Chapter 2
Data and Metadata Management

Paolo Atzeni and Riccardo Torlone

Abstract In this chapter, we illustrate fundamental notions of Data Management. We start from the basic concepts of database schema and instance, illustrating how they are handled within dictionaries in relational databases. We then address, from a general point of view, the notion of data model as a means for the representation of information at different levels of details, and show how dictionaries can be used to manage schemas of different data models. A further level of abstraction, called metamodel, is then introduced with the aim of supporting the interoperability between model based applications. Finally, we discuss a methodology for schema translation that makes use of the metamodel as an intermediate level.

2.1 Introduction

A fundamental requirement in the whole life cycle of any database application is the ability of looking and managing data at different levels of abstractions. According to a widely accepted vision in the field of databases, at the lowest level of abstraction, we have raw data, which cannot provide any information without suitable interpretation. At a higher level we have *schemas*, which describe the structure of the instances and provide a basic tool for the interpretation of data. Actually, at this level we can have further forms of *metadata*, that is, information that describes or supplements actual data. Notable examples are the *constraints*, which allow the specification of properties that must be satisfied by actual data. Then we have a third level of abstraction which involves precise formalisms for the description of schemas called *data models*. A data model is a fundamental ingredient of database applications as it provides the designer with the basic means for the representation of reality in terms of schemas.

Unfortunately, a large variety of data models are used in practical applications and, for this reason, the interoperability of database applications is historically one of the most complex and time-consuming data management problems. In order to alleviate this problem, it is useful to add to the picture a further level of abstraction

P. Atzeni (✉) · R. Torlone
Università Roma Tre, Via della Vasca Navale 79, 00146 Rome, Italy
e-mail: atzeni@dia.uniroma3.it

R. Torlone
e-mail: torlone@dia.uniroma3.it

R. De Virgilio et al. (eds.), *Semantic Web Information Management*,
DOI 10.1007/978-3-642-04329-1_2, © Springer-Verlag Berlin Heidelberg 2010

providing a unified formalism, which we call *metamodel*, for the representation of different models. The definition of a metamodel able to describe all the data models of interest has two main advantages. On the one hand, it provides a framework in which database schemas of heterogeneous models can be handled in a uniform way and, on the other hand, it allows the definition of model-independent transformations that can be used in multiple translations between different models.

The goal of this chapter is to discuss these fundamental issues related to the management of data and metadata. We start from the notion of database schema and show how schemas are handled in practice by relational systems through special tables of the *dictionary*, a portion of the database used by the system to store any kind of metadata. We then illustrate various logical and conceptual data models, showing that it is possible to manage schemas of different models by just adding new tables to the dictionary, one for each new construct of the model at hand. This approach leads to a metamodel made of a collection of model-independent construct types, which we call *metaconstructs*, used in models to build schemas and corresponding to the tables of the extended dictionary. Interestingly, it comes out a rather compact metamodel, coherently with the observation that the constructs of most known models fall into a rather limited set of generic types of constructs: lexical, abstract, aggregation, generalization, function [25]. This approach can greatly simplify the translations of schemas between heterogeneous models since we can use the metamodel as a “lingua franca” in which schemas are converted and transformed to suit the features of the target model. The availability of a repository of basic transformations defined at the metamodel level, and so model-independent, can further simplify the whole translation process.

The rest of the chapter is devoted to a more detailed treatment of this matter and is organized as follows. Database schemas and models are discussed in Sects. 2.2 and 2.3, respectively. In Sect. 2.4, we illustrate the metamodel approach to the management of multiple models and, in Sect. 2.5, we present the translation methodology based on such a metamodel. In Sect. 2.6, we discuss related work and finally, in Sect. 2.7, we sketch some conclusions.

2.2 Databases, Schemas and Dictionaries

The notion of *database* as a large, shared, persistent collection of data to be managed effectively and efficiently is now well accepted and described in textbooks [7, 19, 23, 33]. Among the major features in database management, the most relevant here is the use of structures that allow for the high-level, synthetic description of the content of a database: a database has a *schema*, which describes the features of data, and an *instance* (sometimes called *database*), which includes the actual content. The schema is rather stable over time (it can vary, but usually in a very slow manner), whereas the instance changes continuously. For example, in a relational database we have the database schema that contains a number of relation schemas, each defined by means of an SQL `CREATE TABLE` statement, which specifies the name of the

Employee

EmpNo	LastName	Dept
1001	Rossi	IT
1002	Sagan	FR
1003	Petit	FR
1004	Grand	FR

Department

DeptID	Name	City
IT	Italy	Rome
FR	France	Paris
DE	Germany	Hamburg

Fig. 2.1 A simple relational database

table,[1] its columns (each with the respective type), and the constraints defined over it (for example, its primary key). Then, the actual content of tables is the result of `INSERT` (and also, `UPDATE` and `DELETE`) statements, which refer to the actual rows. Database schemas are a form of *metadata*, that is, data that describes actual data.

