# COMPLEX VALUES IN DATABASES

J.H. ter Bekke

Delft University of Technology
Department of Information Systems
P.O. Box 356, 2600 AJ Delft, The Netherlands
e-mail: j.h.terbekke@is.twi.tudelft.nl

## Abstract

Shortcomings of the relational data model have led to two different approaches for supporting complex values: the object oriented approach, in which they are modeled as a unit and represented as such in the database, and the semantic approach, in which complex values are derived from a simple conceptual structure. This paper contains an overview of the pros and cons of both approaches. An analysis indicates that generally derivation of complex values is superior to modeling and representation of complex values.

**Keywords**: data modeling, object oriented databases, semantic databases, user interfaces, view independence, data independence, knowledge representation.

## 1 Introduction

Classical computer programs use their own data structures. These structures are completely tailored for the program in question; they contain in fact the view required by the program. The approach offers certain advantages because representation and usage of data form a solid unit. This architecture has made it possible in the early days of computer science to develop several applications for large collections of data using limited hardware configurations. The disadvantages of structuring a particular view became clear when new applications needed new views on the same data. Existing tailored data structures were then felt as a disadvantage in stead of an advantage. The relational data model was the first to meet the new requirements by allowing only simple data structures suitable for many applications.

The relational data model has been introduced as a model based on simple mathematical concepts. These concepts make it possible to define application independent structures: tables or normalized relations. An advantage is that different views can be derived from the same data. Data must be gathered only once to use them subsequently for different applications. In spite of this, a number of disadvantages is adhered to the relational approach:

- the relational theory does not allow derivation of complex values, the model does not include operations for this purpose;
- the derivation of a different view is expensive which is caused by the necessary joins of tables.

These disadvantages are mainly felt in new application domains (often CAD/CAM, office automation, etc. are referred). Here, a tendency exists (partly for performance reasons) to restrict modeling to one particular view in which complex values may occur. Besides that, it seems appropriate to include modeling of behavior [1]. The relevant question is whether or not the existing disadvantages from the classical approaches are introduced again.

This paper presents an overview of the problems related to complex values in databases and contains an analysis of two alternatives:

- *modeling of complex values*
  Complex values are modeled and represented as such in the database. This is applied in the object oriented approach.
- *derivation of complex values*
  Complex values can be derived from the database which contains only simple data values. This is applied in the semantic approach.

The paper presents for both alternatives consequences concerning: conceptual modeling, user views, operations and integrity. It shows that modeling and representing complex values has certain disadvantages. Among them: limited structuring capabilities, loss of information and semantics, limited user interfaces and transitive dependencies. Advantages of deriving complex values are contrary to these disadvantages. They include: no logical restrictions in the derivation of views, no ambiguities, no loss of information and semantics, no transitive dependencies, updatable user views and intelligent user interfaces. A separation between access and storage of data makes the latter alternative also attractive when the aspect of performance is considered. This has been demonstrated by two implementations of these semantic concepts in a Unix environment [13, 20].

## 2 Modeling approaches

Early database approaches were based on hierarchies and networks. These approaches favored certain applications above others. Databases and their applications were strongly coupled. The relational data model caused a major change. Application and database were decoupled: a relational database is not limited useful for one single application, but can be used for a broad class of applications instead.

Relational databases use mathematical concepts: relations as subsets of Cartesian products. Relations support sets of similar elements; these elements are called tuples. The values occurring in tuples are always simple. Operations enable users to derive new relations from existing (stored) relations. Relational structures and operations lead to a closed system: derived relations satisfy the same requirements as existing relations. An advantage is that a user does not see any difference between original and derived relations. On the other hand, a disadvantage is that derived relations cannot contain complex values.

The absence of operations by which complex values can be derived has contributed to the impedance mismatch [3]. This has gained a lot of attention for the object oriented database approach in literature. Data structures do not necessarily contain only simple values. A value can be built up from a variable collection of simple data values. This enables us to create complex values.

