Microsoft’s new database modeling tool: Part 1

Terry Halpin
Microsoft Corporation

Abstract: This is the first in a series of articles introducing the Visio-based database modeling component of Microsoft Visual Studio Enterprise Architect. The initial emphasis is on the Object-Role Modeling (ORM) support provided by this tool.

Introduction

The database modeling solution within Microsoft Visio Enterprise 2000 provided basic support for conceptual information analysis using Object-Role Modeling (ORM), as well as logical database modeling using relational, IDEF1X, crowsfoot and object-relational notations. ORM schemas could be forward engineered to logical database schemas, from which physical database schemas could be generated for a variety of database management systems (DBMSs). Physical databases could have their structures reverse engineered to logical database schemas or to ORM schemas. The recently released Microsoft Visio 2002 products include Standard and Professional editions only. The Professional edition subsumes the formerly separate Technical edition, but does not include Enterprise. Although Visio 2002 Professional does include an ORM stencil, this is for drawing only—its ORM diagrams cannot be mapped to logical database schemas and cannot be obtained by reverse engineering from a physical database. Visio 2002 Professional includes a database modeling solution for defining new logical database schemas or reverse engineering them from existing databases, but forward engineering to physical database schemas is not provided.

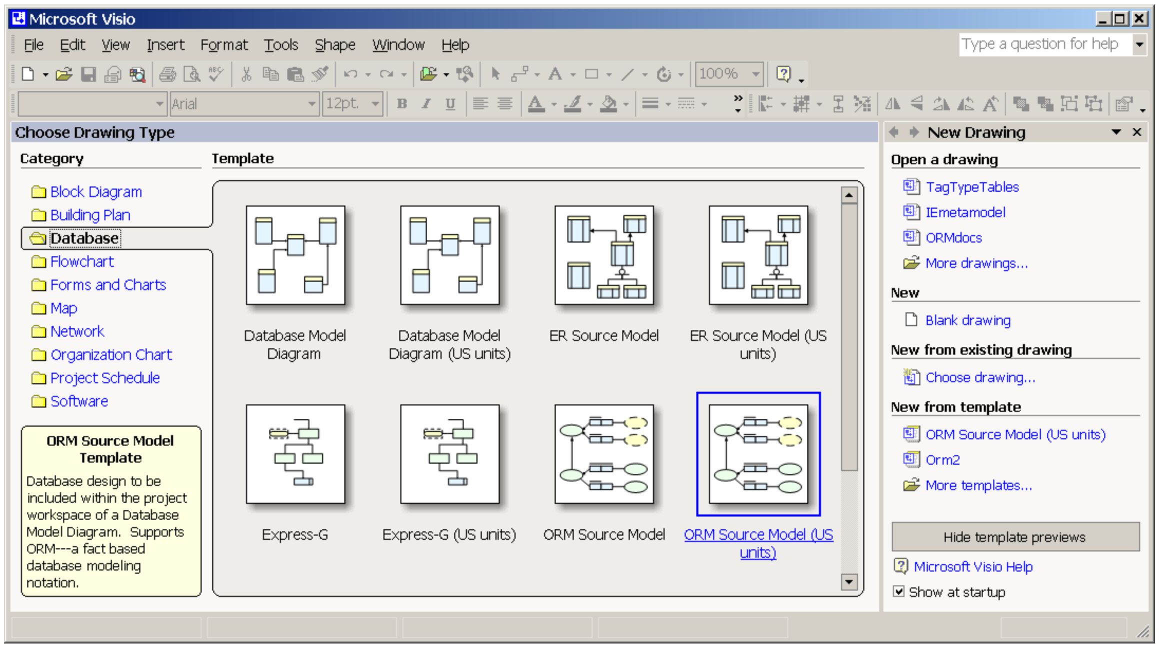
For some time, Microsoft has supported database design and program code design (using UML) in its Visual Studio product range. After acquiring Visio Corporation, Microsoft had two separate products (Visio Enterprise and Visual Studio) that overlapped significantly in functionality, with each supporting database design and UML. As a first step towards unifying these product offerings, the deep modeling solutions formerly within Visio Enterprise have been enhanced and moved into a new Microsoft product known as Visio for Enterprise Architects (VEA), included in Visual Studio .NET Enterprise Architect. These Visio-based modeling solutions are included in beta 2 of Visual Studio. NET Enterprise, with the final release to follow later. The deep ORM solution in VEA is completely different from the simple ORM drawing stencil in Visio Professional, and cannot translate between them. However, the database modeling solution in VEA can import logical database models from Visio Professional, and then forward engineer them to DDL scripts or physical database schemas.

This series of articles provides a simple introduction to using the database modeling solution within VEA. Various aspects of these articles (e.g. company names, product names, user interfaces) are trade-marked, copy-righted or patented by Microsoft Corporation. In this article, the emphasis is on the basics of the ORM solution. Familiarity with ORM and relational database modeling is assumed. Overviews of ORM are online [1, 2]. A deep treatment of ORM and database modeling is discussed in my latest book [3].

Creating a new ORM model

The Visio-based modeling tool runs as a separate solution within Visual Studio .NET Enterprise Architect. When you open the tool, the opening screen of the beta looks like Figure 1. Choose Database as the Drawing Type, then select the relevant ORM template. If in the United States, you normally choose ORM Source Model (US units), as shown—this defaults the page size to Letter, and measurements to inches. As you hover the cursor over a template icon, the icon is highlighted and a tool tip appears on the left. Visio provides both US and International (metric) versions of templates. If you select ORM Source Model (without “(US units)”), the page size defaults to A4 and the units to metric. [*Note*: in the final release, templates for just your country’s standard unit system are installed unless you choose otherwise.]

