Object Relational Mappings

First step to a formal approach
Bachelor thesis

Tom van den Broek

January 30, 2007

Contents

Abstract

The conceptual and technical difficulties that are often encountered when a relational database management system is being used by a program written in an object-oriented programming language are the object relational impedance mismatch. From theoretical viewpoint this is because the difference between the Object and Relational model. There is little scientific research on the topic of Object Relational Mappings this paper tries to give take a first step to a formal approach.

Chapter 1

Introduction

Many of the popular programming languages in use today are object oriented. The (information) systems created with these languages often also follow the object oriented paradigm and are designed/modeled according the object oriented approach by using underlying concepts such as *encapsulation*, *inheritance*, *interfaces*, and *polymorphism*.

In an optimal situation the information contained in these systems would be stored in a form which has the same semantics as it has in their originating system. In this case it would be preferable to store the information as objects. This is possible by using a so called Object Databases System (ODBS). However while gaining popularity, they are used in specialist niche area's like telecommunications, and scientific areas such as high energy physics and molecular biology.

Instead, by far the most used storage solution is a Relational Database System (RDBS). These database systems are often called traditional databases because they have been in use since the early 70's [1] and are based on well understood and proved principles.

The combination of an object oriented system and a relational database is the most common solution for object oriented systems which have a need for persistent storage. However, in order to use a relational database for the storage of objects, a translation mechanism from object to relational data is needed. Such a mechanism is most commonly referred to as an Object Relational Mapping (O/RM). A object relational mapping most often takes the form of a software layer or library that is mapping the objects to relational equivalents while hiding this process from the programmer.

The relational model and the object model are two different models for describing a (information) universe. These models are fundamentally different in the way they represent information.

Due to the intrinsic differences between the two models, the combination of the two have proven to create a lot of, both practical and theoretical difficulties. This set of problems are often called the object-relational impedance mismatch.

The existence of the many different object-relational mapping software packages implies that at least the problem has been addressed from a practical viewpoint. However, there is little literature describing the theory behind the differences between the Object model an the Relational model especially when focusing on the feasibility of an Object Relational Mapping.

When looking at the theoretical feasibility of an Object-Relational Mapping the fundamental question that arises is: **'How can an object oriented model be transformed in a relational model?'**.

This paper will try to find an answer on how the object model can be transformed into a relational model. In order to be able to do this, a large part of the paper will consist of an effort to define the impedance mismatch by specifying the difference between the two models not only from a practical level, but also on a more abstract level.

It would be outside the scope of the bachelor thesis to create and prove a full transformation algorithm for the transformations between the two models. The focus will be on describing the different transformation approaches and creating a starting point for a complete transformation model for the object an relational model.

Chapter 2

The two models

In an object relational mapping there are two models relevant: the object model and the relational model. In order to be able to define the differences between the two models and be able to look into the transformations between those models the two different models need to be well defined.

In this chapter each of the two models will be described and given a clear definition. In the next chapter these definitions will be used in order to compare these models. The definition of the two models is inspired by the paper of M. Fussel [2].

2.1 The object model

Object modeling is an approach to structure an information universe into entities called objects.

A much used modeling language for the visualization of object models is Unified Modeling Language (UML) [3]. The language defines a rich set of modeling notions together with a graphical notation which contains the elements used to visualize them. For this reason where possible UML is used for the examples (i.e. figure 2.1).

Most of the time we perceive the world around us in a manner that is close to the principles behind the Object model as used in this paper. All that we perceive can be classified as objects. So in a sense one could say that everything is an object. This view of a simplistic concept of an objects is better explained by the following example.

When walking in a forest there are objects called trees, bushes, animals.

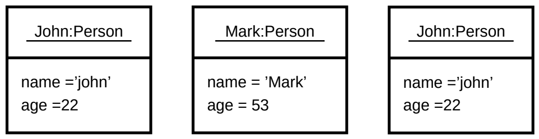

Figure 2.1: Three objects of the class ‘Person’

Each tree can be looked at as a specific kind of tree. An Oak or for example an Conniver, but they are both still trees.

2.1.1 Concepts

With the object model there are some concepts that define for a major part the dynamics of the model. This section will try to give an overview of the concepts relevant in the context of this paper.

Identity

Each object is uniquely identifiable. This means that every object in a universe is distinguishable from any other object in the same universe even when its state is the same. A real world analogy would be: If you see two footballs it is clear that both balls are different balls. When it would be very hard or even impossible to distinguish them, they still would be different balls.