In the usual graphical representation for the relational model (Fig. 2.1), relations are shown in tabular form, with the headings that essentially correspond to the schema (types are omitted, but can be easily guessed or understood) and the bodies that contain the data in the instance.

The major benefit of the distinction between schemas and instances is the fact that most of the actions over the database refer to the schema. In particular, if we want to extract data from a database (usually we say that we *query* the database), we need only mention schema elements, and just a few values (if any) that are relevant to the specific operation. For example, if (with respect to the database in Fig. 2.1), we are interested in finding the list of the employees, each with the city of the corresponding department, we can write an expression that contains only the names of the tables and of the columns, and no specific data. Schemas are also fundamental in the choice of the appropriate physical structures for the database, an aspect that is essential for pursuing efficiency. Physical structures are defined in the final phase of database design, and refer to the database schema alone.

In relational databases, a description of schemas is usually handled within the database itself. Indeed, a portion of the database, called the *dictionary* or the *catalog*, is composed of tables that describe all the tables in the database. In this way, data and metadata are managed in an uniform way. Figure 2.2 shows the idea of the dictionary, with its main tables (a real one contains dozens of them), which refers to the database in Fig. 2.1. Let us briefly comment on it. A database environment usually handles various schemas, here we assume that there is the *system* schema, which describes the dictionary itself, and a *user* one (called Sample), which refers to the database in Fig. 2.1. They are listed in the Schema relation. The relations of interest are both those in the *user* schema and those in the *system* schema. So, we have five of them, listed in the Table relation. Finally, the Column relation has one tuple for each attribute in each of the relations in the database. In the dictionary, we have used numerical identifiers (SchemaID, TableID, ColumnID) over

[1] Following SQL terminology, we use *relation* and *table* as synonyms. Similarly for *column* and *attribute*.

SCHEMA

SchemaID	SchemaName	Owner
1	System	System
2	Sample	Paolo

TABLE

TableID	TableName	Schema
11	Schema	1
12	Table	1
13	Column	1
21	Employee	2
22	Department	2

COLUMN

ColumnID	ColumnName	Schema	Table	Type
101	SchemaID	1	11	int
102	SchemaName	1	11	string
103	TableID	1	12	int
104	TableName	1	12	string
105	Schema	1	12	string
106	ColumnID	1	13	int
107	ColumnName	1	13	string
...	...	...	...	...
201	EmpNo	2	21	int
202	LastName	2	21	string
203	Dept	2	21	string
204	DeptID	2	22	string
205	Name	2	22	string
206	City	2	22	string

Fig. 2.2 A simplified dictionary for a relational database

which cross references have been built (so that, for example, the value 22 in the last row of COLUMN means that City is an attribute of the relation whose identifier is 22, namely DEPARTMENT).

2.3 Data Models

The notion of *data model* has been considered for decades in the database field (see for example the book by Tsichritzis and Lochovsky [36] for an early but still current discussion). A synthesis, useful for our goals, is the following: "a data model defines the rules according to which data are structured" [36, p. 10]. Specifically, we can say that a model includes a set of structures (or better, types of structures), called constructs, together with the rules that can be used to combine them to form schemas. Other features can be important for models, including *constraints* (that is, rules that specify properties that must be satisfied by instances) and *operations* (which specify the dynamic aspects of databases, that is, how they evolve over time). Here, for the sake of space, we concentrate on structures which allow for synthetic and effective arguments, omitting operations and devoting only limited attention to constraints.

It is common to classify data models according to two main categories (indeed, the distinction is useful, but not so sharp as it might appear at first): *logical* models are those used for the organization of data in databases;[2] *conceptual* models are

[2] The term *logical* is used in contrast to *physical*: the latter refer to lower level representation, which takes into account features of secondary storage organization, such as blocks, addresses and allocation, whereas the former refers to the structures that can be seen by users and programs.