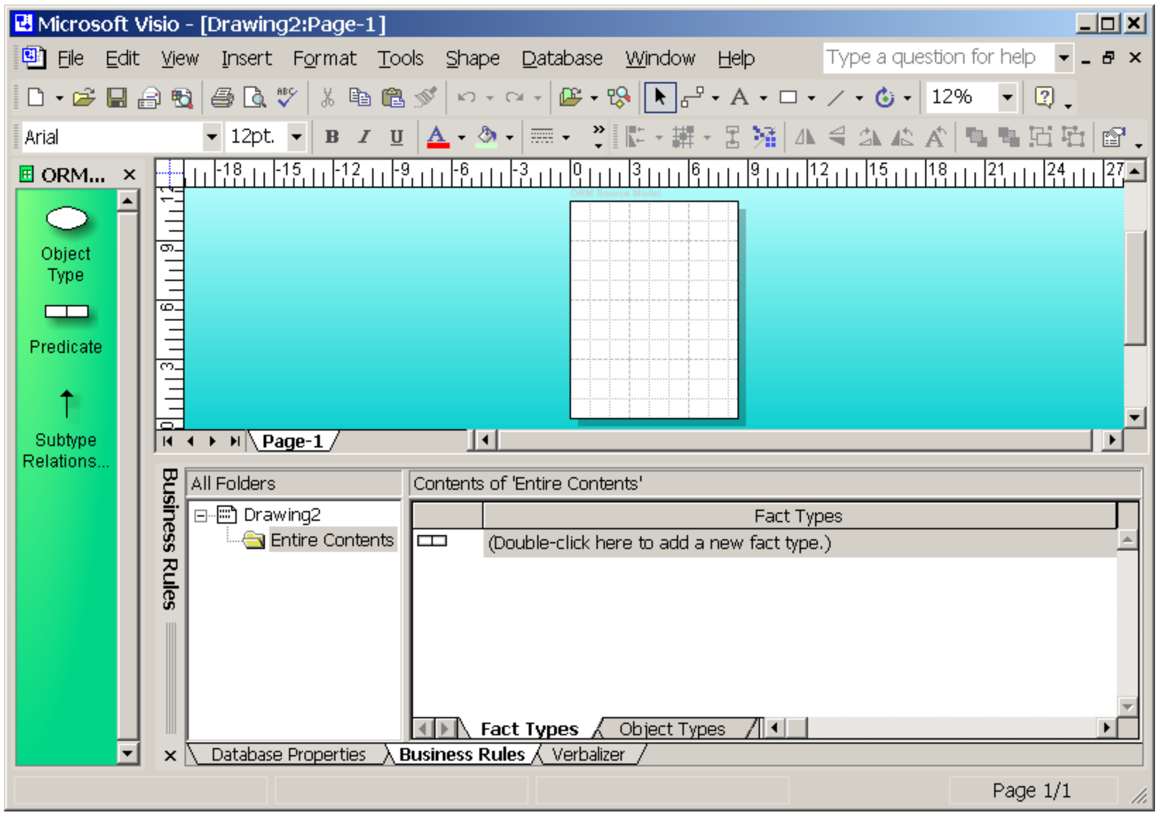
When you select an ORM Source Model template, a screen like Figure 2 appears. In addition to the menus and icons at the top, there is an ORM stencil, a Drawing window, and an area for displaying the Business Rules editor, Database Properties sheet and other windows you might open (e.g. Verbalizer).



ORM Source Model Template

Database design to be included within the project workspace of a Database Model Diagram. Supports ORM---a fact based database modeling notation.

Figure 1 Choosing to work with an ORM Source model



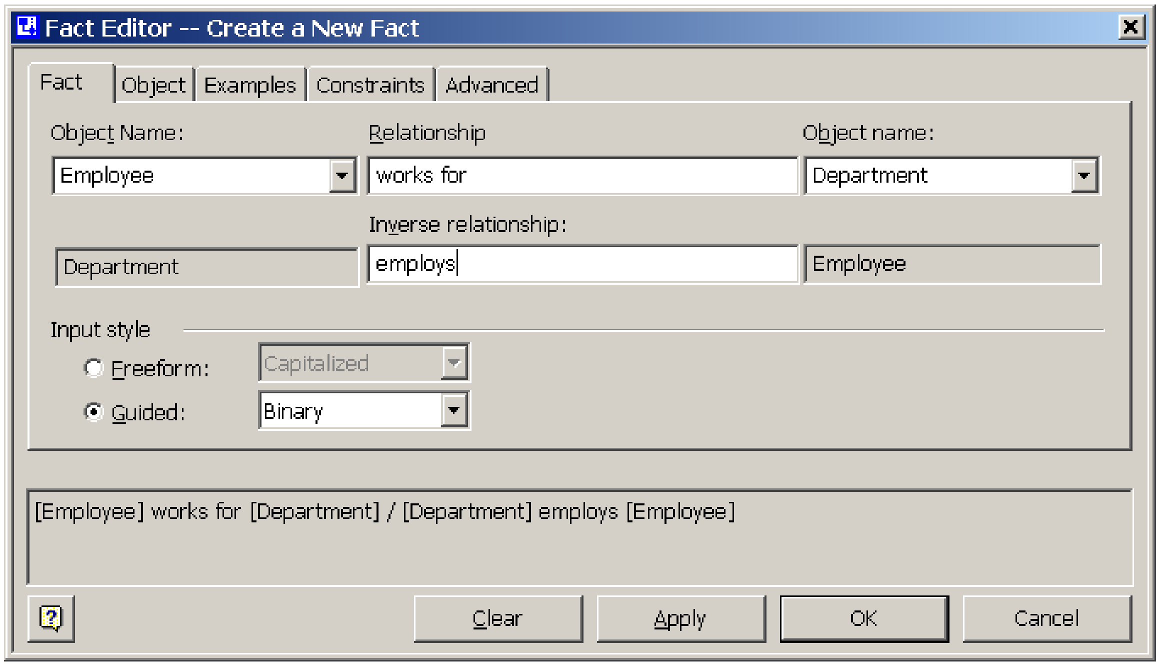
(Double-click here to add a new fact type.)

Figure 2 ORM stencil, Drawing window and Business Rules window

To reduce the space consumed by Figure 2, I've resized the display quite a bit. Normally the drawing window will occupy most of the screen. By default, the three shapes in the ORM stencil appear on the same horizontal row. You can give the drawing window more room by reducing the width of the ORM stencil so that the shapes line up vertically, as shown here. To do this, hover the cursor on the border between the stencil and the drawing window until the cursor changes to a resize cursor, then drag it to the left.