State

The state of an object is also called the ‘internal’ state, it consists of the values of the attributes from an object. In figure 2.2 you see a simple object instance of the type Address. It has three attributes which have assigned to them the values ‘mainstreet’,‘6565TS’ and ’New Hill’. The specific values assigned to the attributes of the object are considered the state in which the object resides.

Behavior & Encapsulation

Behavior and encapsulation are two closely related concepts. Within the object model it is impossible to directly manipulate the ‘state’ of an object.

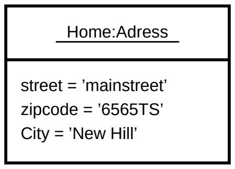

Figure 2.2: An object instance which shows its internal state

Instead the values of the internal attributes of an object can only be manipulated through methods. The set of operations on an object of a certain type is called an interface and is often referred to as the 'behavior' of the object.

As stated above, all possible transformations of the state of an object occur trough the interface. This means the state is never directly manipulated, but only trough the operations defined by the behavior. This concept is called encapsulation, which does exactly what the name implies: shield the internal state from direct manipulation. This can be understood when imagining that the internal attributes of an object are encapsulated inside the object and thus only accessible through predefined behavior of the object.

Class & Type

A type is a specification of the interface an object supports. Two objects are of the same type when they implement the same interface. In theory an object can implement multiple interfaces at the same time. The type of an object that implements multiple interfaces would be defined by the interface consisting of the combination of the multiple interfaces. Often in object oriented models the Class concept is used instead of Type. The only differences is that the class concept defines not only a type but also gives a base implementation for an object of that type. In other words a class is a blueprint of an object of a type. Since the similarities in their definition and meaning, often the concept of a class instead of type is used when discussing the type of an object.

Inheritance

Inheritance is a mechanism that can apply on types and classes. When a class inheritances from another class it means that it will use the implementation

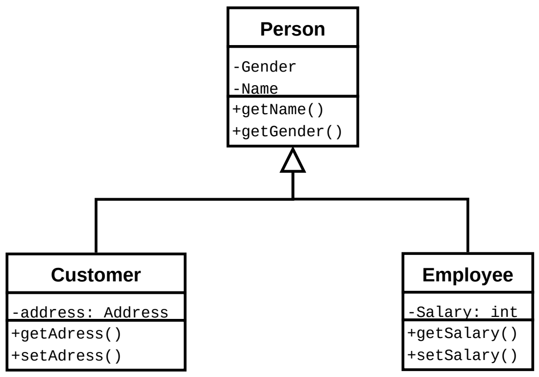

Figure 2.3: A class diagram which consist three classes which inherit from each other

of the its base class. It is also possible that some of the implementation is overridden in the resulting class.

For example, in figure 2.3 there are three classes: Person, Employee, Customer. Employee and customer both inheritance from the class person. This means that an object of class Employee would also implement the method ‘getName()’ and ‘getGender()’ from the class Person.

A real world example: An oak is a type of tree. There are certain characteristics that are true for all trees, for instance they all have leafs. Yet there are specific characteristics for oaks, such as the form of its leafs.

2.2 The relational model

The relational model, developed by E.F. Codd [1] in early 70’s, is based on the predicate logic and set theory. The basic concept of the relational model is that all information is described as predicates and truth statements. This is also called the Information Principle. Which states that all information should be represented by data that participate in a relation.

A Relational model describes the only things that are true in the real world. The model is used to create a representation of information by capturing the data as truth statements about the real world.

Each relation represents some real-world person, place, thing, or event about which information is collected.

A relational database is a collection of two-dimensional tables. The organization of data into relational tables is known as the logical view of the database. That is, the form in which a relational database presents data.

2.2.1 Concepts

With the relational model there are some concepts that define for a major part the dynamics of the model. This section will try to give an overview of the concepts relevant in the context of this paper.

Relation

A relation is the basic form in the relational model. It defines a relation in a universe together with the attributes which play a role in the predicate. As can be seen in the rest of this chapter a relation can be interpreted as a truth predicate.

One aspect of a relation which is important but often not explicitly stated is the meaning of a relation.

For example in the relation Person as shown below. Without the meaning it can be very ambiguous what the meaning of the attributes in the relation are.

