Mapping Object to Data Models with the UML

Rational Software White Paper

Rational
the **e-development** company

Table of Contents

Software Development for Database Application

The development of a database application involves a close working relationship between the software developers and the database development team. The most successful projects are marked by a shared vision and clear communication of project details. Software developers deal with object oriented software development and use the logical class model to represent the main view at the application, while the Database team designs, models, builds and optimizes the database. The areas of interface and overlap between these two distinct responsibilities often represent the most challenging aspect of database application development. This white paper discusses how the UML and the UML Profile for Data Modeling can help resolve this challenge.

The Application Model

The application model uses the class view of the application. Classes tagged as persistent describe the logical data model.

The application model describes the application layer dealing with the database. It contains classes used as the layer interface to the application and persistent classes used as the interface to the database.

The Data Model

The data model describes the physical implementation of the database. This is the model of the internal database structure.

Persistent classes from the application model map to the data model.

The UML Profile for Data Modeling

The UML profile for Data Modeling contains modeling constructs necessary to accurately model a database. These constructs are used to specify detailed information about the database and database modeling. See the Rational White Paper, “The UML profile for Data Modeling” for more details.

Mapping and dependencies between Application and Data Models

The physical data model must map to the database. This is a simple one to one mapping, which is used to forward or reverse engineer the database structures.

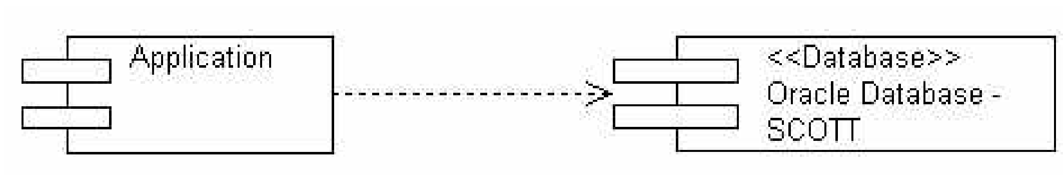
The mapping between the application model and the database model is more complex. As the data model can change because of normalization or de-normalization, the application model can change as well.

Therefore, this mapping must be able to describe any and every possible relationship between the application model and the data model.

Component to Database

The database itself is the physical layer of data storage. It has no mapping into the application model. Instead, the application has interfaces to the database.

The database is associated to the application. The type of the association is a dependency.

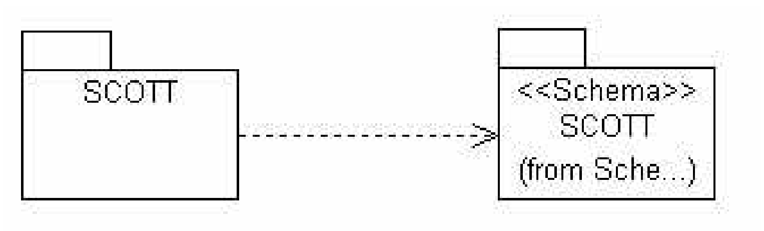


The dependency between database and component is part of a software design and must be modeled manually.

Package to Schema

The persistent view of the application is most commonly modeled in a persistent layer, represented by a package.

This package maps to a Schema. The mapping is used for forward and reverse engineering. The mapping, which can be used on a class diagram, is a dependency.



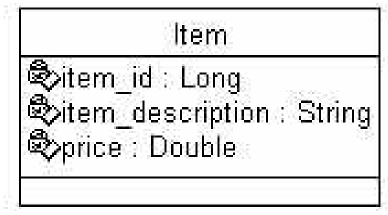
The dependency between package and schema is part of a software design and must be modeled manually.

Classes to Tables

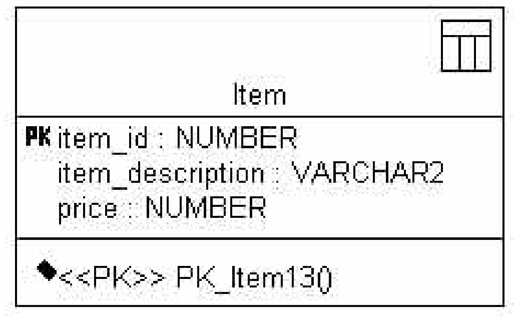
Persistent classes can be mapped to tables. The default mapping is a 1:1 mapping although classes using associations will be in some cases mapped to more than one table. See the “Association to Relationships” section below for details on the mapping of classes associated to another classes.

