Serie Research Memoranda

On Data Models as Meta Models
An Application Designers Point of View

R.J. Veldwijk
E.R.K. Spoor
M. Boogaard
M.V. van Dijk

Research Memorandum 1991 - 24

vrije Universiteit  amsterdam
On Data Models as Meta Models
An Application Designers Point of View

R.J. Veldwijk
E.R.K. Spoor
M. Boogaard
M.V. van Dijk

Abstract
The purpose of this paper is to clarify the role of data models in designing data intensive applications. Data models like the relational model are used by application designers as an aid in creating application specific data models and are therefore meta models from a designer's point of view. It is asserted that the relationship between application designers and application users on the one hand is analogous to the relationship between designers of meta models and application designers on the other. It follows that the requirements of application users are essentially similar to those of application designers. Thus, just as communication with end users is crucial for data model design, communication with application designers is crucial for meta model design. It is contended that due to the lack of communication, the requirements of application designers are not fully met by the relational model. This model permits some constructs that have no real world counterpart and offers no modelling concepts for certain important constructs that do occur in reality. The paper amplifies this position by a number of examples, all based on publications by Codd and Date. Recommendations are made in order to identify shortcomings of meta models and to decide whether to extend the representational power of these models.

1 Introduction
The importance of capturing aspects of reality in data models has been accepted among scientists, DBMS-designers and application designers. Codd's relational model of data has triggered a great deal of research, development and general interest in data models and data modelling. In the field of application design, relational concepts, especially normalization theory, have greatly influenced the way in which databases are designed.

At first, the procedure was to collect information requirements, describe screen and print layouts in detail and then derive a data structure to reflect these requirements in a non-redundant manner. The bottom up normalization procedure by which the database structure was derived was considered mechanical.

In recent years this approach has been more or less abandoned. A top down, semantically oriented, design approach has proved to result in correct database designs in far less time. Therefore, database design now has a place of its own in the overall application design and its product is now generally called a 'data model' too. This shifting attitude is reflected in the publications of Date, see for instance [10] and compare [4] and [5].

Another development has been the recognition of the importance of database constraints in database design. It appeared that many program independent aspects cannot be captured by the...
normalization procedure. Normalization decreases the number of constraints but does not eliminate them altogether. It is quite difficult to explain why a business rule like 'any employee works for at most one department' should be represented in a data model while a rule like 'any department employs at least one employee' should not. This position is also taken by Codd who asserts that constraints should be expressed in a relational language and that constraint enforcement is the responsibility of the DBMS [3, p.243].

The database design is an important tool for future application users because it provides them with a checklist of all rules the application will enforce. From their point of view it is crucial that the design faithfully reflects all relevant microworld aspects. A design that overly constrains the permitted database states results in a misused or unused and therefore unreliable application. A model that permits database states that have no counterpart in reality does not support its users properly and consequently leads to unreliability too.

Interestingly, the application designer is in the same position as the application user with respect to the model he needs to do his job with. He needs a data model that captures aspects of data structures in general. Such a view of data is provided by a number of data models, effectively meta models from the designer's point of view. By far the most influential of these models is Codd's relational model of data. Just as the designer of a data model determines what objects and rules are relevant for a group of application users, the designer of a meta model determines what objects and rules are relevant for a group of people who design data intensive applications. Table 1 provides some objects and rules relevant to a specific data model (an employee registration application) and a meta model (the relational model).

<table>
<thead>
<tr>
<th>Data model</th>
<th>Meta model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objects of interest:</strong></td>
<td><strong>Objects of interest:</strong></td>
</tr>
<tr>
<td>- Departments</td>
<td>- Relations</td>
</tr>
<tr>
<td>- Employees</td>
<td>- Domains</td>
</tr>
<tr>
<td>- Salary payments</td>
<td>- Attributes</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
<tr>
<td><strong>Rules:</strong></td>
<td><strong>Rules:</strong></td>
</tr>
<tr>
<td>- Employee works for at most one department</td>
<td>- Attribute has exactly one domain</td>
</tr>
<tr>
<td>- Department employs at least one employee</td>
<td>- Relation has at least one attribute</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

Table 1: Objects and rules in a data model and in a meta model.