Another approach is used in the semantic data model Xplain. Here type is the fundamental notion. A type is conceived as an aggregation of properties [13, 14]. A type can be considered as a property and offers capabilities of considering complex values. However, a complex value is in this approach still consisting of a definite number of (possibly large) components and not a variable number as offered by the object oriented approach.

Several studies have confirmed that the positive results of deriving complex values can be obtained in technical applications in which complex data structures play a role. Among these are models for: flexible production automation control systems [18], CAD frameworks [17], robotics [5], VLSI data management [19], user interfaces [22], product modeling for CAD/CAM [8] and application reuse-in-the-large [7]. Besides that at least two implementations of the semantic concepts (including the Nelsis framework [19, 20, 21] and the Xplain DBMS [13, 14]) have demonstrated the performance and flexibility of the proposed approach.

This paper presents an analysis of benefits and limitations of both alternatives for handling complex values. The analysis will be carried out using a simple example. The example has been deliberately selected to illustrate different aspects occurring often in complex practical situations. In an earlier paper [16] the same example has been used for a study on the flexibility of data models.

## 3 Design case

This paper uses a simple example about a club of compact disk player owners in which members can borrow compact disks from each other. We assume relevancy of the registration of ownerships, reservations and borrowings.

For registration purposes some data structures must be designed to fulfill the requirements. In the design will occur: compact disk title *CD* (for reservations), compact disk *copy* (for borrowings), *member*, *owner*, *borrower*, *borrowing* and *reservation* (possibly as part of a complex structure). Several views can be relevant for registration and retrieval purposes. These views must be allowed in accordance with the requirements. No logical restrictions may be imposed on data manipulation operations. In other words all operations, retrieve, insert, delete and update must be allowed on modeled/derived complex values. It is evident that under all circumstances the integrity of the database must be guaranteed.

## 4 Modeling complex values

**Conceptual modeling**

The objects: CD, copy, member, owner, borrower, borrowing and reservation play a dominant role in this case. There exists a single inheritance relationship between these user defined objects; both owner and borrower can be considered as subclasses of the member class. The inheritance graph looks at first instance as given in figure 1. All objects have here their own object identifier (OID denoted by ‣).

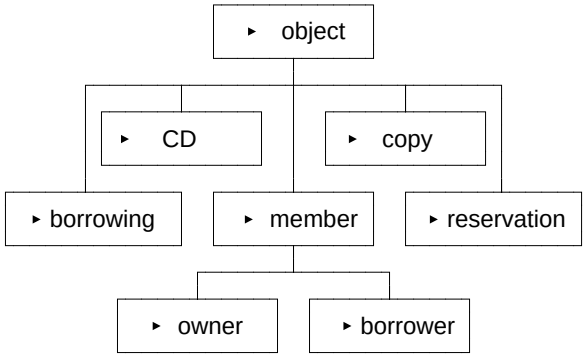


*Figure 1: Class hierarchy*

This hierarchy does not contain the relationship between CD and copy and the relationships between member, reservation and borrowing. Considering the latter, a 1:*n* relationship can be recognized between member and reservation and between member and borrowing. These relationships can be included by the introduction of complex values as presented in figure 2. From the structure can be concluded that the borrower class and the owner class do not play a role anymore. Reservations and borrowings are both part of the member class and do not have their own OID.

It is clear that complex values can handle only a limited number of 1:*n* relationships. The remaining objects are now: member, CD and copy. Two hierarchical relationships cannot be modeled using complex values, as will be explained in the following.