Adding sentence types using the Fact Editor

You can add sentence types (fact types or reference types) to an ORM model by dragging Object Type and Predicate shapes from the stencil to the drawing window. Alternatively you can add sentence types using the Fact Editor. For now, let's use the Business Rules editor to do this. Move the cursor to the bottom row displayed in the Fact Types pane of the Business Rules window (in our example, there is only one row). Now either start typing the fact type or hit the F2 key. This causes the Fact Editor to appear. You can also invoke the Fact Editor by choosing Database > View > Fact Editor from the database menu at the top of the screen. By default, the fact editor's input style is Guided, as shown in Figure 3.


Fact Editor -- Create a New Fact

Object Name: Employee
Relationship: works for
Object name: Department
Inverse relationship: employs
Department
Employee

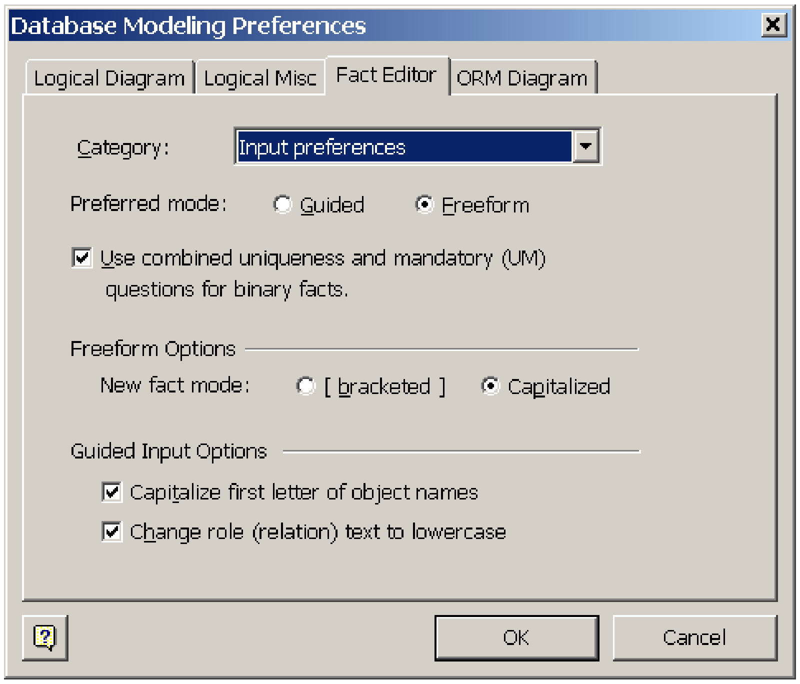
Input style
Freeform: Capitalized
Guided: Binary

[Employee] works for [Department] / [Department] employs [Employee]

Figure 3 Fact Editor using Guided input style

You can enter a binary relationship giving both a forward reading (e.g. Employee works for Department) and an inverse reading (e.g. Department employs Employee). If needed, the arity (number of roles) in the relationship can be changed by choosing a different setting from "Binary". The Object pane allows you to classify the object type as an entity type, value type or external object type. In the case of an entity type with a simple identification scheme, you can add its reference mode (e.g. empNr for Employee and code for Department).

Once you are familiar with the Fact Editor, you will probably want to change its input style to Freeform. This allows you to enter sentence types much faster by using a formal syntax. You can use the radio button to change to Freeform. You can also make Freeform the default by going to the Database menu at the top of the screen, choosing Database > Options > Modeling ..., then opening the Fact Editor pane and setting the preferred mode to Freeform, as shown in Figure 4. In many languages it is convenient to identify object types by starting their name with a capital letter, assuming the name is a single word (e.g. Employee, VicePresident). For languages where this doesn't work, or when the name uses multiple words separated by spaces, bracketed mode should be chosen: this encloses the object type name in square brackets (e.g. [employee], [vice president]).


Database Modeling Preferences

Logical Diagram | Logical Misc | Fact Editor | ORM Diagram

Category: Input preferences

Preferred mode: Guided [✓ Freeform]

[✓ Use combined uniqueness and mandatory (UM) questions for binary facts.]

Freeform Options

New fact mode: [bracketed] [✓ Capitalized]

Guided Input Options

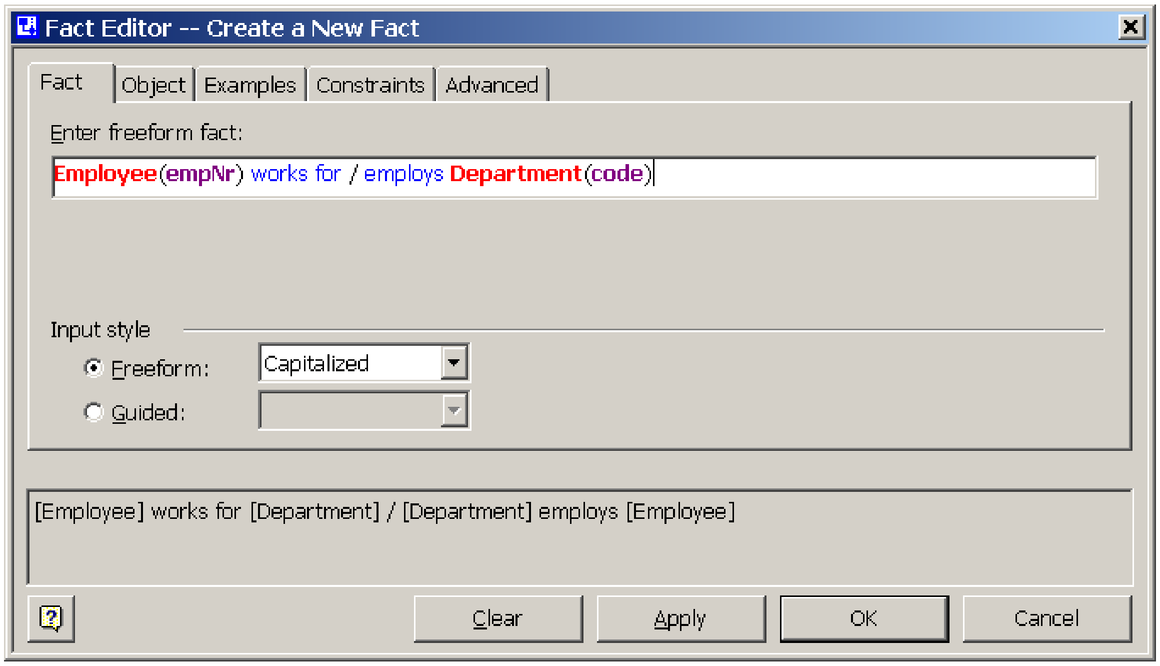
[✓ Capitalize first letter of object names]