If one accepts the analogy between data models and meta models, it follows that just like a data model must closely meet the requirements of an application user, so must a meta model meet the requirements of an application designer. The designer tries to ensure this by means of intensive communication with the user. In contrast communication between designers of meta models and their users is extremely one-sided. A justification for this is the emphasis meta model designers place on keeping their models formal and simple (see for instance [11, p.134]). Although formality and simplicity are highly desirable properties of meta models (and indeed data models) any model is useless if it is unable to capture reality. The demand for formal and simple meta models is necessary but insufficient. In the field of data model design the situation is entirely reverse in the sense that normally the end user view is dominant. As a consequence any lack of consistency in the users view leads to data models that are complex and hard to maintain. This state of affairs is complemented by the lack of representational power of the most widely used meta model, the relational model.
In the following sections we examine the relational model from this vantage point. The purposes
of this examination are (1) to identify some shortcomings of the relational model and (2) to
provide a basis for resolving current and future disputes about the expressive power of the
relational model. In section 2 we take a closer look at meta models as such. In section 3 the
process of capturing reality in a data model by use of a meta model is examined. In the next
three sections the conclusions of sections 2 and 3 are applied to aspects of the relational model.
In section 4 we criticize some design guidelines advocated by Codd and Date. In section 5 we
discuss some constructs allowed by the relational model that do not occur in reality. In section 6
we proceed to discuss some constructs that do occur in reality but cannot be represented by the
relational model. Finally, in section 7 recommendations are made for improving the identified
omissions of the relational model and other meta models.

2 Meta models
Brodie defines a meta model as 'a collection of mathematically well defined concepts that help
one to consider and express the static and dynamic properties of data intensive applications' [1].
Note that not only static (database) concepts but also dynamic (programming) concepts should
ideally be part of a meta model. The current sharp distinction between data and programs
unfortunately obstructs the achievement of this goal. We shall show that this observation holds
for the relational model too, in spite of Codds justified criticism of many other models that their
lack of manipulative operators makes them 'no more than new kinds of data structure or data
typing' [3 p.467].

Note also that Brodies definition stresses the need for a solid formal basis of any meta model
but does not explicitly state that such a model should represent the relevant properties of reality
faithfully. Nevertheless both objectives of data model design are widely accepted as is shown by
Codd's efforts to extend the semantic content of the relational model [2]. Moreover both Codd
and Date stress the semantic nature of relational constructs like relations, domains, keys,
etcetera (see for instance [3 p. VII and VIII] and [9]).

The emphasis on formality and consistency is clearly demonstrated by Dates interpretation
principle which states for any meta model that 'the (...) model in question must have a commonly
accepted (and useful) INTERPRETATION: that is its objects, integrity rules, and operators must
have some generally accepted correspondence to phenomena in the real world' [9 p.145].

Note however that the interpretation principle has nothing to say about the expressive power
of the model. For this reason we introduce the representation principle as a complementary
yardstick. This principle states for any meta model that it must offer constructs to represent all
real world phenomena generally considered significant by application designers. Even if this
constitutes a never ending task and requires intensive communication between the designers of
meta models and their users it should be an important and explicit aim of meta model design
just as it is in data model design.

We argue that the relational model does not fully conform to the interpretation principle in a
broader sense, by which we mean that although the model only uses constructs that have real
world counterparts it allows designers to devise data models that cannot have real world
counterparts. Moreover we argue that, because of the overemphasis on formality, the current
relational model doesn't adhere sufficiently to the representation principle. Before we present
our arguments we have to make clear what the costs of these alleged shortcomings are.

3
3 Data modelling and meta model support

In an abstract sense most meta models offer the application designer a view of the world in terms of objects, constraints on objects and operations on objects. In conceiving a data model it is obviously crucial to decide what real world phenomena are important to represent as objects and what high level operations on these phenomena should be supported. Apart from this it is also important to decide to what rules the contents of the model must conform. Such decisions are equally important as the decision what phenomena should be modelled.