CD and copy have a 1:*n* relationship (because several copies can occur of a certain CD). In spite of this, it turns out to be impossible to represent this relationship using a complex value. The relationship interferes with the existing hierarchy in the member

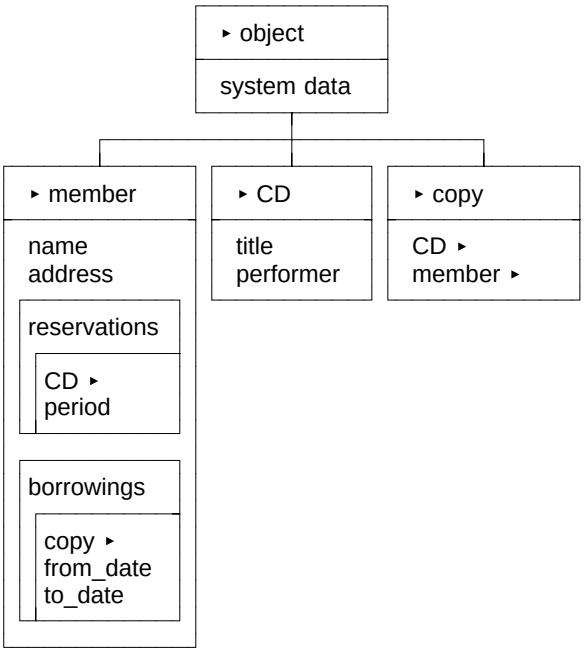


*Figure 2: Modeling of complex values*

class. It requires a cross-link between elements of the complex values for reservations and borrowings. The objects CD and copy must therefore be modeled as two independent classes; that is the only way to make both direct accessible from borrowings and reservations elements. This has consequences for the modeling of the second hierarchical relationship between copy and owner. It is evident that the owner class must be dropped to obtain a non-redundant representation.

**Modeling of user views**
A user view comparable with the complex structure is easy to realize. All other views will require much programming efforts. For instance, it is difficult to derive from this complex structure a view containing reservations per CD. This view requires for each individual CD access to all reservations from all members. The processing scheme does not correspond to the previously defined structure and will therefore lead to a complex algorithm that requires considerable processing time.

**Operations**
From the foregoing discussion it has become clear that complex values enable us to solve only a limited number of practical problems. For example to find an answer on 'who borrowed a certain copy?' is a hard problem under the given circumstances.

It is clear that all operations (in object oriented terminology: methods) are 'one record at a time'. The complex structure does not allow an associative query language with generic solutions. These low level operations can hardly be optimized because of their sequential nature. The programmer has to find his own way through the database (cf. [2]).

**Integrity**
Reservations and borrowings must be related to existing members. This requirement is automatically enforced by the complex structure. However, the constraint that borrowings must be related to existing copies from existing members (and not from the member in question) must be enforced in another way using methods [6]. These exceptions make the collection of integrity procedures difficult to handle.

It is clear that each (title, performer)-combination of CD occurs only once in the database. This is not true for complex values in the database. A complex value may lead to contradictions as will be shown in the following.

The current example allows identical collections of reservations to occur together with different members. Both identical complex values carry different semantics. So, they should not be considered as identical here. The same phenomenon occurs in Codasyl databases. It is known there under the term 'information bearing set'. This phenomenon received much attention in literature during the seventies.

## 5 Deriving complex values

**Conceptual modeling**
The semantic data model Xplain is based on three abstractions: classification, aggregation and generalization. They make use of the fundamental *type-attribute relationship*.

These abstractions are introduced informally in the following. An extensive treatment is given in [13, 14, 15]. The concepts may be considered as a further evolution of abstractions introduced by Smith and Smith [10, 11].

*Classification*
The real world is described by considering the properties of relevant objects, a property being defined as a fundamental notion, without assigning a value to it.

The abstraction leading to a property is called classification. The examples required for the recognition of a property are purely meant as an illustration. The property is *not* being defined hereby. Properties are represented by rectangles in diagrams, see figure 3. The counterpart of classification is called *instantiation*.

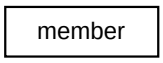


*Figure 3: Classification*

*Aggregation*
Aggregation is defined as the collection of a certain number of properties in a type, which in itself can be regarded as a new property (note the analogy with the mathematical set concept). A property from an aggregation is called an attribute of the type. The type should be named clearly and meaningfully.

Aggregation allows view independence: we can discuss the obtained type (possibly as a property) without referring to the underlying attributes. By applying this principle repeatedly, a hierarchy (in the sense of an acyclic network) of properties can be set up. An example is given in figure 4. Normally the hierarchy contains only aggregated types.