[✓ Change role (relation) text to lowercase]

OK | Cancel

Figure 4 Setting the default input style for the Fact Editor to Freeform

In Freeform mode, reference modes appear in parentheses after the object type name. If an inverse reading is supplied, a slash "/" is used to separate forward and inverse readings.
Figure 5 shows an example.


Fact Editor -- Create a New Fact

Fact | Object | Examples | Constraints | Advanced

Enter freeform fact:

Employee(empNr) works for / employs Department(code)

Input style

[✓ Freeform:] Capitalized

Guided:

[Employee] works for [Department] / [Department] employs [Employee]

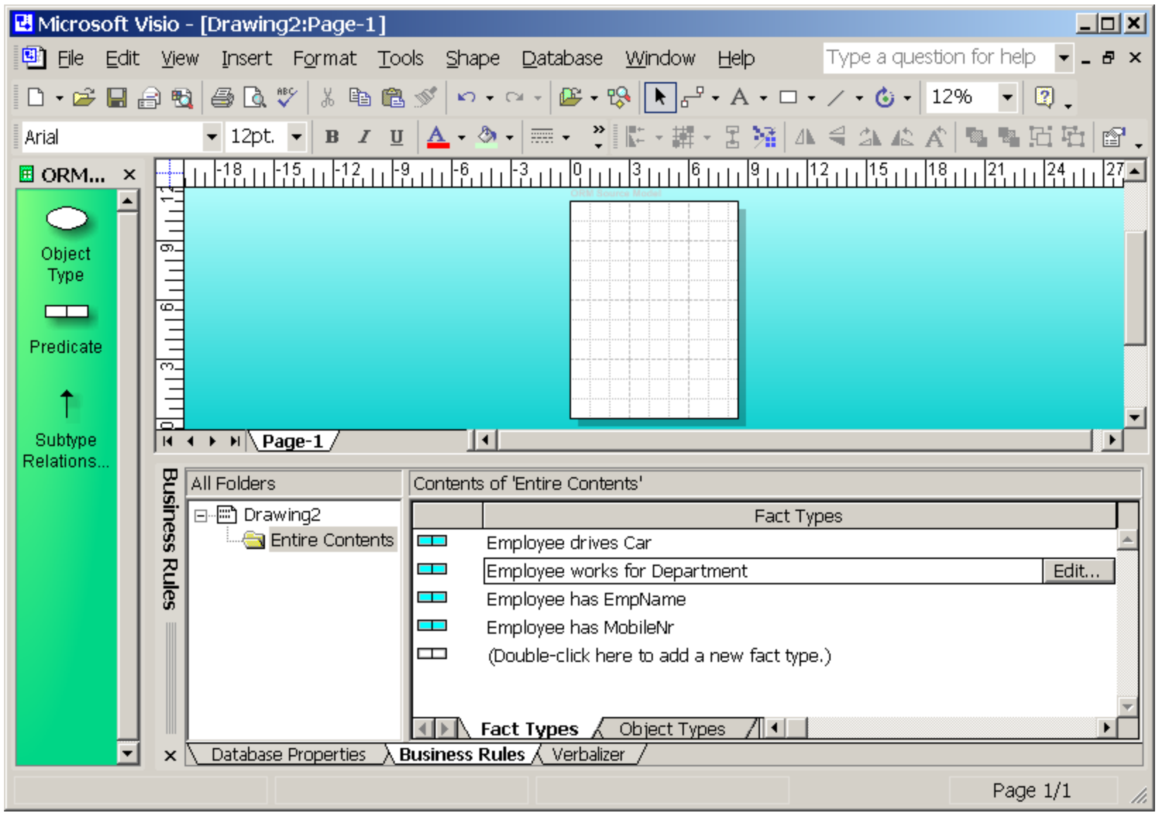
Clear | Apply | OK | Cancel

Figure 5 Fact Editor using Freeform input style

Once a reference scheme has been supplied for an entity type, there is no need to repeat the reference scheme in specifying later fact types. Unlike entity types, value types (e.g. EmployeeName, RoomNr) have no reference scheme, since their instances are just literal constants (e.g. character strings or numbers used to name or refer to entities) so they identify themselves. In Freeform mode, value types are indicated by appending empty parentheses "()". Examples of some fact types are given below using the formal, Freeform syntax:

```
Employee(empNr) works for / employs Department(code)
Employee has EmployeeName()
Employee has MobileNr()
Employee drives / is driven by Car(regNr)
```

Use the fact editor now to enter these fact types (using either Guided or Freeform input). Hit the Apply button after each of the first three fact types to have the fact type added. When you have entered the fourth fact type, hit OK: this adds the last fact type and then closes the fact Editor. The fact types do not yet appear on the drawing window, but should now be listed in the Business Rules editor. If you move the cursor to one of these fact types, an Edit button appears to its right (see Figure 6). If you click the Edit button, it brings up the Fact Editor with that fact type displayed for editing. This provides one way to add basic constraints and examples to fact types.