A meta model provides data model designers with inherent, explicit and implicit constraints [1]. Inherent constraints are rules that can never be violated in the meta model. In the relational model examples of such rules (or metarules) are 'tuples must be unique within a relation' and 'every relation has at least one attribute'. Explicit constraints are rules that can be defined by using some combination of mechanisms provided by the meta model. In a relational environment an example of such a rule is the assertion that 'no two tuples in the relation DEPARTMENT have identical values for the attribute DEPT#'. Implicit constraints are rules that are implied by other rules. Consider for instance the rule 'for every value of the attribute DEPT# in DEPARTMENT there exists at most one value of the attribute DEPTNAME in DEPARTMENT'. This rule is implied by the explicit constraint given above in combination with the inherent constraint that 'attributes are atomic'.

These concepts provide a framework to appraise the design decisions of competent data model designers. Their strategy in data model design seems to be directed at minimizing the number of explicit constraints in a data model. To put it another way, data model design aims at expressing as many rules as possible in terms of inherent constraints provided by the meta model. It is obvious that if the set of inherent constraints the meta model supports is expressive the data model designer will need relatively few explicit constraints.

The normalization procedure is an excellent example of this strategy. If a rule holds that attribute B is functionally dependent on attribute A, the relational model makes it possible to express this in an implicit manner by applying the inherent constraints which assert that 'attributes are atomic' and that 'no two tuples in a relation have identical primary key values'. Together these constraints imply the functional dependency of attribute B on attribute A.

Of course there is much more to data model design than normalization. There are numerous possibilities to conceive fully normalized wrong data models. The general strategy directed at minimizing the number of explicit constraints provides a framework to distinguish between good and bad data models on the one hand and expressive and inexpressive meta models on the other.

The rationale for this strategy must still be explained. On a low level of abstraction the reward for designing good data models is decreased programming effort and improved maintainability of application programs. On a high level of abstraction the reward for good design is improved understanding of the application in general because constructs in the real world (like classes of objects) can easily be mapped onto constructs in the relational model (like relations). In other words, a good data model applies both the interpretation and the representation principle. Hence, these principles are as relevant with regard to data models as they are to meta models.
4 Some criticisms of proposed design guidelines
The preceding sections provide (among others) a frame of reference for assessing guidelines for
data model design that are advocated by authorities like Codd and Date. It appears that several
of these are at odds with the described constraint minimization strategy. The guidelines to be
discussed are concerned with normalization, composite keys, the assessment of alternative
relational meta models and the classification of explicit constraints.

4.1 Normalization
The guideline to decompose relations into at least 3NF is generally accepted among data model
designers. Unfortunately, some authorities in the field now display a different attitude towards
normalization. For instance Date is of the opinion that '... if a relationship that is currently many-
to-one might eventually become many-to-many ... then it would be better to represent it in a separate
table right away, in order to avoid future disruptive changes to the design' [10, p.439]. He suggests
that many-to-one relationships can be divided in two kinds: those that are inherently many-to-
one and those that are currently many to one but need not remain so. Figure 1 gives an
example of a possibly inherent many-to-one relationship.

<table>
<thead>
<tr>
<th>INHERENT MANY-TO-ONE</th>
<th>NOT INHERENT MANY-TO-ONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE(EMP#, DEPT#,)</td>
<td>EMPLOYEE(EMP#, )</td>
</tr>
<tr>
<td>ASSIGNMENT(EMP#, DEPT#)</td>
<td>DEPARTMENT(DEPT#)</td>
</tr>
</tbody>
</table>

Figure 1: Any employee works for exactly one department.

If the data model designer chooses to treat the many-to-one relationship as not inherent he has
to introduce one extra relation, one extra attribute, two extra keys and two rules expressing in a
relational language that 'any EMPLOYEE-tuple must by referenced by at least one
ASSIGNMENT-tuple' and that 'no two tuples in ASSIGNMENT have the same value for
EMP#'.
Beside making the obvious point that this design criterion is a very soft one in practice it is
clear that this approach does not lead to a minimal number of explicit constraints. Since Date
also supports this objective [12, p.212], the advice not to always normalize all the way is not only
impractical but also questionable on theoretical grounds, using arguments Date himself agrees
with. If one takes the constraint minimization strategy seriously it appears that Dates guideline is
relevant for discussing the relational model, not for discussing the data modelling process. If
instability in the relationships between classes of objects is a normal phenomenon in the real
world it follows that the relational model does not adequately support the representation
principle and should therefore be extended or improved\(^2\). We feel that the process of data
model design is fuzzy enough as it is and that following Dates guideline generally doesn't
improve matters.