Aggregation is indicated by a line connecting the centers of two facing rectangle sides, while the aggregated type is placed above its attributes. Of course, aggregation also has its counterpart: the description of a type as a set of certain attributes is called *decomposition*. A type is defined by listing its attributes, so we can consider the following type definitions in our example:

*type* copy = CD, member.
*type* member = name, address.
*type* CD = title, performer.

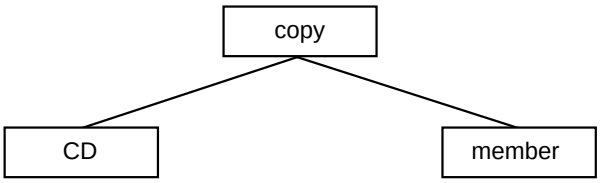


*Figure 4: Aggregation hierarchy*

Aggregation can be described using the verb *to have*. According to the above type definitions, a member has a name and an address. Identifications are denoted by the type names. This interpretation implies singular identifications.

*Generalization*
The third type of abstraction, important to conceptual models, is generalization; it is defined here as the recognition of similar attributes from various types and combining these in a new type (note the analogy with the intersection operation from mathematical set theory). The new type should, again, be named clearly and meaningfully. We can equally discuss the new type without mentioning underlying attributes, and it can in itself again serve as a property (i.e. allows view independence). Generalization's counterpart (i.e. the union of attributes from different types) is called *specialization*. Specializations are commonly associated with the verb *to be*. Several examples of generalization and specialization occur in [14].

These foregoing semantic concepts lead to the following uniform solution for the CD-club example (see also figure 5). All database objects have here their own singular identification:

*type* CD = title, performer.
*type* member = name, address.
*type* reservation = CD, member, period.
*type* copy = CD, member.
*type* borrowing = copy, member, from_date, to_date.

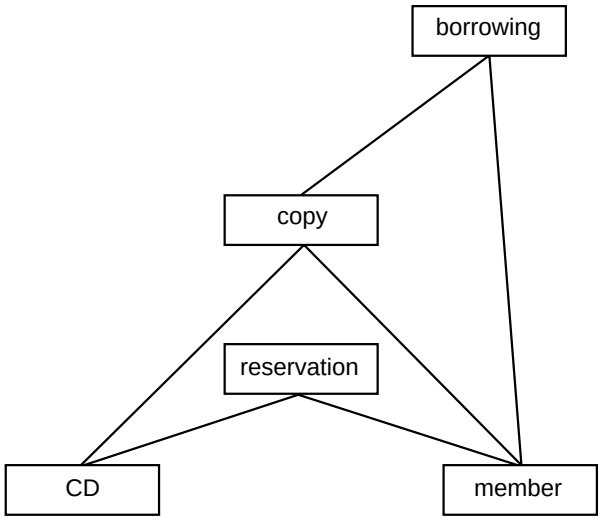


*Figure 5: Uniform semantic solution*

**Modeling of user views**
It is possible to derive numerous views from the foregoing simple conceptual structure. Figure 6 gives an example with corresponding operations as generated by the Xplain DBMS.

Different user views can be created easily in a declarative way (using the contents of a data dictionary) without programming efforts. These views are always updatable: operations that can be performed on a derived view are completely determined by relationships expressed in the user view. When these relationships can be mapped unambiguously on the conceptual structure, all modification operations are allowed (an example is the view in figure 6). Derived attribute values (e.g. 'reservation *its* member' and 'borrowing *its* member') can be added by the DBMS itself to obtain a valid database. Besides that, one may use certain simple views containing relevant details (for example a view for CD attributes in the view above). These simple views can be defined by a user or generated dynamically by the DBMS using the contents of a data dictionary.