Person: {Name, Gender, Date of birth}

The meaning of this relation could be. There exists a person which has the name 'Name', is of the gender 'Gender' and is born on 'Date of birth'. This is a quite logical example, but one can see that there are situations in which without the explicit meaning the relation could be easily misinterpreted.

Attribute

Attributes are the elements that participate in a relation. They are defined by a name and a domain of the value it can store. Such a domain define what values are allowed for the attribute. A domain specifies the possible value for the data and the operations available on the data. Examples of domains are *Strings and Integers.*
For example of a relation with the domains of the attributes specified see below.

person:Person

Name	Gender	Zipcode	Country
John	male	4321BA	Netherlands
Susan	female	4321BD	Spain
Mark	male	9876QZ	Netherlands

Table 2.1: Table view of the relational variable ‘person’ of the relation ‘Person’

Person: {Name : STRING , Gender: ENUM(male,female), Zipcode:ZIPCODE, Country: STRING}

Relation value & Relation variable

The value of a relation is the complete set of all the tuples that satisfies the relation. While the relational variable or relvar is the variable that contains the relation value

An example of a relation value could be the collection of the next three tuples.

{{“John”, male, 4321BA, “Netherlands”}, {“Susan”, female, 4321BD, “Spain”}, {“Mark”, male, 9876QZ, “Netherlands”}, }

Often a relation variable is displayed in the form of a table. This can be seen in Table 2.1.

Chapter 3

Impedance mismatch

In the previous chapter the two different models are described. This was done with the intention to bring the reader up to speed in order to be able to discuss the differences and problems related to using a combination of both models.

As stated in the introduction of this paper, the set of conceptual and technical difficulties which are often encountered when a relational database management system is being used by a program written in an object-oriented programming language or style is often referred to as the object relational impedance mismatch.

When looking from a more theoretical viewpoint it becomes clear that this actually is comparable with the problems encountered when trying to transform from the object model to the relational model and the other way around.

The impedance mismatch is not a clearly defined definition and thus is only used as a convenient term when referencing to the problems stemming from the global differences between the object and relational model. The members of the impedance mismatch are not only theoretical differences. There are also differences from a more philosophical and practical viewpoint when looking at the manner of how the two models are used.

3.1 Object Identity

The object and relational model have a different concept of identity. In the object model as shown before the identity is an intrinsic part of an object and does not depend on the rest of the model.

In the relational model identity has a completely different interpretation. Two relation values, rows, in relational database are considered identical.

Objects however, always have a unique identity. Two objects which happen to have the same state at a given point in time are not considered to be identical. Relations, on the other hand, are viewed as data records with no concept of identity.

3.2 Inheritance

One of the most distinguishing concepts in the object model is inheritance. Inheritance makes it possible for an object to be derived from another object. The problem when transforming an object model to a relational equivalent is that in the relational model there is no such concept as inheritance. Several approaches are possible to be able to model inheritance in a relational model. This topic will be further explored in section 4.2 which is about possible transformations solutions.

3.3 Structure vs. Behavior

The object model primarily focuses on ensuring that the structure of the model is reasonable whereas a relational model focus on what kind of behavior the resulting run-time system has. Object-oriented methods generally assume that the primary user of the object-oriented code and its interfaces are the application developers. In relational systems, the end-users' view of the behavior of the system is considered as more important.

The difference in focus between the object and relational model can be clearly seen when looking at important aspects of the object model as inheritance an encapsulation.

3.4 State

One of the problems is where the state resides in the different models. In a relational model the state of the system resides in the relation variables or tables. This means that the database is the only authoritative repository of state.Any copies of such state held by an application program are just that temporary copies which may be out of date, if the underlying relation was subsequently modified by a transaction.

This view of a central leading state is a contradiction to the approach preferred in object oriented systems. There the state is viewed as an in memory representations of the objects and the database is used as a backing storage and persistence mechanism. This can lead to all kind of consistency problems when mixing these views.

3.5 Access Rules

In the object model all the operations on the objects attribute are performed through methods which are defined in the objects own type. This is called the objects interface and defines how the object can 'behave'. In the relational model the state can only be accessed or altered by using the relational operators available in the relational model.

Chapter 4

Object Relational Mapping

Many of the popular programming languages in use today are object oriented. While the most used database systems are relational database. This results in need for translation mechanism from objects to relational data. Such a mechanism is most commonly referred to as an Object Relational Mapping or O/RM.