When a class is mapped to a table all of the necessary transformations will be done. See the “Attributes to Columns” section below.

The Item Class, below, is an example of a persistent table, which will be mapped to a database.



The corresponding table is the Item table.



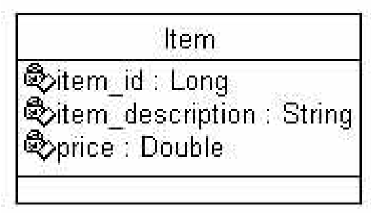
The mapping assigns the table to the class although, as will be seen later, the mapping is additionally provided on the detailed level of a column.

The mapping does not require the special design of a class or a table. The forward and reverse engineering itself provides the ability to generate primary keys and primary key constraints.

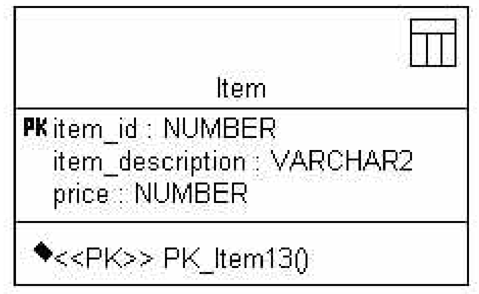
Attributes to Columns

Attributes of persistent classes map to columns of a table. Mapping of attributes must consider the datatype conversion between application datatype and database datatype. Although SQL-92 defines standard datatypes for the database, most vendors implement additional datatypes or change the name of standard datatypes.

The Item class also provides an example of the attribute to column conversion.



The data types used are Long for the id, String for the description and double for the price. In the case of an Oracle 8.x database this mapping is done as follows.



The Long is mapped to NUMBER(10)[1], the String to VARCHAR2, and the Double to NUMBER(20).

See the Rational Rose Data Modeler Online Help for a complete list of mapping for every database.

Associations to Relationships

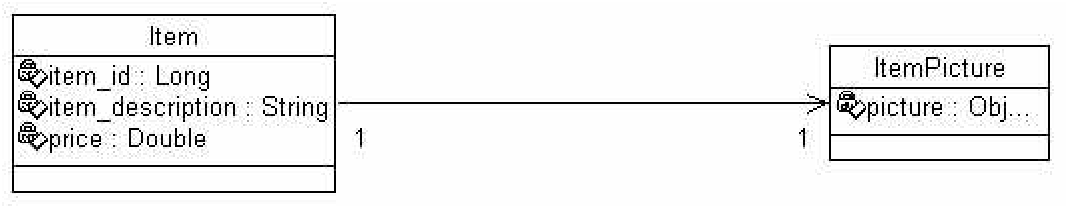
In a persistent data class model any of the provided association types can be used. To make it even more complex, all of the possible cardinalities of the class roles in the association must be supported.

It is difficult to use relationships in a data model, because the relational data model understands only identifying and non-identifying relationships and the1:N cardinality. The 1:1 cardinality must be forced through constraints.

Following are some examples of the mapping between the persistent class model and the relational data model.

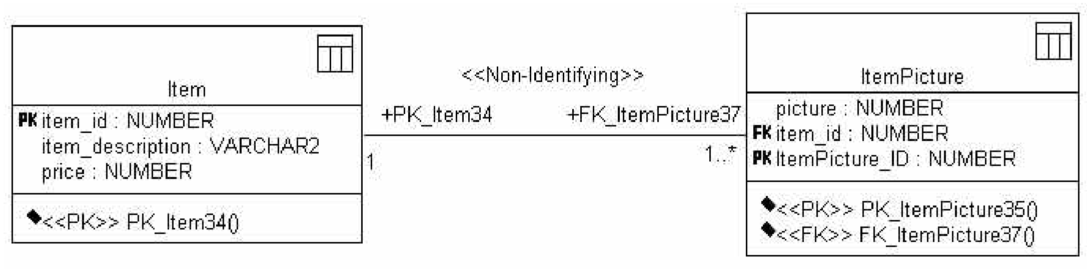
1:1 association maps to a non-identifying relationship

In the object-oriented design often a 1:1 association represents the relationship between two independent objects (in the example, Item and Picture). Each Item has to have a Picture, whereas the Picture has to be assigned to the Item. The association is unidirectional.