---
4.2 Composite keys and surrogates

The relational model permits the use of composite keys. Date lists a number of arguments against the use of such keys [11]. Again his arguments include the possibility of future changes in the data model design. Although it is good practice to minimize the use of composite keys it is not wise to avoid them completely because it leads to unnatural data models and an increased number of explicit constraints. Table 2 gives an example of a data model containing information about companies and their yearly balance sheets.

<table>
<thead>
<tr>
<th>COMPOSITE KEYS</th>
<th>NON COMPOSITE KEYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPANY(COMP#, NAME, .......)</td>
<td>COMPANY(COMP#, NAME, .......)</td>
</tr>
<tr>
<td>RD Royal Dutch</td>
<td>RD Royal Dutch</td>
</tr>
<tr>
<td>UNL Unilever</td>
<td>UNL Unilever</td>
</tr>
<tr>
<td>BALANCE(COMP#, YEAR, DATE APPROVED)</td>
<td>BALANCE(BAL#, COMP#, YEAR, DATE APPROVED)</td>
</tr>
<tr>
<td>BALITEM(COMP#, YEAR, ITEM, AMOUNT)</td>
<td>BALITEM(BAL#, BAL#, ITEM, AMOUNT)</td>
</tr>
<tr>
<td>RD 1989 LAND 2500</td>
<td>3531 567 LAND 2500</td>
</tr>
<tr>
<td>RD 1989 DEBT 250</td>
<td>3532 567 DEBT 250</td>
</tr>
<tr>
<td>RD 1989 CRED 300</td>
<td>3533 567 CRED 300</td>
</tr>
<tr>
<td>RD 1989 EQTY 4000</td>
<td>3534 567 EQTY 4000</td>
</tr>
</tbody>
</table>

Table 2: Companies, yearly balances and balance items

The composite key alternative is intuitively much more appealing and intuition is right if the alternatives are judged by the constraint minimization objective. In this case using noncomposite keys leads to the introduction of two attributes lacking natural interpretation together with the introduction of two alternate keys: COMP#, YEAR in BALANCE and BAL#, ITEM in BALITEM. Although the relational model supports alternate keys (i.e. explicit constraints) the result is an unnecessarily complicated data model leading to unnecessarily complex application programs.

Another related argument supplied by Date is that composite keys lead to 'logical redundancy'. In the example in table 2 the fact that company Royal Dutch produced a balance in 1989 is represented many times, once in the table BALANCE and many times in the table BALITEM. Note that this logical redundancy also occurs in the noncomposite key solution, although on a more limited scale. This form of redundancy, if redundancy it is, never leads to consistency problems. The reason is that whenever consistency is violated a referential integrity constraint is violated too. Redundancy in the traditional sense always leads to the introduction of explicit constraints. Logical redundancy leads to the introduction of an implicit constraint (see section 3) and thus doesn't complicate the data model design. The advantage of redundancy, easier retrieval, does apply to logical redundancy too. Dates logical redundancy argument thus justifies the use of composite keys in certain cases.

A third consequence of always using noncomposite keys is the introduction of meaningless attributes. These attributes are effectively analogous to the surrogates Codd introduced in his RM/T paper [2]. From the preceding discussion we conclude that the introduction of meaningless attributes doesn't pave the way for capturing more meaning in our data models and
that data model designers will therefore prefer the present version of relational model.

4.3 Alternative relational meta models

Alternative relational meta models like the binary relational and the irreducible relational model can also be assessed from a constraint minimization point of view [6 ch. 5]. These models reject the use of n-tuples to couple attributes in a one-to-one fashion. In stead of using the inherent constraints of the relational model these models use explicit constraints. Figure 2 compares the relational and the binary approach for a relation representing employees.