4.1 Object identity

When transforming an object model to a relational model one of the primary obstacles is how to represent the unique object identity in the relational model. The most common solution is to model each object as an ID attribute with a primary key constraint.

In the relational model the primary key constraint defines a unique tuple in a relation. This means that the use of the primary key is a simple method to model object identity.

This is done by adding to each relation an attribute with a function to be a unique key used for identifying the object. How exactly this is done is shown in the next section. In some of the approaches an attribute needs to reference to an other object. This can be solved by the Foreign Key constraint that specifies that the value a attribute in a relation should be the same as a value present in an attribute in another relation. This way when needed a reference to an object can be simulated by having a foreign key to the objects Primary Key.

In the following section a lot of the examples will make use of this tech-

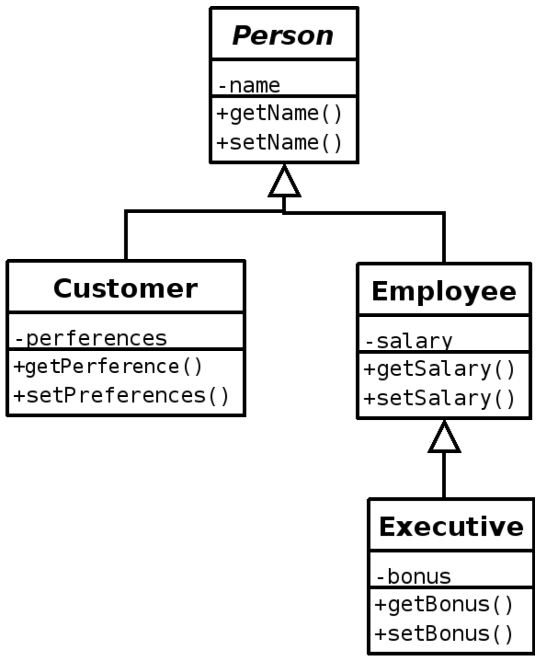

Figure 4.1: simplified UML object model of example structure

nique. See figures 4.2,4.3,4.4.

4.2 Mapping inheritance

When trying to transform the object model to a relational model one of the biggest differences is the lack of a concept of inheritance in the relational model. The transformation of this feature of the object model causes more difficulties than most other features of the model as described in Section 2.1. This is also the reason that most approaches for transforming the object model to a relational model focus mainly on how the concept of inheritance is modeled.

There are four main approaches on mapping an object model. The first three are methods that mainly focus on how to model the concept of inheritance the fourth is a completely different approach on the transformation. In the rest of this section there will occasionally be made use of examples to clarify these methods. These examples examples make use of the case in which the object model as displayed in 4.1 is needed to be transformed in an equivalent relational structure.

4.2.1 One class hierarchy single table

The first approach is the most straight forward. Each class hierarchy is modeled into one relation. This means that all objects of the same type or of a generalized type are stored in one single relation. In the example shown in figure 4.2 each of the derivable classes is transformed in the same relation. The Executive, Employee and Customer although different from each other all inherit direct or indirect, in the case of Executive class, from the abstract class Person. The relation shown in the example exploits the attribute of inheritance that each type that generalizes from a base class can be represented by the type of that base class. This means that an Object of type Customer can be represented as type Person since Customer inherits from Person.

In the example two attributes are added to the relation PersonID and PersonType. The first column is used to identify the objects by use of a Primary Key as explained in Section 4.1. The second column ‘PersonType’ is used to store the actually type of class of the object stored. For example an object of class Customer would have a value corresponding with the Class

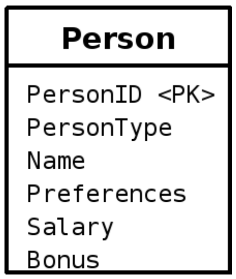

Figure 4.2: Example of a relational diagram which shows the one class hierarchy to a single relation approach.

'Customer' when stored in this structure.

Undefined values

One of the problems when using this approach is that is suffers of possible undefined values in the resulting relational model. The problem is that in the relational model it is in theory not allowed to have so called 'NULL' value in a relational variable. These undefined values impose all kind of difficulties on the relational model when allowed.

While most agree that undefined values are bad manner when applied in a relational model there is yet no conclusive stand on if they should be allowed. This discussion is a bit moot since most available relational database do allow undefined values and they are often (mis)used in practical application. For more information the reader is encouraged to look at the paper of Carlo Zaniolo about null values in databases [4] for a more in depth discussion of this topic.