The mapping to a data model uses two tables and a non-identifying relationship.

[1] The length is specified in the detail specification of the column.



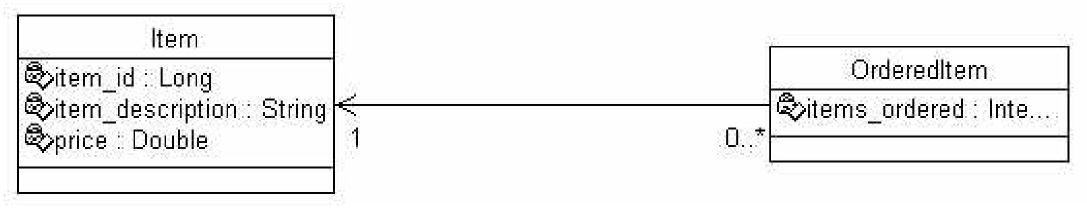
The foreign key of the table ItemPicture (item_id) uses the primary key of the Item table to build the relationship. The foreign key constraint must be generated.

Because of the basic functionality of the relational data model the relationship is always bi-directional.

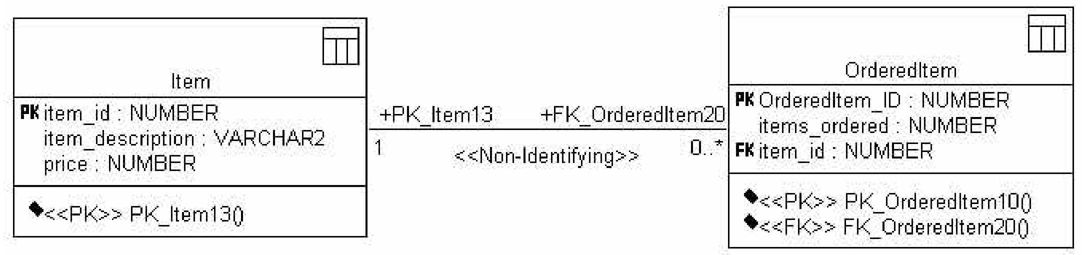
1:N association maps to a non-identifying relationship

The 1:N association is used as the association between the Item and the OrderedItem in this example. Every instance of an ordered item has to have an association to exact one instance of the Item – only existing Items can be ordered.

An instance of an Item may or may not be associated with every OrderedItem – not every Item must be ordered.



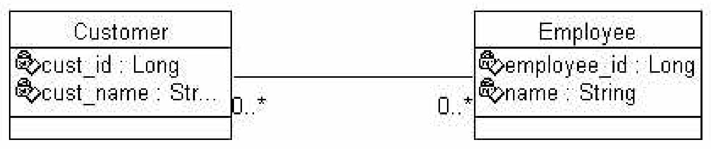
A non-identifying relationship is used to specify the relationship between the tables.



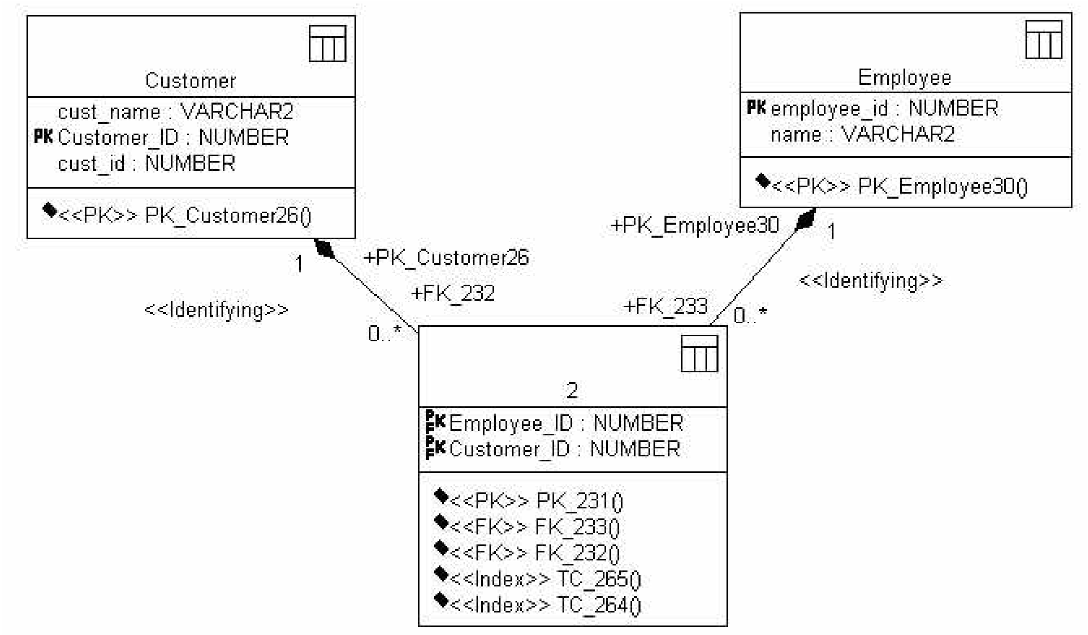
A foreign key on the column item_id is generated to build the relationship. As with every foreign key a foreign key constraint is also generated.