	EMPLOYEE		
	EmpNo	LastName	Dept
e#1	1001	Rossi	d#1
e#2	1002	Sagan	d#2
e#3	1003	Petit	d#2
e#4	1004	Grand	d#2

	DEPARTMENT		
	DeptID	Name	City
d#1	IT	Italy	Rome
d#2	FR	France	Paris
d#3	DE	Germany	Hamburg

Fig. 2.3 A simple object-relational database

used in the early steps of design, mainly to produce a description of the concepts of the application world whose data are to be stored in the database.

The most popular logical model nowadays is the relational one, used in the previous section to illustrate the main ideas at the basis of the organization of data in current databases. Various extensions exist for the relational model, mainly under the generic name of *object-relational*, which however includes many variations. Growing interest is being devoted to the management of XML documents in databases, which has lead to the idea that *DTD* and *XSD*, languages for specifying schemas of XML documents, can be seen as data models. Each of these models has a number of constructs. For example, a simple[3] version of the object-relational model could have:

- relations and attributes as in the relational model;
- *typed tables*, another construct for handling sets of elements, where each element beside attribute values (as in traditional relations) has a system-handled identifier;
- *references* between typed tables, implemented by means of the system-handled identifiers;
- *subclass* relationships between typed tables, used to specify that the elements of a typed table refine those in another one.

A tabular representation for an object-relational database is shown in Fig. 2.3: here we emphasize only two new constructs, typed tables, with system-managed identifiers associated with elements (put outside the borders of tables, to suggest that their values are not really visible), and references, implemented by using identifiers in the Dept column to correlate each employee with the department he or she belongs to.

Within the category of conceptual models, there is probably even a greater variety, as the various methodologies and design tools use different models. However, we have again a few well known representatives, each with variants. Let us comment on two of the models.

The most common conceptual model in the database world is the *entity-relationship* (*ER*) model [17]. It is based on three major constructs: (i) *entity*, used to represent sets of things, facts, people of the real world that are of interest for the application at hand; (ii) *relationship*, a logical association between entities, where

[3] As we said, there are many implementations of the idea of an object-relational model, and here we are only interested in the key points rather than the details.

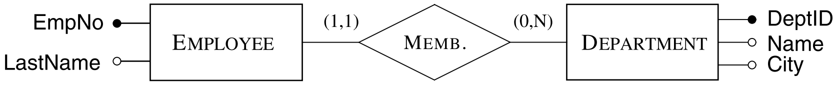
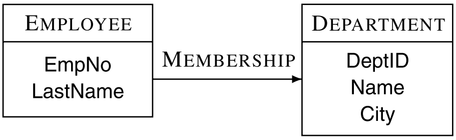


Fig. 2.4 A simple ER schema

Fig. 2.5 A simple object-oriented schema



an occurrence of a relationship is a pair[4] of occurrences of entities; (iii) *attribute*, a function that associates a value with each occurrence of an entity or of a relationship. Schemas in the ER model have an effective graphical representation, as shown in Fig. 2.4, where we have a schema with two entities, a relationship and a few attributes, which correspond to the same application domain as the examples shown earlier for the relational and object-relational model.

Other conceptual models are based on the object-oriented paradigm. In particular, we consider a model similar to *UML class diagrams*, but with many simplifications and some variations, useful for our discussion. From our perspective, and adapting the notation, the constructs of this model are: (i) *class*, very similar to the notion of entity discussed above; (ii) *reference*, a directed one-to-many relationship whose occurrences connect one occurrence of a class with an occurrence of another; (iii) *field*, essentially an attribute for a class. Figure 2.5 shows a schema in this model, according to a suitable notation.

It is worth noting that we could define dictionaries for describing schemas in each of the models. Obviously, we could use any of the models for each dictionary. In order to prepare for a discussion to be carried out in the next section, we show in Figs. 2.6 and 2.7 portions of the relational dictionary for the entity-relationship and for the object-oriented model, with data that describe the schemas in Figs. 2.4 and 2.5, respectively. With respect to Fig. 2.4, let us observe that, while we have omitted some columns, we have shown without explanation a few of them, to give the idea that a number of details can be easily modeled in this way, concerning the identifiers of entities (the IsKey column of ATTRIBUTEOFENTITY) and cardinalities of relationships (IsOpt1, IsFunctional1).

2.4 Management of Multiple Models

The existence of a variety of models, discussed and exemplified in the previous section, suggests the need for a unified management of them. This has various goals, the most important of which is the support to the integration of heterogeneous schemas and their translation from a model to another.

[4] Relationships can be n-ary, but here we refer to binary ones for the sake of simplicity.