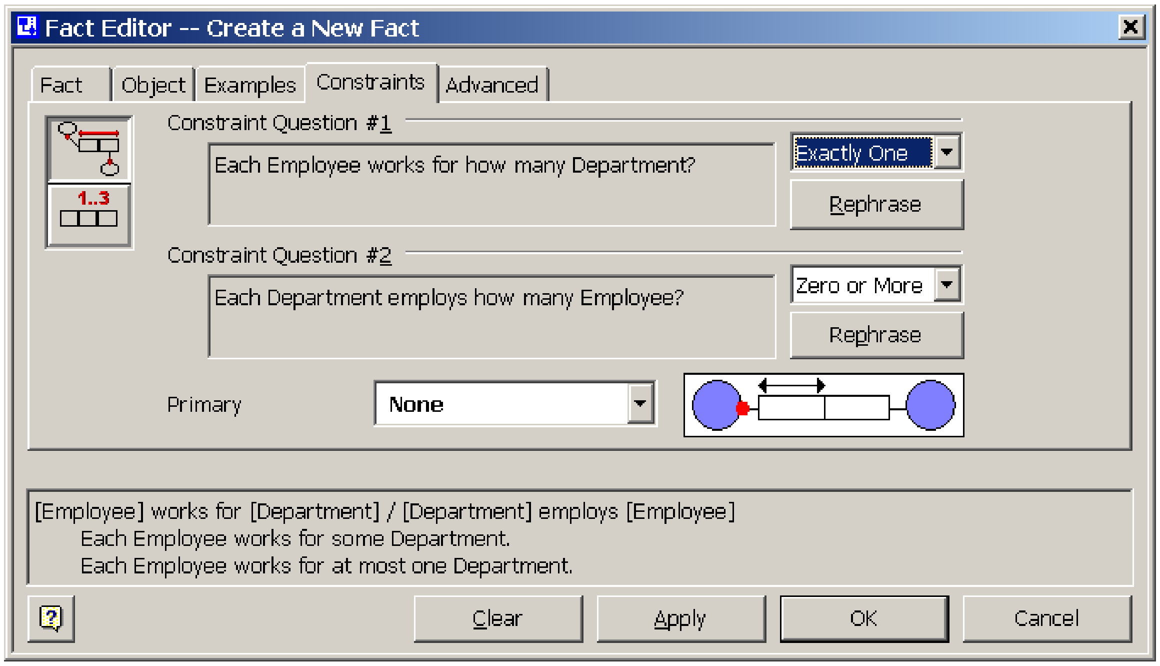
Employee drives Car
Employee works for Department
Employee has EmpName
Employee has MobileNr
(Double-click here to add a new fact type.)

Figure 6 Fact types are listed in the Business Rules editor, and may be edited

Adding basic, internal constraints using the Fact Editor

A constraint is internal if it applies to just one predicate; otherwise it is external. The fact editor allows you declare the following internal constraints: internal uniqueness, simple mandatory, internal frequency and ring constraints. It does not allow you to specify internal, set-comparison constraints (e.g. an exclusion constraint between two roles of the same predicate), external constraints (e.g. an external uniqueness constraint, or a set-comparison constraint between two predicates) or value constraints (e.g. restricting Sexcode values to {'M', 'F'}). In practice, constraints declared in the fact editor are best restricted to simple internal uniqueness and simple mandatory constraints. To declare any other kind of constraint, there is a much faster method (see Part 2 of this series of articles).

To add a constraint on a fact type displayed in the Fact editor, choose the Constraints tab. By default, the constraints pane combines uniqueness and mandatory constraints to make it faster to specify them. For instance, in Figure 7 choosing "exactly one" means both "at least one" (mandatory) and "at most one" (unique). The constraint symbols and verbalization automatically appear to help you see the result of your choice. If you don't want to use this default shortcut, open the Database Modeling Preferences dialog (Figure 4) and uncheck the option that indicates combined uniqueness and mandatory (UM).



Constraint Question #1: Each Employee works for how many Department? — Exactly One

Constraint Question #2: Each Department employs how many Employee? — Zero or More

Primary: None

[Employee] works for [Department] / [Department] employs [Employee]
Each Employee works for some Department.
Each Employee works for at most one Department.

Figure 7 Adding constraints in the Fact Editor

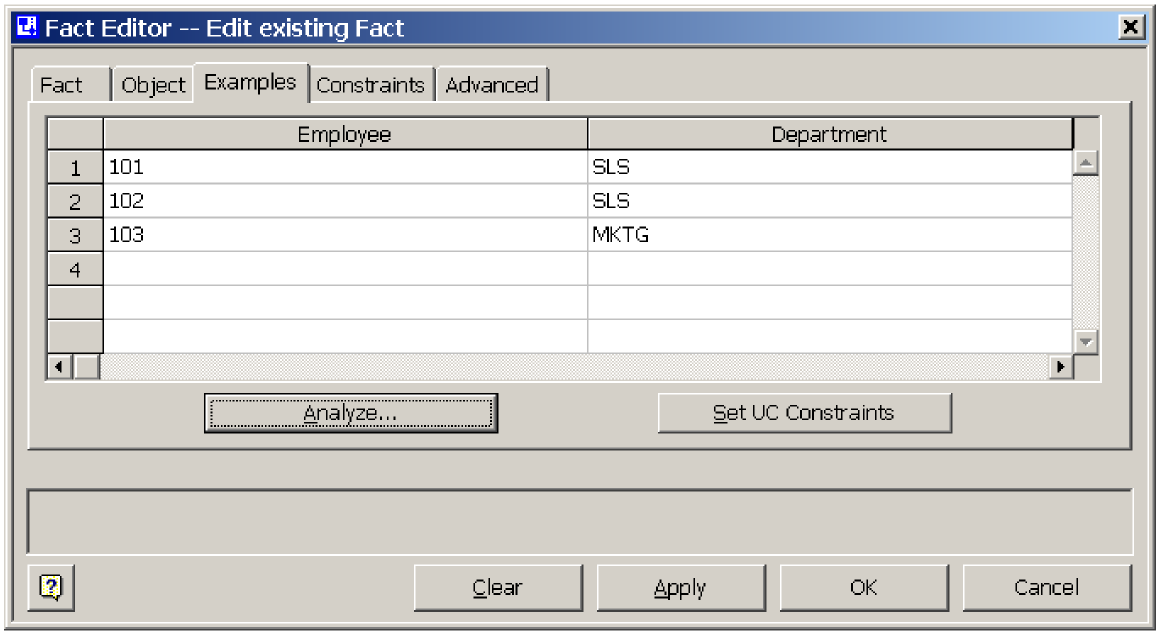
For practice, use the Fact editor to add each of the following constraints. In the current version of the tool, "some" is used instead of "the same" in the final constraint, which indicates that the drives relationship is optional and many-to-many.