<table>
<thead>
<tr>
<th>RELATIONAL APPROACH</th>
<th>BINARY RELATIONAL APPROACH</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE(EMP#, DEPT#, NAME, SALARY)</td>
<td>EMP_DEPT(EMP#, DEPT#)</td>
</tr>
<tr>
<td></td>
<td>EMP_NAME(EMP#, NAME)</td>
</tr>
<tr>
<td></td>
<td>EMP_SLRY(EMP#, SALARY)</td>
</tr>
</tbody>
</table>

Figure 2: Representing employees

The connection between the various attributes constituting the original EMPLOYEE-relation is now established using explicit constraints of the referential integrity type. More precisely, two explicit constraints have to be defined asserting that any EMP_NAME tuple is referenced by a tuple in EMP_DEPT and by a tuple in EMP_SLRY. From a constraint minimization point of view the traditional relational approach seems preferable.

4.4 Ad hoc versus generalized explicit constraints

In the preceding three subsections we have tried to demonstrate that minimizing the number of explicit constraints using the inherent constraints of the meta model is a sensible thing to do. An explicit constraint is a constraint that is not inherent to the meta model and that is not enforced by other constraints. The meta model must provide the data model designer with the means to express these constraints. In the relational model this can be done using a relational language like relational calculus or SQL. It is clear that while explicit constraints can be of any degree of complexity the majority of explicit constraints fall in just a few categories. The best example of such a class of constraints is referential integrity. Referential integrity is generally considered so important that a data model design in which referential integrity rules are not specified is considered unacceptable. Referential integrity is not considered just another explicit constraint. It is interesting to see that Codd and Date have different views about whether to classify commonly occurring explicit constraints. Codd takes the position that constraints should not be casted in the data structure but should instead be expressed linguistically [3 p. 244]. Date takes the position that while it must be possible to specify any constraint in a relational language, identifying generalized constraints is highly desirable for certain commonly occurring cases [12 p. 208]. We agree with Date for a number of reasons.

First, as described above the separation between inherent constraints and generalized explicit constraints is quite fuzzy. Just as inherent constraints are preferable to explicit constraints, generalized explicit constraints are preferable over other one-of-a-kind assertions. While

---

3 Date uses the misleading term 'special case constraint'.

7
referential integrity is a very important generalized explicit constraint there may be other classes of constraints that deserve attention.

Second, ad hoc constraints are easily overlooked in the design process. If not, they have to be coded over and over again, leaving a lot of room for errors.

Third, we feel that expressing constraints in a relational language is insufficient if one wants to conceive sophisticated RDBMSs or applications. The reason for this is that constraints express a great deal of the semantics of the data model. Sophisticated systems must be able to access this information in order to display smart or flexible behaviour. Representing constraints by means of constraint classes, each having a specific meaning is important to prevent RDBMSs from becoming nothing but complex trigger mechanisms [11 p.127].

5 The relational model and the interpretation principle
As we have seen, the framework sketched in sections 2 and 3 is useful for deciding between design alternatives permitted by the relational meta model. In this section we argue that the relational model supports some constructs that have no counterpart in reality and thus does not fully conform to the interpretation principle. This shortcoming may result in bad data models and in the introduction of additional explicit constraints. We shall substantiate our position by discussing the relevance of multiple target relations and the treatment of self referencing relations.

5.1 Multiple target relations
Codd's definition of the relational model requires that a foreign key must reference a tuple in some relation, not necessarily in one specific relation. The possibility of multiple targets occurs whenever two or more primary keys are defined on the same domain. Date explicitly disagrees with Codd because he feels that it complicates the relational model and because it is hard to come up with a realistic example where such a facility is useful [11]. We take the somewhat stronger position that a data model in which a foreign key references tuples in more than one relation is always proof of clumsy data model design.

In his latest book Codd presents two examples in which multiple target relations occur [3 p.25]. In the first example a SUPPLIERS-relation is split up horizontally separating domestic suppliers from foreign suppliers. Figure 3 elaborates this example.