M:N association maps to 3 tables

The object-oriented design allows m to n (many-to-many) associations between classes. Many Employees in the example care about one Customer. One Employee cares about many Customers.



As the relational data model does not allow m to n relationships, an additional table, called an associate table, must be created. The associatetable splits the m to n relationship into two 1 to n relationships, using identifying relationships.

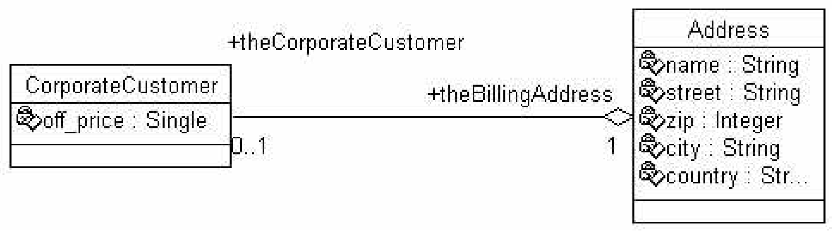


The primary keys of both basic tables are used as primary and foreign keys in the relationship table. The corresponding primary and foreign key constraints are generated.

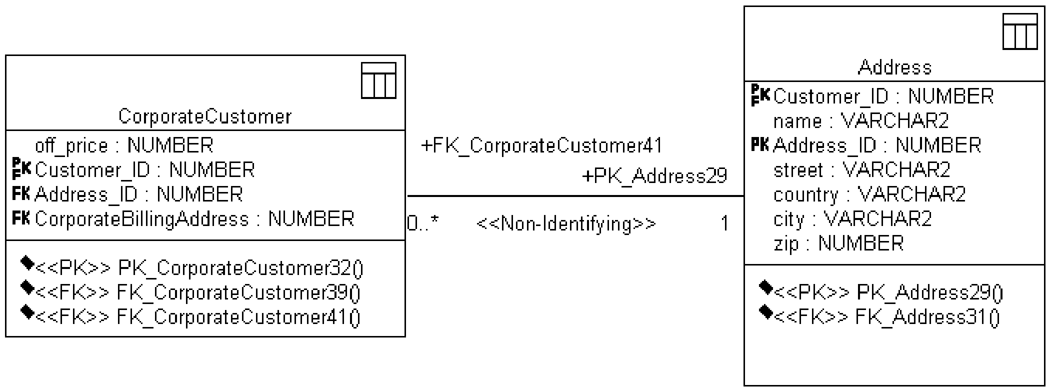
Indexes can be built to improve the performance of database access.

Aggregation by reference maps to a non-identifying relationship

An optional aggregation by reference, as in the case of the BillingAddress, maps to a non-identifying relationship. An Address can be the billing address of a CorporateCustomer.