Each Employee works for some Department
Each Employee works for at most one Department
Each Employee has some EmployeeName
Each Employee has at most one EmployeeName
Each Employee has at most one MobileNr
It is possible that the same Employee drives more than one Car
    and that the same Car is driven by more than one Employee

Adding examples to fact types

It is a good idea to include examples for all fact types. To add a constraint on a fact type displayed in the Fact editor, click the Examples tab, and enter enough examples to illustrate the relevant constraints. For example, Figure 8 shows three fact instances for our Employee works for Department fact type. Here employees 101 and 102 works for the sales department(SLS) and employee 103 works for the marketing department

(MKTG). The population is consistent with our situation where each employee works for at most one department (first column values are unique) but the same department may employ more than one employee (SLS is duplicated in column 2).



	Employee	Department
1	101	SLS
2	102	SLS
3	103	MKTG
4		

Figure 8 Adding example fact instances for Employee works for Department

You can use the Analyze button to request the tool to induce the constraints from your examples, or to check for inconsistencies between your data and your constraint specification. Try it out for yourself. This is a very useful feature for validating constraints.

Saving a model

To save your model, choose File > Save from the File menu, or click the Save (diskette) icon. This opens the SaveAs dialog box. Choose the folder where you want to save the model, add a filename for the model, hit the Save button in the dialog, then hit OK in the properties dialog. The file will be saved with the extension ".vsd" (Visio document).

Displaying sentence types on the drawing

To display on the diagram any sentence types entered using the Fact Editor, first locate the fact types of interest in the Business Rules editor. To select a contiguous series of fact types, hold the Shift key down as you select the first and last fact type in the series. All but the first fact type will appear highlighted. You can then drag the fact types on to the drawing page where you want them displayed.

Try this now for the four fact types in our model. The diagram appears by default as shown in Figure 9. You can finesse the display by moving the predicate text and object types around.

A handy alternative is to open the Object Types pane of the Business Rules window, drag out one or more the relevant object types, and use the Show Relationships option. For example, if you drag the Employee object type onto any drawing page, right-click Employee, and select Show Relationships from its right-click menu, all the relationships in which Employee plays will be displayed on that page. This ShowRelationships feature is an extremely useful feature in schema browsing and in reverse engineering. It is one of many new features that were not provided previously in VisioModeler or Visio Enterprise.

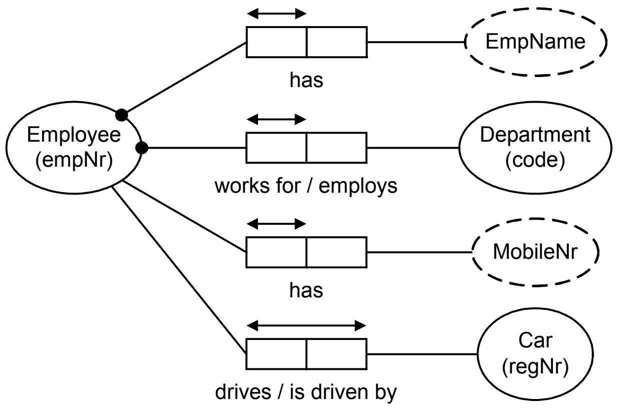


Figure 9 Diagram formed by dragging the four fact types from the Business Rules editor

Mapping an ORM model to a logical database model

To map an ORM model to a logical database model, you first add the ORM model to a database model project, then build it. From the File menu, open the logical database modeling solution by choosing File > New > Database > Database Model Diagram (US units). If you want the metric template, choose just Database Model Diagram without the "(US units)". The screen should now look like Figure 10, except that I have reduced the size of the drawing window significantly. The "Entity Relationship" stencil can be used to create logical database models from scratch, but for now we will derive the database model from our ORM model.