SCHEMA		
SchemaID	SchemaName	...
3	SchemaER1	...

ATTRIBUTEOFENTITY					
AttrID	AttrName	Schema	Entity	IsKey	...
301	EmpNo	3	61	true	...
302	LastName	3	61	false	...
303	DeptID	3	62	true	...
304	Name	3	62	false	...
305	City	3	62	false	...

ENTITY		
EntityID	EntityName	Schema
61	Employee	3
62	Department	3

BINARYRELATIONSHIP							
RelID	RelName	Schema	Entity1	IsOpt1	IsFunctional1	Entity2	...
401	Membership	3	61	false	true	62	...

Fig. 2.6 The dictionary for a simple ER model

FIELD					
FieldID	FieldName	Schema	Class	IsID	...
501	EmpNo	4	71	true	...
502	LastName	4	71	false	...
503	DeptID	4	72	true	...
504	Name	4	72	false	...
505	City	4	72	false	...

SCHEMA		
SchemaID	SchemaName	...
4	SchemaOO1	...

CLASS		
ClassID	ClassName	Schema
71	Employee	4
72	Department	4

REFERENCEFIELD					
RefID	RefName	Schema	Class	ClassTo	isOpt
801	Membership	4	71	72	false

Fig. 2.7 The dictionary for a simple object-oriented model

In this section, we illustrate an approach to the unified management of schemas (and data as well, but we do not discuss this issue here), based on the idea of a metamodel, which we have developed in the last few years [6, 8].

We use the term *metamodel* to refer to a set of *metaconstructs*, which are basically construct types used to define models. The approach is based on Hull and King's observation [25] that the constructs used in most known models can be expressed by a limited set of generic (i.e., model-independent) types of constructs: lexical, abstract, aggregation, generalization, function. Our metamodel is defined by a set of generic metaconstructs, one for each construct type. Each model is defined by its constructs and the metaconstructs they refer to. With this approach, the models discussed in the previous sections could be defined as follows (with a few more details than those we noted):

- the relational model has (i) aggregations of lexicals (the tables), with the indication, for each component (a column), of whether it is part of the key or whether nulls are allowed; (ii) foreign keys defined over components of aggregations;

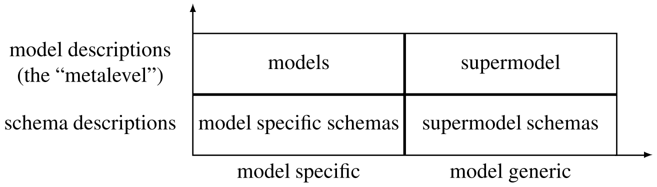


Fig. 2.8 The four parts of the dictionary

- a simplified object-relational model has (i) abstracts (tables with system-managed identifiers); (ii) lexical attributes of abstracts (for example, EmpNo and LastName), each of which can be specified as part of the key; (iii) reference attributes for abstracts, which are essentially functions from abstracts to abstracts (in the example, in Fig. 2.3, the Dept attribute in table Employee);
- a binary entity relationship model has (i) abstracts (entities); (ii) lexical attributes of abstracts, each of which can be specified as part of the key; (iii) binary aggregations of abstracts (relationships);
- an object-oriented model has (i) abstracts (classes), (ii) reference attributes for abstracts, which are essentially functions from abstracts to abstracts, (iii) lexicals (fields or properties of classes).

A major concept in this approach is the *supermodel*, a model that has constructs corresponding to all the metaconstructs known to the system. Thus, each model is a specialization of the supermodel and a schema in any model is also a schema in the supermodel, apart from the specific names used for constructs. It is important to observe that the supermodel is rather stable but not frozen: in fact, the set of models that can be represented depends on the richness of the supermodel. Given a supermodel, should a wider class of models be needed, with "new" constructs, then the supermodel could be extended.

The approach can be synthesized and implemented by means of a complex dictionary [4], capable of handling information on the supermodel, on the various models, and on schemas. The dictionary has four parts, as shown in Fig. 2.8. The core of the dictionary is the upper right portion: it contains a description of the supermodel, with all the constructs it includes, usually a rather limited number. Figure 2.9 shows the relational implementation of such a portion. It includes three tables: SMConstruct, which lists the constructs in the supermodel, SMReference, which specifies how the constructs are related to one another, and SMProperty, which indicates the features of interest for each construct. For example, we can see that each AttributeOfAbstract is related to an Abstract and has an IsId property (to specify whether it is part of the identifier).