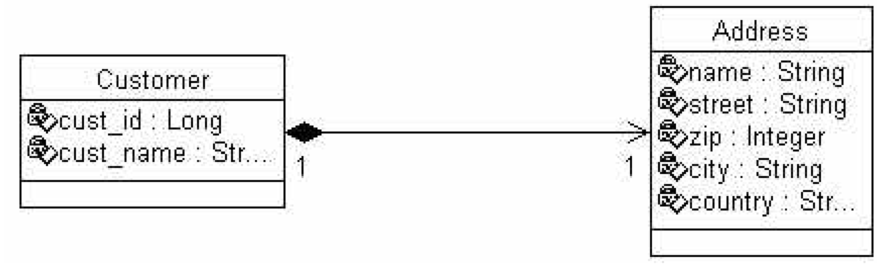
The data model uses a non-identifying relationship to represent this type of relation.



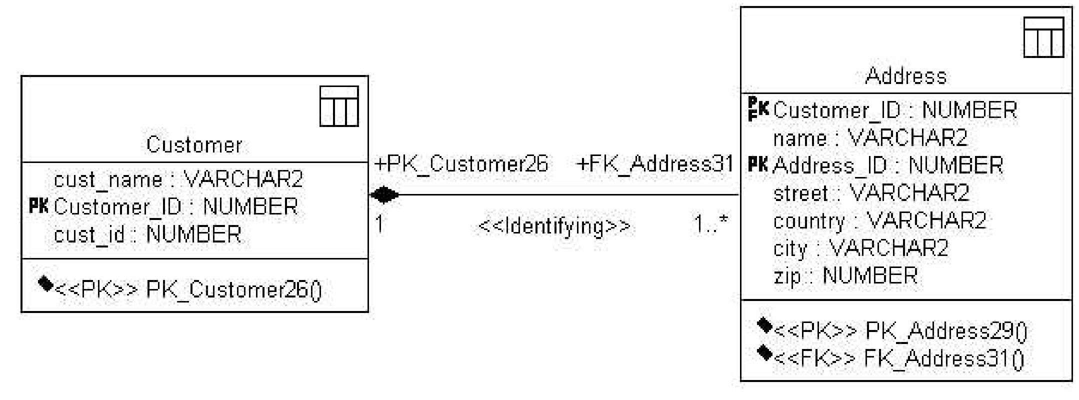
The CorporateBillingAddress is used as a foreign key for the CorporateCustomer table and may be null. The foreign key constraint must be generated.

Aggregation by value (composite aggregation) maps to an identifying relationship

A non-optional aggregation (by value) is used in this example to represent the Address as a part of the Customer, resulting in two separate objects which act as one.



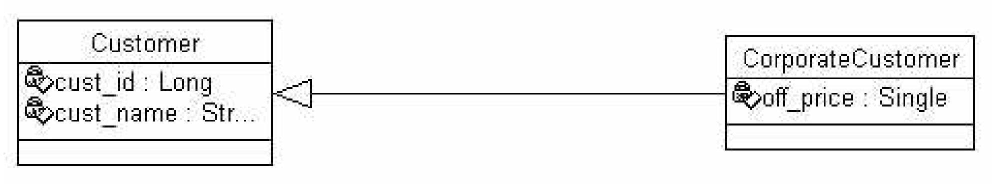
The corresponding representation in the data model is the identifying relationship.



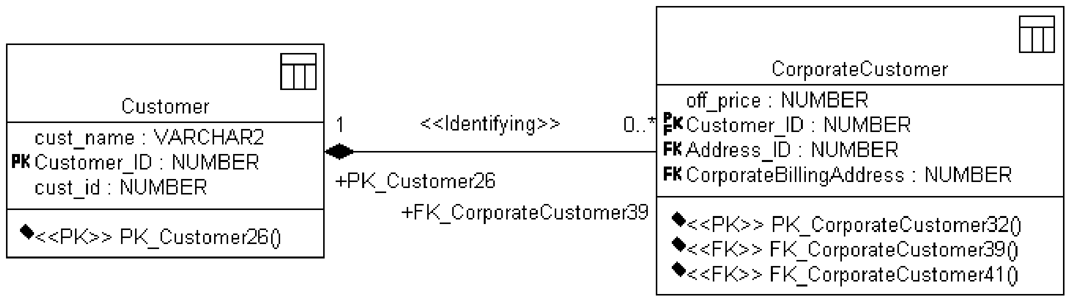
The primary key of the Customer table migrates to the Address as foreign and primary key. A composite primary constraint and a foreign key constraint are built for the address table based on the relationship.