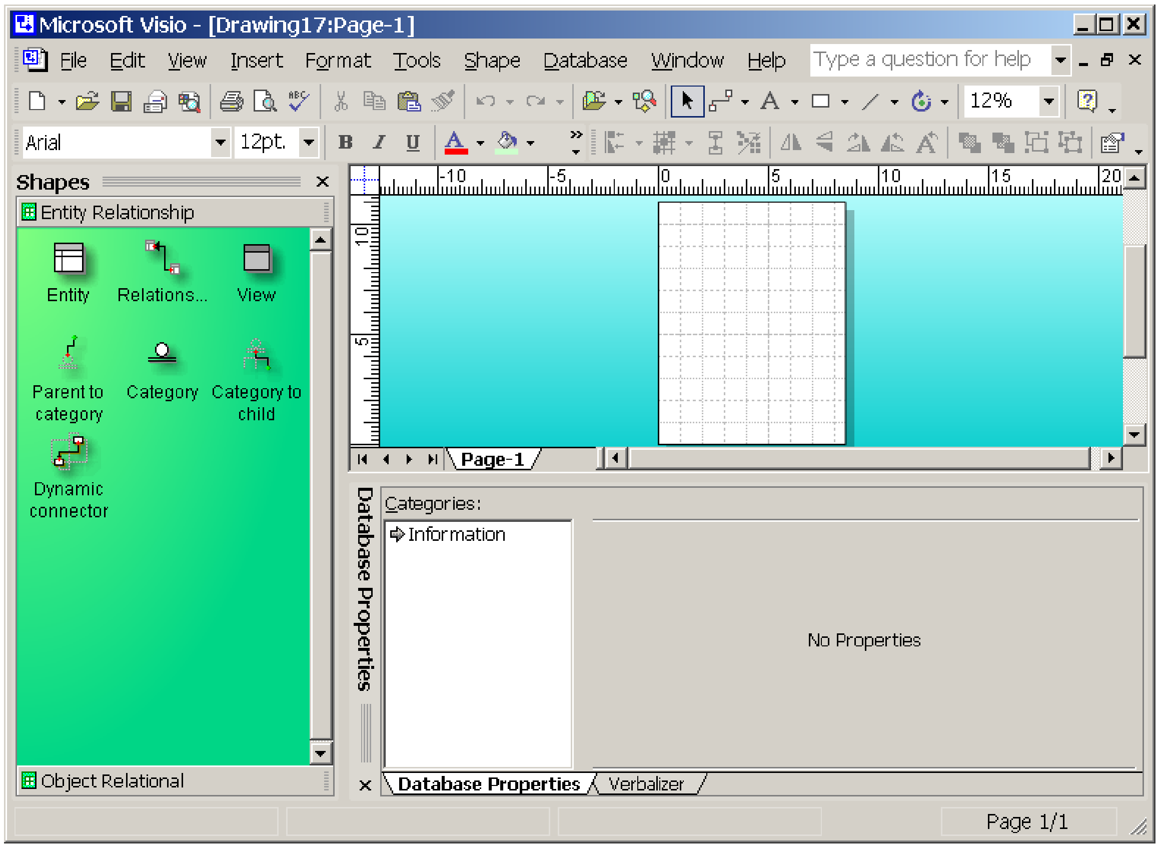


Figure 10 The logical database modeling solution

To create a database model project, choose Database > Project > Add existing document from the Database menu. An Add Document to Project dialog box should now appear. Use the "Look in:" field to navigate to your saved ORM model, then hit the Open button. The ORM model (mine was called JCM1.vsd) should then be listed in the project window. Now save the project file by hitting the Save icon on the main menu and giving it a filename (I chose ProjJCM1). The project file will also have the extension ".vsd". The name and page of your current model is always listed in the title bar at the top of the screen. Figure 11 shows what the screen should look like at this stage.

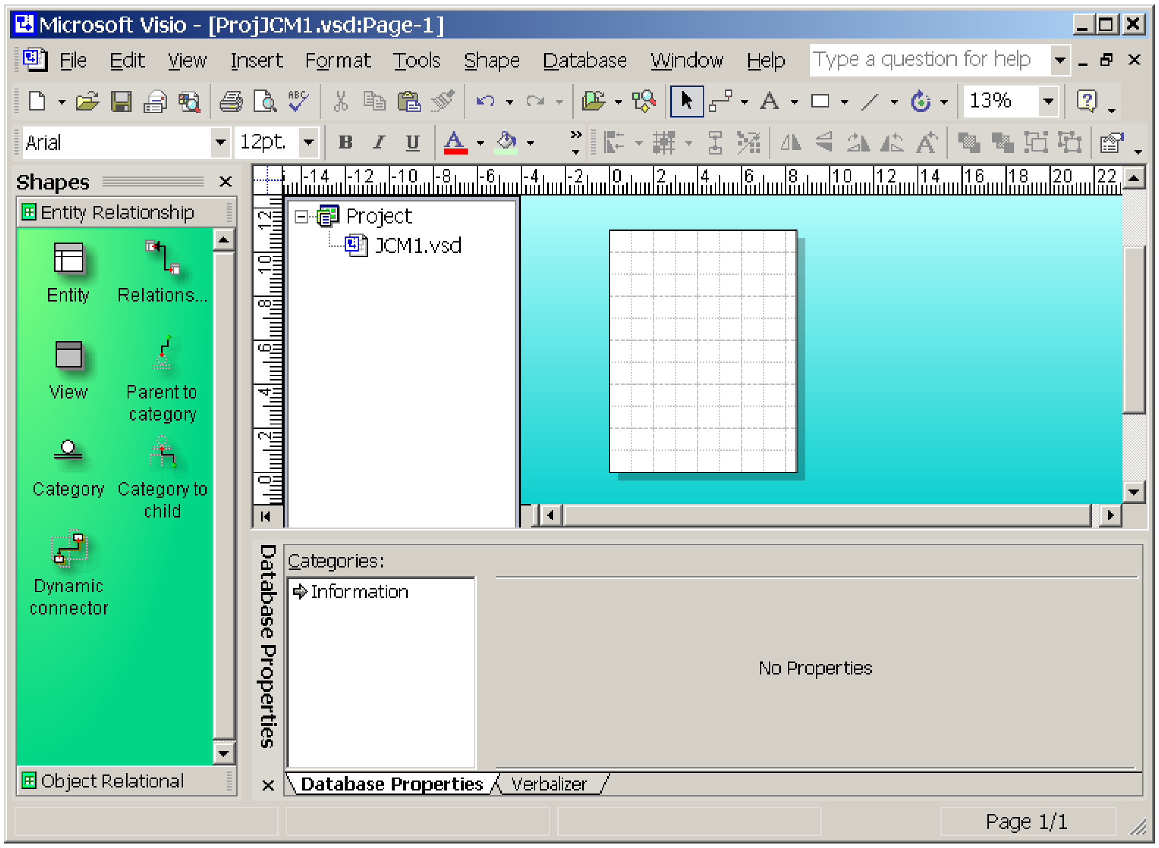
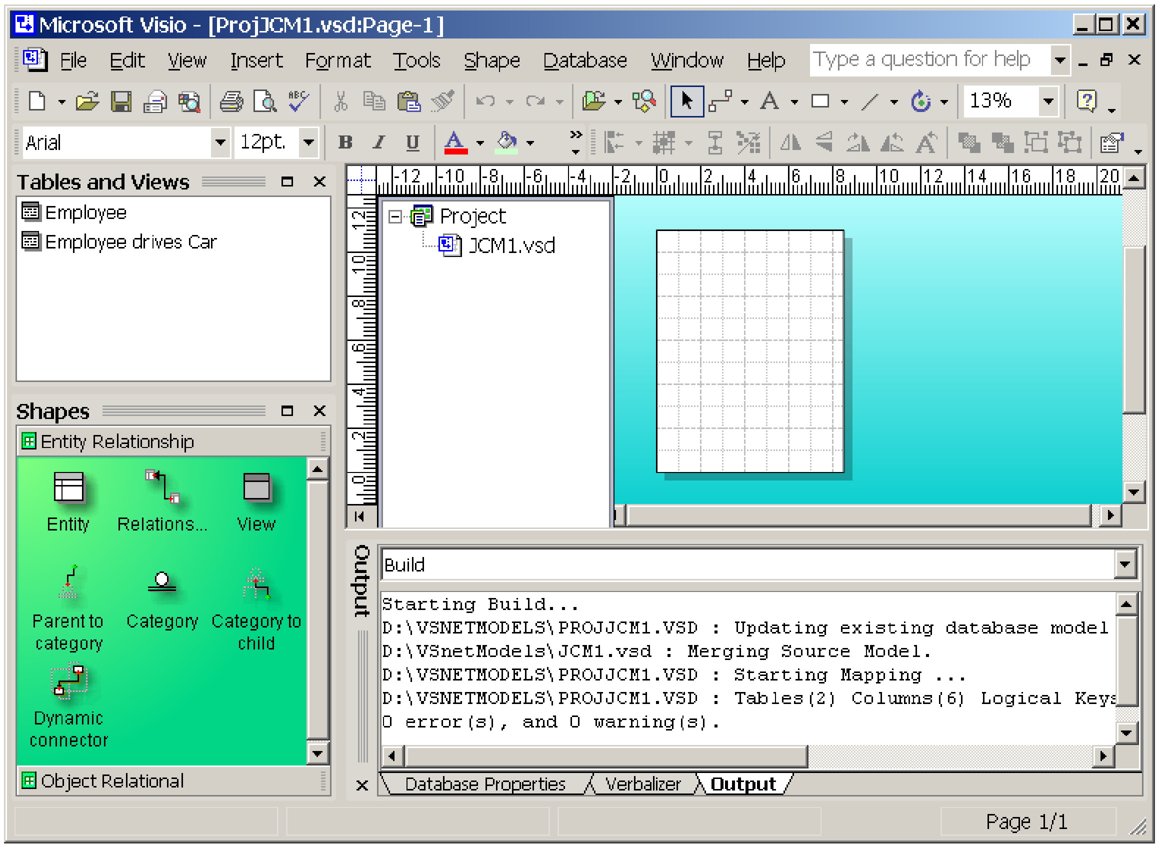