<table>
<thead>
<tr>
<th>SINGLE TARGET RELATION</th>
<th>MULTIPLE TARGET RELATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMEST SUPPL(S#, STATE, CITY)</td>
<td>DOMEST SUPPL(S#, NAME, STATE, CITY)</td>
</tr>
<tr>
<td>FOREIGN SUPPL(S#, COUNTRY)</td>
<td></td>
</tr>
<tr>
<td>SUPPLIER(S#, NAME)</td>
<td></td>
</tr>
<tr>
<td>INVOICE(INV#, S#, .......)</td>
<td></td>
</tr>
<tr>
<td>FOREIGN SUPPL(S#, NAME, COUNTRY)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Single versus multiple target relations: generalization.
We argue with Date that the single target solution is a much cleaner one. In any case it does explicitly distinguish between classes and subclasses and avoids ad hoc explicit constraints enforcing that domestic and foreign suppliers must have different values for the S#-attribute.

Codd's second example deals with a relation that is split up in two relations with the same primary key for performance reasons (see figure 4).

<table>
<thead>
<tr>
<th>ONE RELATION</th>
<th>TWO RELATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPPLIER(S#, NAME, STATE, CITY)</td>
<td>SUP_1(S#, NAME)</td>
</tr>
<tr>
<td>INVOICE(INV#, S#, ....)</td>
<td>SUP_2(S#, STATE, CITY)</td>
</tr>
<tr>
<td>INVOICE(INV#, S#, ....)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Single versus multiple target relations; performance optimization.

A simple solution to this problem would be to arbitrarily define one relation as referencing the other. All other relations referencing the original relation can then retain a single target relation. However we feel that performance oriented activities such as this should take place below the relational level together with the definition of constructs like indexes and clusters. If not Codd's proposition that the ANSI term 'conceptual schema' corresponds to the set of base relations does not hold [3 p.34]. Splitting up a relation for performance reasons definitely takes place below the conceptual level. Consequently, some base tables should be excluded from the conceptual schema and a view constituting the original relation should be included. Again the conclusion must be that there is no need for multiple target tables. Unless someone comes up with a realistic example the possibility to define foreign keys having multiple targets should be excluded from the relational model⁴. Even if it is possible to conceive a non contrived example it is questionable if the relational model should support a construct that rarely occurs and creates numerous opportunities for bad data model design.

5.2 Self referencing relations
The relational model permits foreign key references within a single relation. Such self referencing relations often occur in practice, especially when data model designers attempt to create more generalized data models. Figure 5 gives an example of a relation representing employees and their managers.

<table>
<thead>
<tr>
<th>EMPLOYEE(EMP#, EMPNAME, EMP# MGR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Clark 2</td>
</tr>
<tr>
<td>2 Scott 3</td>
</tr>
<tr>
<td>3 Barker 4</td>
</tr>
<tr>
<td>4 White 2</td>
</tr>
<tr>
<td>5 Blake 4</td>
</tr>
</tbody>
</table>

Figure 5: A self referencing relation

It seems that the constraint expressing that 'cycles must never occur' always holds. We have

---

⁴ In fact we cannot think of a realistic example in which two or more relations have primary key attributes defined on the same domain in which none of those attributes is part of a foreign key.
never come across an example in which this constraint does not apply. Yet every time a self referencing relation occurs it is up to the designer to identify and describe the constraint and see to it that it is incorporated as screening routines in the application programs. The question is whether there are realistic examples in which the constraint does not apply. If not, an argument can be made for incorporating it in the relational model, just like referential integrity is incorporated.

An argument against extending the relational model can be based on the fact that observations like this rely heavily on induction. Yet suppose the constraint applies almost always there is still an argument for incorporating it in the relational model on the basis of the representation principle. In the rare situation in which the constraint does not apply the only consequence is that instead of applying a referential integrity constraint the designer must introduce an ad hoc constraint. If this is unacceptable so is the current situation in which an ad hoc constraint occur almost always.

Of course there is a compromise between repeated ad hoc solutions and the extension of the relational model. This compromise is the declaration of a generalized constraint as described in section 4.4. We conclude that the problems described in this section warrant a serious discussion.

6 The relational model and the representation principle

In the previous section we have tried to show that the relational model underconstrains its users, resulting at best in unnecessary work for both designers and programmers. Unfortunately there are also situations in which the relational model overconstrains its users. This leads to the now familiar problem of having to introduce explicit constraints. Examples of this occur with regard to object representation, object identification and integration of data and functional aspects.