Generalization maps to identifying relationship

Generalization specifies “a kind of” relation between two tables. CorporateCustomer is a kind of Customer.



The corresponding data model specifies two tables and an identifying relationship.

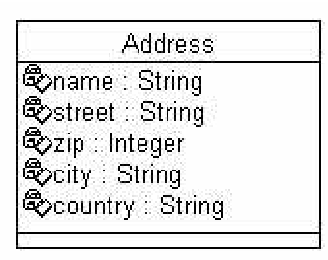


The primary key of the base table migrates to the child table as a primary and foreign key. The primary and foreign key constraints are specified.

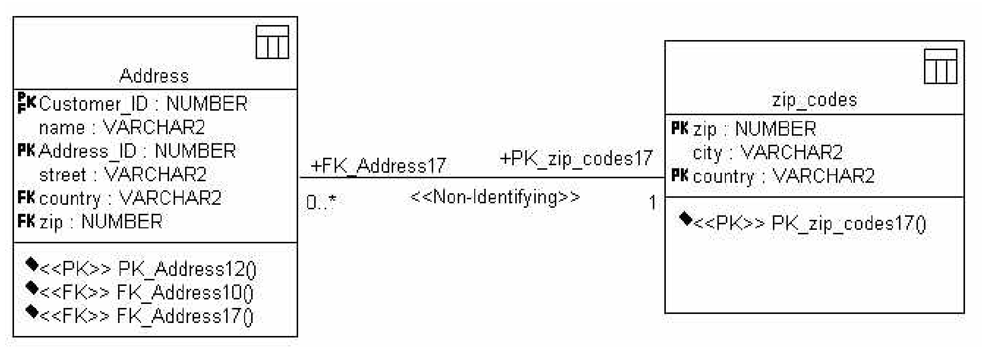
One class can map to several tables

In the process of normalization the data model is often split into more tables to reduce data redundancy.

The Address table, for example, is quite redundant because of the zip code definition.



The data analyst will in most cases split one table into two – the Address table and the Zip_codes table.



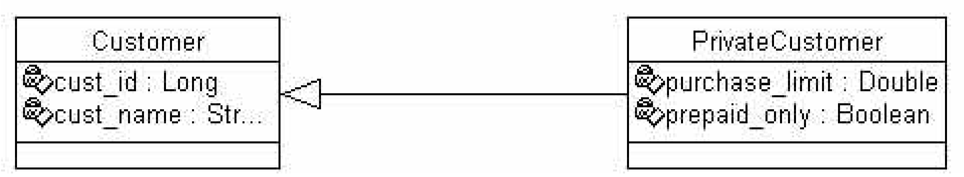
A non-identifying relationship is used to specify the relationship between the tables. The primary key of the outsourced table is used as foreign keys in the Address table.

The foreign key constraints must be generated.

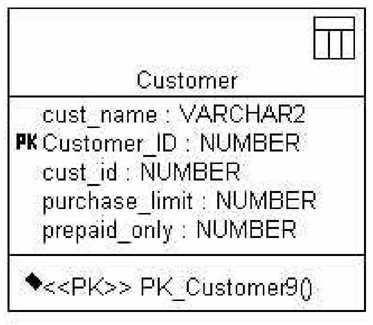
Multiple classes can map to one table

Because of performance and data accessibility the data model is often de-normalized. This results in the mapping from multiple classes to one table.

In the example the PrivateCustomer just adds some attributes to the Customer. The attributes are never used with other types of customers, but the data analyst could decide to make the columns within the PrivateCustomer nullable.



As a result just one table of Customer is used in the data model. This table can be further refined for other types of customers, but it already contains all of the columns of the PrivateCustomer. A data modeler may add additional columns as well. For example, to map the PrivateCustomer, a column called cust_type could be created.



In most cases, merging tables requires additional checks at the application logic to qualify data, or the definition of additional views on a table for different accessibility.

In most cases, he data analyst makes decisions about merging tables based on optimizing the database for data access.

Summary

Mapping object to data models is not easy. The object-relational mapping must be updated continuously as the requirements, object and data model change.

There are several levels of mapping – from the database, schema, up to table and column. The examples in this white paper are not complete. There are additional types of associations and additional mapping examples.

Tracking of object-relational mapping is the key to success when building database applications.

Rational

the e-development company