constraint besides the role connector linking roles played by Patient and Time indicates that any given (patient, time) pair that populates that role pair does so exactly 2 times. In the context of the 3-role uniqueness constraint and the {'D', 'S'} value constraint on BPtype, this ensures that any time a patient has his/her BP recorded, both diastolic and systolic readings must be recorded.
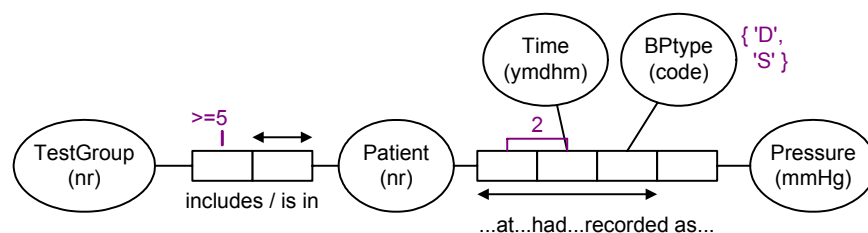

Figure 6   Two frequency constraints

To add these constraints to the fact types, hold the shift key down as you select both predicates, then right-click, and choose Add Constraints from the context menu. This causes the Add Constraints dialog to appear with both predicates included, as shown in Figure 7. To add the first frequency constraint (each test group has at least 5 patients), choose Frequency as the Constraint type, select the TestGroup role, set the Minimum frequency to 5, and delete any entry in the Maximum frequency box (no entry here means no upper limit). Read the constraint verbalization to ensure this is consistent with your intention, and hit the Apply button. The tool will now display ">=5" next to the relevant role on the ORM diagram, and reset the entries in the Add Dialog window.
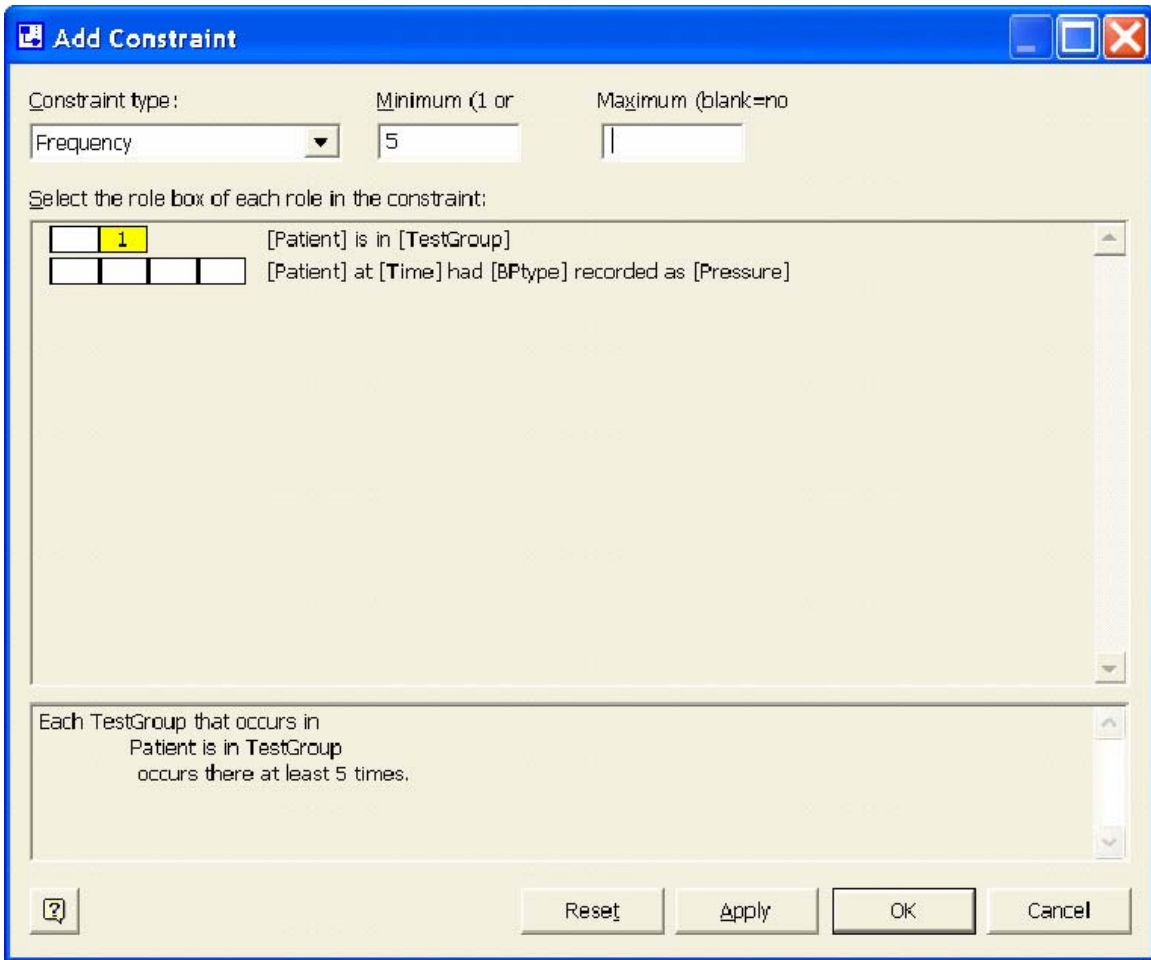
Figure 7 Adding the frequency constraint that each test group includes at least 5 patients