The various models are defined in the "models" part (the upper left part of Fig. 2.8) with respect to the supermodel. We show a portion of this part in Fig. 2.10: tables are similar to those in Fig. 2.9, the main difference being the fact that constructs, references and properties have names that are specific to the various models and that there is a table for representing models. Each construct, reference, property is also related to a corresponding item in the supermodel (via smConstruct,

SMCONSTRUCT

ConstructID	ConstructName
1001	Abstract
1002	AttributeOfAbstract
1003	BinaryAggregationOfAbstracts
1004	AbstractAttribute
...	...

SMREFERENCE

RefID	RefName	Construct	ConstructTo
2001	Abstract	1002	1001
2002	Abstract1	1003	1001
2003	Abstract2	1003	1001
...	...	...	...

SMPROPERTY

PropID	PropertyName	Construct	Type
3001	AbstractName	1001	string
3002	AttributeName	1002	string
3003	IsId	1002	bool
3004	IsFunctional1	1003	bool
3005	IsFunctional2	1003	bool
...	...	...	...

Fig. 2.9 The description of the supermodel in the metalevel dictionary

smRef, smProperty, respectively). We omit further discussion as the figure is self-explanatory.

The metalevel dictionary in Fig. 2.10 describes the structure of the various dictionaries we have seen in the previous section. In fact, the rows in the CONSTRUCT table correspond to the tables in the various dictionaries, specifically those in Figs. 2.6 and 2.7. Also, the rows in tables REFERENCE and PROPERTY describe the various columns of the tables in Figs. 2.6 and 2.7. These tables constitute the "model specific schema" component of the multilevel dictionary (lower left part in Fig. 2.8).

We have described so far three of the components of the dictionary sketched in Fig. 2.8. The fourth, the lower right part, is probably the most interesting for the management of schemas in different models and for supporting their translations. In fact, it describes schemas in supermodel terms. An example is in Fig. 2.11. Two observations are important. First, we can see that here we have a table for each construct in the supermodel (so for each row in table SMCONSTRUCT in Fig. 2.9), and contains the descriptions of schema in the various models, according to supermodel terminology. Second, the content of the table is essentially the union of the contents of the respective tables in the model-specific dictionaries, when the two models both have a construct for a given supermodel construct. For example, table ABSTRACT is basically the union of tables ENTITY (in Fig. 2.6) and CLASS (in Fig. 2.7).

MODEL

ModelID	ModelName
5001	ER
5002	OODB
...	...

CONSTRUCT

ConstructID	ConstructName	Model	smConstruct
6001	Entity	5001	1001
6002	AttributeOfEntity	5001	1002
6003	BinaryRelationship	5001	1003
6004	Class	5002	1001
6005	Field	5002	1002
6006	ReferenceField	5002	1004
...	...	...	...

REFERENCE

RefID	RefName	smRef	Constr	ConstrTo
7001	Entity	2001	6002	6001
7002	Entity1	2002	6003	6001
7003	Entity2	2003	6003	6001

PROPERTY

PropID	PropertyName	smProperty	...
8001	EntityName	6001	...
8002	AttributeName	6002	...
8003	IsKey	6002	...
8004	IsFunctional1	6003	...
8005	IsFunctional2	6003	...
...	...	...	...
8011	ClassName	6011	...
8012	FieldName	6012	...
8013	IsId	6013	...
...	...	...	...

Fig. 2.10 The description of some models in the dictionary

2.5 Model-generic Schema Translation

The dictionary we have illustrated in the previous section is interesting as it allows for a model-generic description of schemas which is also *model-aware*, in the sense that it specifies the features of the various models and the model each schema belongs to. We have been experimenting, with a number of applications, a platform that includes such a dictionary towards a more general model management system [3]. The most interesting results we have obtained concern a model-generic approach to schema translation [6, 8]. The problem can be stated as follows: given two data models M_1 and M_2 and a schema S_1 of M_1, find a schema S_2 of M_2 that properly represents S_1. A possible extension of the problem considers also the data level: given a database instance I_1 of S_1, find also an instance I_2 of S_2 that has the same information content as I_1.

SCHEMA

SchemaID	SchemaName	...
3	SchemaER1	...
4	SchemaOO1	...

ATTRIBUTEOFABSTRACT

AttrID	AttrName	Schema	Abstract	IsKey	...
301	EmpNo	3	61	true	...
302	LastName	3	61	false	...
303	DeptID	3	62	true	...
304	Name	3	62	false	...
305	City	3	62	false	...
501	EmpNo	4	71	true	...
502	LastName	4	71	false	...
503	DeptID	4	72	true	...
504	Name	4	72	false	...
505	City	4	72	false	...