Derivation of user views can be defined as above using a WYSIWYG interface but could also be formulated using operations from a data language. The view above would need specifications for two complex values; they can be derived from the conceptual structure by two extensions containing the necessary complex values (as a result of the *list* function [12]):

- *derivation of reservations per member:*
  *extend* member *with* reservations =
    *list* reservation *its* CD, period
    *per* member.

- *derivation of borrowings per member:*
  *extend* member *with* borrowings =
    *list* borrowing *its* copy, copy *its* CD *its* title,
      copy *its* member, from_date, to_date
    *per* member.

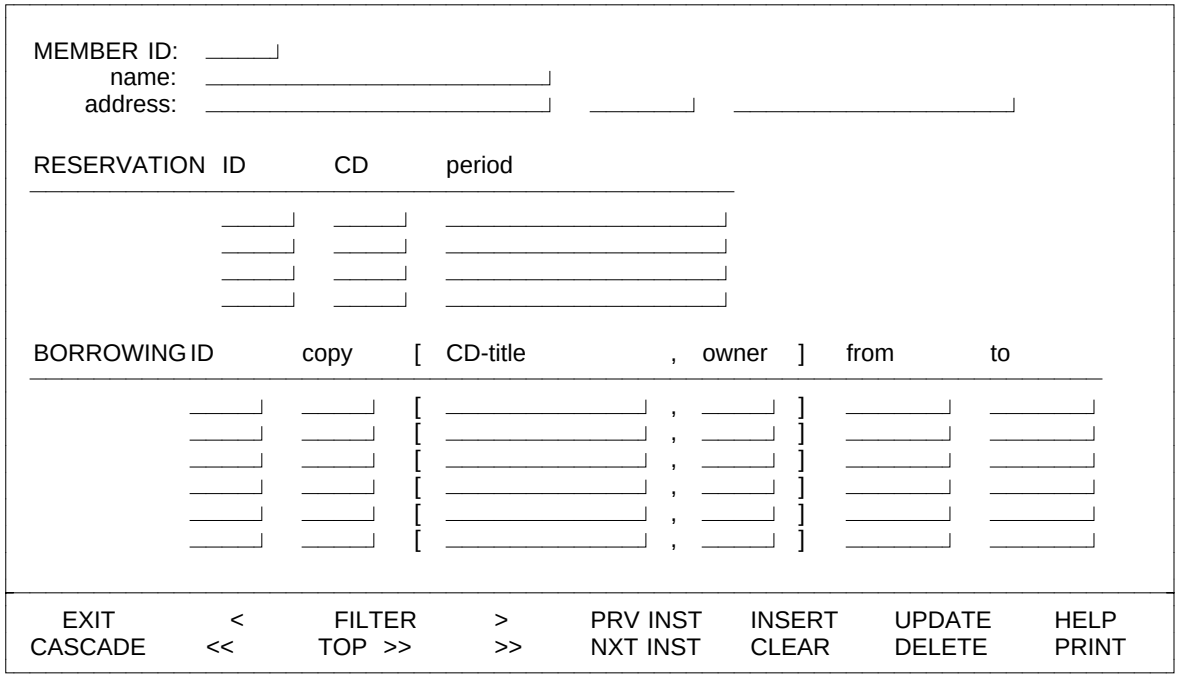


*Figure 6: Derived application view*

- *retrieval of the desired information:*
  *get* member *its* name, address, reservations, borrowings.

Other views can be derived similarly without any restrictions imposed on it. All views may contain several complex values derivable from the given conceptual structure. For example at least the following 16 composite views (including view 13 from figure 6) could be derived with comparable efforts:

**1** borrowings per borrower;
**2** borrowings per owner;
**3** borrowings per CD;
**4** borrowings per copy;
**5** reservations per CD;
**6** reservations per member;
**7** copies per owner;
**8** copies per CD;
**9** borrowings and copies per borrower;
**10** borrowings and copies per owner;
**11** borrowings and copies per CD;
**12** reservations and borrowings per CD;
**13** reservations and borrowings per member;
**14** copies and reservations per member;
**15** copies, reservations and borrowings per member;
**16** copies, reservations and borrowings per CD.