To add the frequency constraint on the quaternary predicate, choose Frequency as the Constraint type, select the Patient and Time roles, and then set both the Minimum frequency and Maximum frequency to 2, as shown in Figure 8. Check the verbalization (not shown here) and then hit OK to apply the constraint and exit the dialog. A "2" should now appear next to the Patient-Time role connector as shown in Figure 6. If desired, you can reposition things by using standard Visio controls (e.g. use Flip Vertical on the predicate to move its uniqueness constraint to the other side, and select and drag relevant constraints and predicate text).
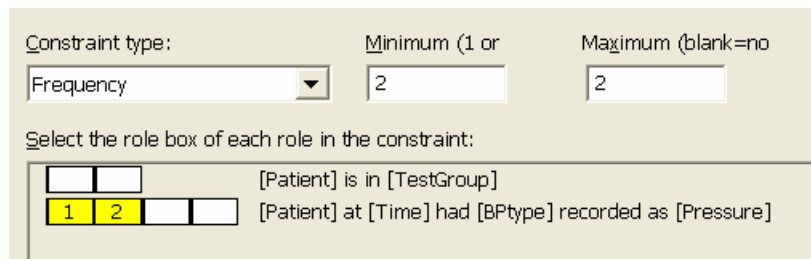

Figure 8 Adding the constraint that each (Patient, Time) pair in the quaternary fact table is there twice

If you build a project out of just the ORM model shown in Figure 6, you get a two table schema as shown in Figure 7. There are no foreign key constraints because the ORM model did not include any mandatory roles. The value constraint on bpType, and the two frequency constraints discussed earlier exist in the model but are not displayed on the diagram
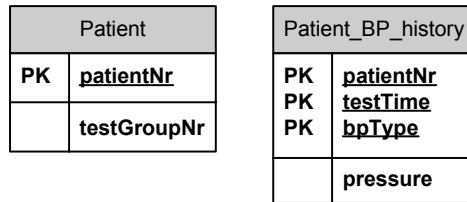
Patient	
PK	patientNr
	testGroupNr

Patient_BP_history	
PK PK PK	patientNr testTime bpType
	pressure

Figure 9   The relational model mapped from Figure 6

If you generate the DDL for this schema, the frequency constraints will appear, coded as stored procedures. You can decide whether to run these procedures or translate them into triggers. For those target DBMSs that do not support stored procedures (e.g. Microsoft Access), the stored procedures will be documented as comments, which you can use to help develop an alternative way to enforce the constraints.

Conclusion

This article discussed how to map functional details of ORM subtypes to separate tables, and how to specify frequency constraints on an ORM model. The next article will complete our discussion of ORM constraints by covering ring constraints, index constraints and constraint layers. 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).
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*.

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.