ABSTRACT

AbstractID	EntityName	Schema
61	Employee	3
62	Department	3
71	Employee	4
72	Department	4

BINARYAGGREGATIONOFABSTRACT

AggID	AggName	Schema	Abstract1	IsOpt1	IsFunctional1	Abstract2	...
401	Membership	3	61	false	true	62	...

ABSTRACTATTRIBUTE

AbsAttID	AbsAttName	Schema	Abstract	AbstractTo	...
801	Membership	4	71	72	...

Fig. 2.11 A model-generic dictionary based on the supermodel

Since, as we saw in the previous sections, many different data models exist, what we need is an approach that handles the various models in an effective and efficient way. Indeed, with many constructs and variants thereof, the number of different models grow. Indeed, each of the models we have mentioned in the previous sections has many variants, which, in the translation process, require attention. For example, if we just consider the object-relational model, we have a number of features that can appear in different ways, including the following:

- keys may be specifiable or not;
- generalizations may be allowed or not;
- foreign keys may be used or not;
- nested attributes may be allowed or not.

As these features are mainly independent from one another, it is clear that we can easily have hundreds or even thousands of models. Therefore, it would be impossible to have a specific translation for each pair of models, as, for n models, we would need n^2 translations. The supermodel, as we have discussed it in the previous section, can give a significant contribution to this issue: as every schema, in every model, is also a schema in the supermodel (modulo construct renaming), we can use the supermodel as a "pivot" model: we can think of translations as composed of first a "copy" from the source model to the supermodel, then an actual translation within the supermodel, and finally a "copy" back to the target model. This would drastically reduce the needed number of translations, from a quadratic number to a linear one.

However, even a linear number of translations can be too big, if we have hundreds or thousands of models. However, as we said, the size of the space of models is due to the combination of independent variants of constructs. Then, it is possible to define simple translation operations each concerned with a specific transformation concerning a feature of a construct, so that a translation is composed of a sequence of simple steps. For example, if we want to translate an object-relational schema (in a model that does not allow for keys but has generalizations) into the relational model, then we can proceed with four steps:

1. eliminate generalizations;
2. add keys;
3. replace references with foreign keys;
4. replace objects with tables.

The interesting thing is that if we have other variants of the object-relational model, we probably need another combination of elementary steps, but with significant reuse. Let us comment this with the help of Fig. 2.12 in which we show some of the models of interest and a number of elementary translations between them. In the

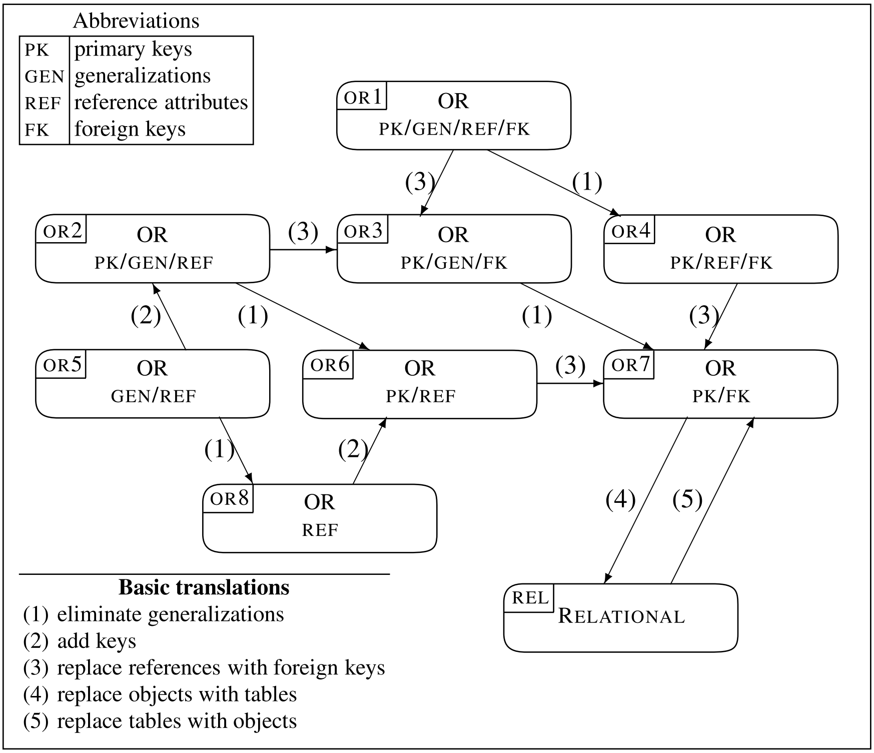


Fig. 2.12 Some models and translations between them

case above, the source model would be OR5 and we would reach the target one (REL) by means of the application of elementary translations (1)–(4). Other translations over models in the same picture could be implemented by means of other sequences of the same elementary translations.