Our solution however does not take into account the problems allowing null values in your relational model could produced. When this wouldn't be allowed this solution would be invalid in its current form. However it is quite possible that with some refactoring this solution would be possible without the need for allowing undefined values.

4.2.2 One concrete class single table

With this approach a table is created for each concrete class, each table including both the attributes implemented by the class and its inherited at-

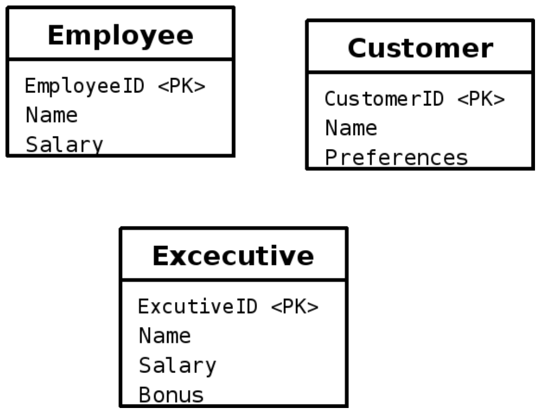

Figure 4.3: Example of a relational diagram which shows the three different concrete classes mapped to three tables

tributes. Figure 4.3 shows the object model transformed to a relational model using this approach. There are tables corresponding to each of the Customer and Employee classes because they are concrete, objects can instantiated from them. There is no Person Table since the person class in an abstract class which means that no objects can be instantiated from this class.

Again each table has its own primary key to be able to identify the different objects. These fields are customerID, employeeID and executiveID.

4.2.3 One class single table

This approach is reasonable straight forward. Each class is directly mapped to its own table. This approach differs from that previous approach that not only concrete classes are mapped but also that each table only consists of the attributes that are defined in that class and not those who are inherited. The linkage between the inherited classes is provide by a baseID field. This field is a foreign key to the base class of the object.

The example, in figure 4.4, shows the different classes linking to each other by use of the ‘PersonID’ field as a foreign key to the id field of the base class of that object.

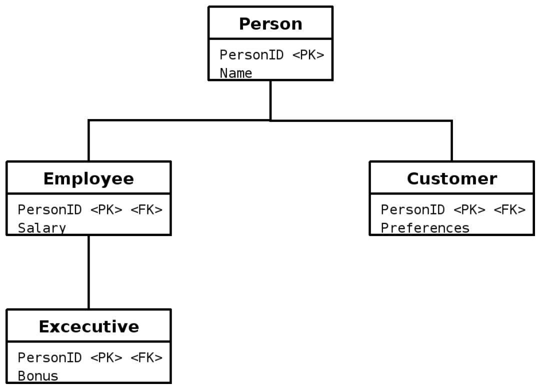

Figure 4.4: Example diagram of the mapping of each class to its own table

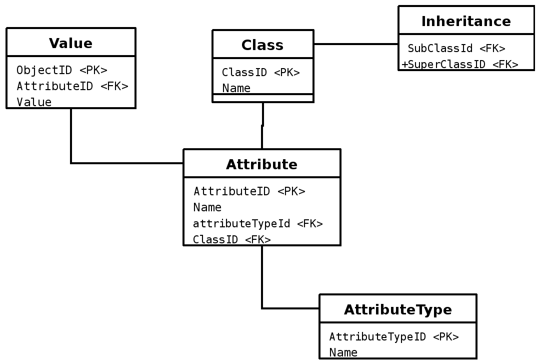

Figure 4.5: simplified UML object model of example structure

4.2.4 Meta-model in relational structure

The final method uses a different approach from storing an object hierarchy in a relational structure. instead of transforming each new object universe to a relational equivalent, it consists of a relational representation of the object model in a relational structure. This means that each object universe should be able to be stored in this structure without changes to the relations.

In Figure 4.5 you can see an example diagram of the relations for storing generic objects. It should be noted that this is only a simple example which is far from complete. It probably would be a lot more complex in its final form.

Chapter 5

Comparison

In this chapter the different approaches to transforming an object model to a relational model are evaluated. Hopefully this creates a coherent overview of their strengths and weaknesses, for each of the four techniques are compared and there possible uses are discussed.

5.1 The simple approach

This straight forward approach is about using one relation to store all objects with the same base class.

