



## Database Modelling in UML - Part 1

By Geoffrey Sparks, [www.sparxsystems.com.au](http://www.sparxsystems.com.au)

### Introduction

When it comes to providing reliable, flexible and efficient object persistence for software systems, today's designers and architects are faced with many choices. From the technological perspective, the choice is usually between pure Object-Oriented, Object-Relational hybrids, pure Relational and custom solutions based on open or proprietary file formats (eg. XML, OLE structured storage). From the vendor aspect Oracle, IBM, Microsoft, POET and others offer similar but often-incompatible solutions.

This article is about only one of those choices, that is the layering of an object-oriented class model on top of a purely relational database. This is not to imply this is the only, best or simplest solution, but pragmatically it is one of the most common, and one that has the potential for the most misuse.

We will begin with a quick tour of the two design domains we are trying to bridge: firstly the object-oriented class model as represented in the UML, and secondly the relational database model.

For each domain we look only at the main features that will affect our task. We will then look at the techniques and issues involved in mapping from the class model to the database model, including object persistence, object behaviour, relationships between objects and object identity. We will conclude with a review of the UML Data Profile (as proposed by Rational Software).

Some familiarity with object-oriented design, UML and relational database modelling is assumed.

### The Class Model

The Class Model in the UML is the main artefact produced to represent the logical structure of a software system. It captures the both the data requirements and the behaviour of objects within the model domain. The techniques for discovering and elaborating that model are outside the scope of this article, so we will assume the existence of a well designed class model that requires mapping onto a relational database.



The class is the basic logical entity in the UML. It defines both the data and the behaviour of a structural unit. A class is a template or model from which instances or objects are created at run time. When we develop a logical model such as a structural hierarchy in UML we explicitly deal with classes.

When we work with dynamic diagrams, such as sequence diagrams and collaborations, we work with objects or instances of classes and their inter-actions at run-time.

The principal of data hiding or encapsulation is based on localisation of effect. A class has internal data elements that it is responsible for. Access to these data elements should be through the class's exposed behaviour or interface. Adherence to this principal results in more maintainable code.

### **Behaviour**

Behaviour is captured in the class model using the operations that are defined for the class. Operations may be externally visible (public), visible to children (protected) or hidden (private). By combining hidden data with a publicly accessible interface and hidden or protected data manipulation, a class designer can create highly maintainable structural units that support rather than hinder change.

### **Relationships and Identity**

Association is a relationship between 2 classes indicating that at least one side of the relationship knows about and somehow uses or manipulates the other side. This relationship may be functional (do something for me) or structural (be something for me). For this article it is the structural relationship that is most interesting: for example an Address class may be associated with a Person class. The mapping of this relationship into the relational data space requires some care.

Aggregation is a form of association that implies the collection of one class of objects within another. Composition is a stronger form of aggregation that implies one object is actually composed of others. Like the association relationship, this implies a complex class attribute that requires careful consideration in the process of mapping to the relational domain.

While a class represents the template or model from which many object instances may be created, an object at run time requires some means of identifying itself such that associated objects may act upon the correct object instance. In a programming language like C++, object pointers may be passed around and held to allow objects access to a unique object instance.



Often though, an object will be destroyed and require that it be re-created as it was during its last active instance. These objects require a storage mechanism to save their internal state and associations into and to retrieve that state as required.

Inheritance provides the class model with a means of factoring out common behaviour into generalised classes that then act as the ancestors of many variations on a common theme. Inheritance is a means of managing both re-use and complexity. As we will see, the relational model has no direct counterpart of inheritance, which creates a dilemma for the data modeller mapping an object model onto a relational framework.

Navigation from one object at run time to another is based on absolute references.

One object has some form of link (a pointer or unique object ID) with which to locate or re-create the required object.

### **The Relational Model**

The relational data model has been around for many years and has a proven track record of providing performance and flexibility. It is essentially set based and has as its fundamental unit the 'table', which is composed of a set of one or more 'columns', each of which contains a data element.

### **Tables and Columns**

A relational table is collection of one or more columns each of which has a unique name within the table construct. Each column is defined to be of a certain basic data type, such as a number, text or binary data. A table definition is a template from which table rows are created, each row being an instance of a possible table instance.

### **Public Data Access**

The relational model only offers a public data access model. All data is equally exposed and open to any process to update, query or manipulate it. Information hiding is unknown.

### **Behaviour**

The behaviour associated with a table is usually based on the business or logical rules applied to that entity.

Constraints may be applied to columns in the form of uniqueness requirements, relational integrity constraints to other tables/rows, allowable values and data types.

Triggers provide some additional behaviour that can be associated with an entity.

Typically this is used to enforce data integrity before or after updates, inserts and deletes.



Database stored procedures provide a means of extending database functionality through proprietary language extensions used to construct functional units (scripts). These functional procedures do not map directly to entities, nor have a logical relationship to them.

Navigation through relational data sets is based on row traversal and table joins. SQL is the primary language used to select rows and locate instances from a table set.

### **Relationships and Identity**

The primary key of a table provides the unique identifying value for a particular row. There are two kinds of primary key that we are interested in: firstly the meaningful key, made up of data columns which have a meaning within the business domain, and second the abstract unique identifier, such as a counter value, which have no business meaning but uniquely identify a row. We will discuss this and the implications of meaningful keys later.

A table may contain columns that map to the primary key of another table. This relationship between tables defines a foreign key and implies a structural relationship or association between the two tables.

### **Summary**

From the above overview we can see that the object model is based on discrete entities having both state (attributes/data) and behaviour, with access to the encapsulated data generally through the class public interface only. The relational model exposes all data equally, with limited support for associating behaviour with data elements through triggers, indexes and constraints.

You navigate to distinct information in the object model by moving from object to object using unique object identifiers and established object relationships (similar to a network data model). In the relational model you find rows by joining and filtering result sets using SQL using generalised search criteria.

Identity in the object model is either a run-time reference or persistent unique ID (termed an OID). In the relational world, primary keys define the uniqueness of a data set in the overall data space.

In the object model we have a rich set of relationships: inheritance, aggregation, association, composition, dependency and others. In the relational model we can really only specify a relationship using foreign keys.



## Aho Engineering Group

Having looked at the two domains of interest and compared some of the important features of each, we will digress briefly to look at the notation proposed to represent relational data models in the UML.