This example has shown that modeling and representation of only one particular view (as shown in the previous section) causes severe restrictions in practice. It creates enormous difficulties in deriving other equivalent user views.

The declarative knowledge representation from the semantic approach enables the DBMS to derive and add information whenever necessary to support complex user views. It results in an intelligent and flexible user interface. User views are not limited to only textual views; friendly graphical views are derivable with comparable efforts. This has been demonstrated with two implementations of the semantic concepts (i.e. the Nelsis framework [19, 20, 21] and the Xplain DBMS [13, 14]).

**Operations**

Extensions can be fruitful for query purposes. For querying they are generally combined with other set functions (for example: *count*, *total*, *max*, *min*, etc.). It enables us to derive allowable views and offers certain perspectives (they are illustrated with numerous examples in [14]):

- *generic query formulations*
  Extensions can be used for different purposes. An extension offers the user in fact a facility to use a certain derivation for several problems. It is particularly true when extensions are combined with terminology from the application domain.
- *reduction of complexity*
  Extensions reduce the complexity of a query significantly. An additional advantage is caused by the fact that all required extensions are completely determined by the conceptual structure. It leads to uniform query specifications. The query language contains therefore orthogonal components.
- *simple query optimization*
  It is evident that the proposed generic operations do not assume a particular processing sequence and could be executed in parallel. This facilitates optimization. An advantage is that query specifications never contain aliases (as known from SQL). Processing strategies are therefore not artificially limited. This aspect has been demonstrated with the efficiency of the Xplain system.

**Integrity**
A major advantage of the proposed semantic solution is the uniform specification of all integrity aspects. Relationships between objects require an inherent (i.e. structural) specification facility. Besides that, all explicitly defined static and dynamic constraints are equivalent to query language operations (see [14]).

## 6 Other considerations

There is a limited choice for the implementation of a data structure containing complex values. An existing file organization technique can only be useful at the highest level of an object; the resulting overhead is not acceptable for lower levels in the hierarchy (so not for individual complex values). Therefore elements from a complex value can only be accessed using (possibly linked) lists. Access to these elements is therefore sequential. It implies abundance of several efficient file organization techniques. Access and storage are strongly connected; there is unfortunately no data independence (see figure 7). Processing can only be efficient when the structure of the stored data agrees the structure of the view.

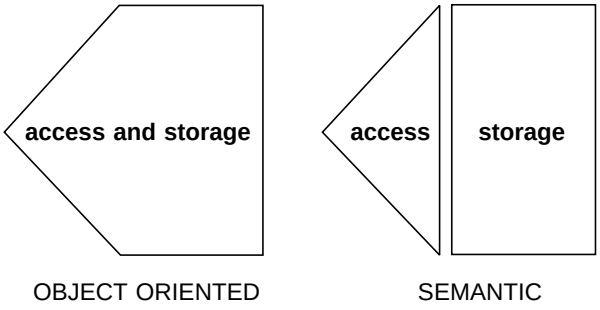


*Figure 7: Storage and access strategies*

Derivation of complex values offers the opportunity to use several modern file management techniques. Associative queries do not dictate processing to take place in a particular order, so processing structure and storage structure can be different in the semantic approach. The need for a certain data structure does not necessarily require the use of other related data structures. Transitivity is therefore prevented. However, the implementation based on the use of (visible) pointers, as follows from the object oriented approach, will not prevent transitivity [4, 9].

## Conclusion

This paper has presented and compared two approaches for handling complex values in databases: the object oriented approach and the semantic approach. Both approaches offer different facilities; the object oriented approach by modeling and representing complex values, while the semantic approach by deriving complex values from a simple conceptual structure.