This approach can become very effective if a library of elementary transformations is defined and complex transformations are built by sequences of elementary ones. We have also developed techniques for the automatic selection of translations in the library [6]. In principle, elementary translations could be developed by using any language. We have developed a complete prototype [5] that uses a Datalog with OID invention and handles a significant number of models within various families, namely relational, object-relational, object-oriented, entity-relationship and XSD-based.

2.6 Related Work

Data models are extensively discussed in every database textbook [7, 19, 23, 33] and have always been one of the most investigated database research areas: in the last thirty years, we have witnessed a huge list of proposals for data models, at different levels of abstraction. Early, but still current, comparisons of the main features of various data models can be found in the book by Tsichritzis and Lochovsky [36] and in the survey by Hull and King [25].

On the practical side, even if the ER model is still often used in conceptual database design [9], many variants exists and recently new models which better capture the requirements of novel application domains have emerged [16]. At the logical level we have a similar situation: even if the relational model has imposed itself as a de-facto standard, a large variety of data models (in many cases just extensions of the relational model) are actually used to represent, store, and process data [1]. It follows that sharing and exchange of information between heterogeneous data sources have always been one of the most time-consuming data management problems. Indeed, work on database interoperability dates back to the early 1970s [35] and various aspects of this general problem, such as database federation [34], database integration [10, 26], schema matching [32], schema merging [30], and data exchange [20, 21], have been deeply investigated in the literature.

Recently, Bernstein set the various problems within a very general framework that he called *model management* [11–13]. Its value as a methodology for approaching several meta-data related problems with a significant reduction of programming effort has been widely accepted (see [14] for a number of practical examples).

According to Bernstein [11], model management is based on five basic operators that, suitably combined, can be used to address the vast majority of problems related to database interoperability: (i) *Match*, which takes as input two schemas and returns a mapping between them, (ii) *Compose*, which takes as input two mappings between schemas and returns a mapping that combines them, (iii) *Merge*, which takes as input two schemas and returns a schema that corresponds to their "union" together with two mappings between the original schemas and the output schema,

(iv) *Diff*, which takes as input two schemas and a mapping between them and returns the portion of the first schema not involved in the mapping, and (v) *ModelGen*, which takes as input a schema in a source model and a target model and returns the translation of the schema into the target model.

Several studies have been conducted on model management, basically by focusing on some of the above operators. The Compose operator has been investigated in [27]. Rondo [28] is a rather complete framework for Model Management, but it does not address the problem of model translation, corresponding to the ModelGen operator. Cupid [27] focuses on the Match operator, whereas Clio [24, 29] provides, in a sense, an implementation of both the Match and of the Merge operators. The latter operator has also been studied in [30].

In this chapter, we have discussed a metamodel approach to the management of multiple models that corresponds to an implementation of the ModelGen operator [6]. It should be said that many studies have been done on issues related to model translation: some of them have focused on specific problem (e.g., the XML-relational mapping [22]) or on specific data models (survey papers on this subject can be found in [37]); others have studied the problem from a more general perspective [2, 15, 18, 31]. The main difference between our approach and these works relies on our notion of metamodel that introduces a higher level of abstraction with two main benefits. On the one hand, it provides a very natural way to describe heterogeneous models and, on the other hand, it allows us to define generic transformations between primitives that are independent of the specific models involved in a translation.