Figure 11 A database model project that includes an ORM source model

Now build the logical model by choosing Database > Project > Build from the Database menu. The relational schema is now automatically built, and the resulting table schemes appear in the Tables and Views window at the left of the screen (see Figure 12).

To see these table schemes on a diagram, drag them onto the drawing page. The result is shown in Figure 13. There are two table schemes, with one foreign key connection between them. Each table has its name in the shaded header, with its columns listed below. Primary keys are underlined, marked "PK" and appear in the top compartment for the columns. Mandatory (not null) columns are in bold. Foreign key columns are marked FK*n* where *n* is the number of the foreign key with a table. In this case we have only one foreign key that targets the primary key of the Employee table. The foreign key connection itself is depicted as an arrow from the foreign key to its target key.

In this example, the names of the tables and columns are those that are generated automatically by default. In practice we would normally rename many of these and also change many of the default data types that have been chosen. Various configuration options exist for controlling how table and column names are generated. In practice it's best to set the data types on the ORM model, where object types correspond to conceptual domains. The correct data types then automatically propagate to all the attributes based on these domains. Such issues are ignored in this first article.



```
Starting Build...
D:\VSNETMODELS\PROJJCM1.VSD : Updating existing database model
D:\VSnetModels\JCM1.vsd : Merging Source Model.
D:\VSNETMODELS\PROJJCM1.VSD : Starting Mapping ...
D:\VSNETMODELS\PROJJCM1.VSD : Tables(2) Columns(6) Logical Keys
0 error(s), and 0 warning(s).
```

Figure 12 Two table schemes are built by mapping the ORM model

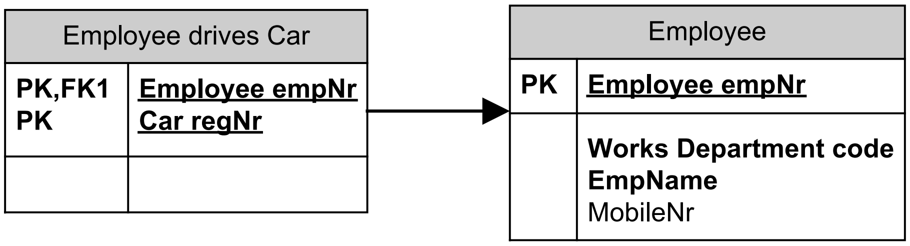


Figure 13 The relational schema mapped from the ORM model

Generating the physical database schema

You can generate the internal schema for a selected target DBMS by clicking the Database > Generate from the Database menu. Generation gives you the option of generate the DDL script or than having the tool build the tables for you. It is usually best to first generate the DDL script, which you can later execute within your chosen DBMS. Follow the steps in the generate wizard, choose your driver (e.g. Microsoft SQL Server 2000), enter a database name (e.g. mydb), accept defaults for the next screen, and choose Yes to view the generated DDL script, then save the DDL script as a text file.

Conclusion

This article has only touched on the basics of creating a very simple ORM model and then mapping it to a logical and then physical database schema. Later articles will examine how to specify much more powerful ORM models involving advanced constraints and nesting, and also provide more details about the logical database modeling facilities. You can gain some idea of the more advanced features by playing with the Employee ORM Source Model that is included in the sample files issued with the product. If you have any constructive feedback on this article, please e-mail me at: TerryHa@microsoft.com.

References

1. Halpin, T. A. 1998a, 'Object Role Modeling: an overview', white paper, (online at www.orm.net).
2. Halpin, T.A. 1998b, 'Object-Role Modeling (ORM/NIAM)', *Handbook on Architectures of Information Systems*, Bernus, P., Mertins, K. & Schmidt (eds), Springer, Heidelberg, Ch. 4. (online at www.orm.net).'
3. Halpin, T.A. 2001, *Information Modeling and relational Databases*, Morgan Kaufmann Publishers, San Francisco (www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672-6).