From the analysis can be concluded that both alternatives offer different perspectives. The object oriented approach offers only a limited number of structuring alternatives, the consequences are: loss of structural semantics, limited and non-intelligent user interfaces. The approach is only useful in combination with at most one user view. The semantic approach offers: a uniform conceptual model, extensive structuring facilities for user views, intelligent/flexible user interfaces with updatable views, uniform view definition facilities and generic queries. The analysis has made it clear that the collection of meaningful user views is only restricted by the inherent conceptual structure of a database. This implies that generally the derivation of complex values is preferable. One of the major motivations for object oriented databases, as expressed in the first 'commandment' of the object oriented database system manifesto, is not always justified.

**Acknowledgments**
I wish to thank the many students at Delft University of Technology who graduated on subjects related to the Xplain project. My colleagues Bert Bakker and Dolf van der Ende are acknowledged for the frequent discussions on meta modeling and the support for user views.

## References

[1] M.P. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik, "The object oriented database system manifesto", Proceedings First Int. Conf. on Deductive and Object-Oriented Databases, (W. Kim, J-M. Nicolas and S. Nishio eds.), Elsevier, Amsterdam (1990), pp. 223-240.

[2] C.W. Bachman, "The programmer as navigator", Communications of the ACM 16 (1973) 1, pp. 653-658.

[3] F. Bancilhon and D. Maier, "Multilanguage object-oriented systems: new answer to old database problems?", Report Technique Altair 21-88.

[4] E. Bertino and L. Martino, "Object-oriented database management systems: concepts and issues", IEEE Computer 24 (1991) 4, pp. 33-47.

[5] N. Boneschanscher, Plan generation for flexible assembly systems, Dissertation Delft University of Technology, Delft (1993).

[6] The committee for advanced DBMS function, "Third generation database system manifesto", ACM SIGMOD Record 19 (1990) 3, pp. 31-44.

[7] H. Li, RITL: an information system for application reuse-in-the-large, Dissertation Delft University of Technology, Delft (1993).

[8] P. Martens, CAD/CAM for assembly planning, Dissertation Delft University of Technology, Delft (1993).

[9] J.P. Rosen, "What orientation should Ada objects take?", Communications of the ACM 35 (1992) 11, pp. 71-76.

[10] J.M. Smith and D.C.P. Smith, "Database abstractions: aggregation", Communications of the ACM 20 (1977) 6, pp. 405-413.
[11] J.M. Smith and D.C.P. Smith, "Database abstractions: aggregation and generalization", ACM Transactions on Database Systems 2 (1977) 2, pp. 105-133.
[12] J.H. ter Bekke, "A data manipulation language for relational data structures", Systems for large data bases, Proceedings 2nd Int. Conf. Very Large Data Bases (eds. P.C. Lockemann and E.J. Neuhold), North-Holland, Amsterdam (1976), pp. 159-168.
[13] J.H. ter Bekke, Semantic data modeling in relational environments, Dissertation Delft University of Technology, Delft (1991).
[14] J.H. ter Bekke, Semantic data modeling, Prentice Hall, Hemel Hempstead (1992).
[15] J.H. ter Bekke, Database design (3rd edition), Kluwer, Deventer (1993) (in Dutch).
[16] J.H. ter Bekke, "Object databases: structure and behavior", Delft University of Technology, Delft, Faculty of Technical Mathematics and Informatics (1993), Report 93-58.
[17] P. van den Hamer and M.A. Treffers, "A data flow based architecture for CAD frameworks", Proceedings ICCAD-90, IEEE (1990), pp. 482-485.
[18] P.R. van der Weerd, Unified design and implementation of flexible production automation control systems, Dissertation Delft University of Technology, Delft (1992).
[19] P. van der Wolf and T.G.R.M. van Leuken, "Object type oriented data modeling for VLSI data management", Proceedings 25th ACM/IEEE Design Automation Conference, ACM/IEEE (1988), pp. 351-356.
[20] P. van der Wolf, Architecture of an open and efficient CAD framework, Dissertation Delft University of Technology, Delft (1993).
[21] T.G.R.M. van Leuken, Data management for VLSI design in an open and distributed environment, Dissertation Delft University of Technology, Delft (1988).
[22] J.M. Versendaal, Separation of user interface and application, Dissertation Delft University of Technology, Delft (1991).