5.1.1 Comparison

Below a number of points on which this method has been evaluated.

+ Fast data access. All the object from one hierarchy reside in one relation. It will be fast and efficient when accessing or manipulation objects since all objects only exist of one row each in the relational model.

+ Although, not discussed in depth in this paper this approach makes it easy to change the type of class by simply changing the 'objectType' attribute of the row. This is also called polymorphism

- Expensive in the operations on the object hierarchy. When the object hierarchy changes in any way, the relation has to be modified, which means that for all stored object rows need to be modified. This applies for all operations on the class structure or the class tree structure.

- Since each class is transformed to the same relation, possibly quite a lot of storage is wasted in the relation. Since for each object attributes in the relation are reserved which are not part of this object.
- Tables can grow quickly for large hierarchies.
- This solution depends on allowing undefined values (NULL) in the database tables. This is considered bad practice.

5.1.2 Uses

This approach is an easy and quick solution for simple class hierarchies whose structure does not change. However due to its inefficiency for large class hierarchies and lots of objects this approach will quickly lose its usefulness.

5.2 One table per concrete class

This approach uses a relation for each concrete class.

5.2.1 Comparison

Below a number of points on which this method has been evaluated.

+ The same as with the previous method. Only this time each concrete class has its own relation. Still, it will be fast and efficient when accessing or manipulating objects, since all objects only exist of one row each in the resulting relational model.
- When you modify a class you need to modify its table and the table of any of its subclasses. This is less expensive than with the previous approach, since now only the relation of the class and its subclasses need to be modified and not all classes in the hierarchy.

5.2.2 Uses

This approach is best used when the object hierarchy is relatively simple and is not likely to change a lot.

5.3 One table per class

5.3.1 Comparison

Below a number of points on which this method has been evaluated.

+ Easy to understand because of the one-to-one mapping.
+ Supports polymorphism very well as you merely have rows the appropriate tables for each type.
+ Very easy to modify classes and add new subclasses as you merely need to modify/add one relation.
+ Data size grows in direct proportion to growth in the number of objects.
- There are many tables in the database, one for every class.
- Since each object consist of multiple rows it is complex to manipulate one complete object. How severe this will impact the performance depends on how this operation is implemented how severe this will impact the performance (the use of views, etc).

5.3.2 Uses

This approach is different from the previous methods in that its focus is more on the manipulation of the structure of the object hierarchy and less on efficient behavior when dealing with large numbers of objects.

5.4 Generic schema

The final method was about creating a relational implementation of the object model or a so called meta model.

5.4.1 Comparison

Below a number of points on which this method has been evaluated.

+ Due to the type of the approach it is very easy to change the fundamentals of the object model. If a change to the object model would occur it would be easy to extend the meta model.

+ Is very efficient when dealing with manipulation of the structure of objects or object hierarchies.

- It is the most complex approach and thus can be hard to implement.

- Each object consists of many rows. For this reason the performance is slow compared to the other solutions.

5.4.2 Uses

This method is very useful when dealing with a complex object universe in which the object structure is likely to change and the number of objects stay with in known boundaries.

Chapter 6

Conclusions

In this paper the goal was to create an overview of the main difficulties when considering the concept of an object relational mapping from a theoretical viewpoint.

As stated in the introduction the main research question was the following: **'How can an object oriented model be transformed in a relational model?'**.

In order to answer this question first the Object model and Relation model were formalized in chapter 2. With use of the description of the models and how they relate to eachother an overview of the problems, all together called the impedance mismatch, was given. In the final part of this paper the different approaches to transforming an object model to a relational model are introduced and finally compared.

The methods discussed for transforming the object model certainly show that it is possible from a theoretical viewpoint to create a theory on object relational mapping. Each of the different methods discussed has its own strengths and weaknesses. For now it is not possible to recommend one of the approaches as the best. Which solution is best suited depends on the kind of needs of the situation.

Future research could focus on creating a complete formalization of the concept of Object Relational Mapping. For now this paper could serve as an introduction to the subject.

Bibliography

[1] E.F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *IBM Research Report*, 1969.

[2] Mark L. Fussell. *foundations of object-relational mapping.* White Paper, ChiMu Corp*, 1997.*

[3] Object Management Group. Unified modeling language (uml) specification, version 2.0. 2006.

[4] Carlo Zaniolo. Database relations with null values. In Symposium on Principles of Database Systems*, pages 27–33, 1982.*