2.7 Conclusion

In this chapter, we have addressed some issues at the basis of the management of data and metadata. In particular, we have considered schemas and models, the fundamental ingredients of database applications, and have shown how schemas of different models can be represented in a unified way by suitably extending the dictionary of a database management system. We have then introduced the notion of metamodel, a tool for the uniform representation of different models, to support the translation of heterogeneous schema from one model to another.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kauffman, San Mateo (1999)
2. Abiteboul, S., Cluet, S., Milo, T.: Correspondence and translation for heterogeneous data. Theor. Comput. Sci. **275**(1–2), 179–213 (2002)
3. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: From schema and model translation to a model management system. In: Sharing Data, Information and Knowledge, BNCOD 25. LNCS, vol. 5071, pp. 227–240. Springer, Berlin (2008)

4. Atzeni, P., Cappellari, P., Bernstein, P.A.: A multilevel dictionary for model management. In: ER Conference. LNCS, vol. 3716, pp. 160–175. Springer, Berlin (2005)
5. Atzeni, P., Cappellari, P., Gianforme, G.: MIDST: model independent schema and data translation. In: SIGMOD Conference, pp. 1134–1136. ACM, New York (2007)
6. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB J. **17**(6), 1347–1370 (2008)
7. Atzeni, P., Ceri, S., Paraboschi, S., Torlone, R.: Databases: Concepts, Languages and Architectures. McGraw-Hill, New York (1999)
8. Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: EDBT Conference. LNCS, vol. 1057, pp. 79–95. Springer, Berlin (1996)
9. Batini, C., Ceri, S., Navathe, S.: Database Design with the Entity-Relationship Model. Benjamin and Cummings, Redwood City (1991)
10. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Comput. Surv. **18**(4), 323–364 (1986)
11. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR Conference, pp. 209–220 (2003)
12. Bernstein, P.A., Halevy, A.Y., Pottinger, R.: A vision of management of complex models. SIGMOD Rec. **29**(4), 55–63 (2000)
13. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: SIGMOD Conference, pp. 1–12 (2007)
14. Bernstein, P.A., Rahm, E.: Data warehouse scenarios for model management. In: ER, pp. 1–15 (2000)
15. Bowers, S., Delcambre, L.M.L.: The Uni-Level Description: A uniform framework for representing information in multiple data models. In: ER Conference. LNCS, vol. 2813, pp. 45–58. Springer, Berlin (2003)
16. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kauffman, San Mateo (2003)
17. Chen, P.: The entity-relationship model: Toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976)
18. Cluet, S., Delobel, C., Siméon, J., Smaga, K.: Your mediators need data conversion! In: SIGMOD Conference, pp. 177–188 (1998)
19. ElMasri, R., Navathe, S.: Fundamentals of Database Systems, 5th edn. Addison-Wesley, Reading (2007)
20. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
21. Fagin, R., Kolaitis, P.G., Popa, L.: Data exchange: getting to the core. ACM Trans. Database Syst. **30**(1), 174–210 (2005)
22. Florescu, D., Kossmann, D.: Storing and querying XML data using an rdmbs. IEEE Data Eng. Bull. **22**(3), 27–34 (1999)
23. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems: The Complete Book, 2nd edn. Prentice-Hall, Englewood Cliffs (2008)
24. Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M.: Clio grows up: from research prototype to industrial tool. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14–16, 2005, pp. 805–810. ACM, New York (2005)
25. Hull, R., King, R.: Semantic database modelling: Survey, applications and research issues. ACM Comput. Surv. **19**(3), 201–260 (1987)
26. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246 (2002)
27. Madhavan, J., Halevy, A.Y.: Composing mappings among data sources. In: VLDB, pp. 572–583 (2003)
28. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: A programming platform for generic model management. In: Halevy, A.Y., Ives, Z.G., Doan, A. (eds.) SIGMOD Conference, pp. 193–204. ACM, New York (2003)
29. Miller, R.J., Haas, L.M., Hernández, M.A.: Schema mapping as query discovery. In: VLDB, pp. 77–88 (2000)

30. Pottinger, R., Bernstein, P.A.: Merging models based on given correspondences. In: VLDB, pp. 826–873 (2003)
31. Poulovassilis, A., McBrien, P.: A general formal framework for schema transformation. Data Knowl. Eng. **28**(1), 47–71 (1998)
32. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
33. Ramakrishnan, R., Gehrke, J.: Database Management Systems, 3rd edn. McGraw-Hill, New York (2003)
34. Sheth, A., Larson, J.: Federated database systems for managing distributed database systems for production and use. ACM Comput. Surv. **22**(3), 183–236 (1990)
35. Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P., Lum, V.Y.: Express: A data extraction, processing, and restructuring system. ACM Trans. Database Syst. **2**(2), 134–174 (1977)
36. Tsichritzis, D., Lochovski, F.: Data Models. Prentice-Hall, Englewood Cliffs (1982)
37. Zaniolo, C. (ed.): Special issue on Data Transformations. Data Eng. **22**(1) (1999). IEEE Computer Society