6.1 Object representation

In the relational model individual real world objects are represented by tuples. A tuple belongs to one relation and every relation represents a distinct object class. Problems arise whenever an object does not fit precisely into a relation. This occurs among others in situations in which generalization or super/subtyping plays a role. Figure 6 shows an example in which certain individual employees are also managers and therefore also belong to another object class.

<table>
<thead>
<tr>
<th>EMPLOYEES IN SUPERCLASS</th>
<th>EMPLOYEES IN SUBCLASSES</th>
<th>EMPLOYEES IN BOTH CLASSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP(EMP#, NAME, MGR, BONUS)</td>
<td>WHCOL(MGR#, NAME, BONUS)</td>
<td>EMP(EMP#, NAME, MGR, BONUS)</td>
</tr>
<tr>
<td>1 Clark NO</td>
<td>2 Scott 2500</td>
<td></td>
</tr>
<tr>
<td>2 Scott YES 2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Barker YES 5500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 White NO</td>
<td>5 Blake</td>
<td></td>
</tr>
<tr>
<td>1 Clark</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 White</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Blake</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: Classes and subclasses

None of the design alternatives shown in figure 6 is very satisfying. If the manager subclass is

Note that in contrast to incorporating referential integrity this does not require extra design effort. Referential integrity requires explicit specification if a fully relational DBMS is to recognize it. It is not always possible to deduce referential integrity from the domain specifications even when multiple targets are prohibited.
neglected explicit constraints are needed to the effect that only managers can have bonuses. If the employee superclass is neglected the problems described in section 5.1 occur. If both classes are modelled objects lose their one-to-one correspondence to tuples and constraints must be specified to the effect that superclass tuples must be referenced by subclass tuples depending on some condition. Depending on specific circumstances, like the nature of the programs operating on these data structures, every solution can be the best because none of the alternatives captures the structure of classes and subclasses.

We feel that the problems of missing information, generalization and the representation of historic information are different aspects of a general problem with regard to the representational power of the relational model and that extensions of this model or alternative meta models should aim at the solution of these problems.

6.2 Object Identification

The relational model requires that every base relation has exactly one primary key by which any tuple within the relation can be identified. The justification is that a real world object represented in the data model can now always be identified by means of its relation name together with a set of values for its primary key attributes.

Difficulties occur whenever one encounters classes that contain at most one object because in this case the relation name is sufficient identification. The relational model requires the data model designer to arbitrarily specify a primary key and to express an explicit constraint to the effect that this relation must not contain more than one tuple. As argued by Warden it would be a good idea to allow the primary key of any such relation to be the empty set [14].

Representation difficulties may also occur when a relation has multiple candidate keys. This is caused by the rule that foreign keys must reference primary keys, never alternate keys. The justification is that allowing foreign keys to reference alternate keys adds complexity, not representational power [11 p.135]. However, this argument does not hold in the situation in which alternate keys designate different statuses of objects and references to these objects are made depending on their status. Consider for example figure 7 which shows a part of a data model for an order entry application. It is assumed that orders received are at some time delivered and an invoice is sent to the customer. The customer pays for the delivery later, possibly in several instalments. Orders received and invoices sent must be consecutively numbered.

If the foreign key in PAYMENT references the primary key in ORD the data model designer has to introduce an explicit constraint to the effect that ORD-tuples with a null value for INV# cannot be referenced by PAYMENT-tuples. If PAYMENT were to reference the alternate key in ORD the semantics of the situation would be better captured because the existence of the alternate key expresses the fact that the order has a status in which payments are possible.

<table>
<thead>
<tr>
<th>FOREIGN KEY REFERENCES PRIMARY KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORD(ORD#, CLIENT, ODATE, AMOUNT, INV#, IDATE)</td>
</tr>
<tr>
<td>PAYMENT(ORD#, SEQ#, PAYDATE, AMOUNT)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOREIGN KEY REFERENCES ALTERNATE KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORD(ORD#, CLIENT, ODATE, AMOUNT, INV#, IDATE)</td>
</tr>
<tr>
<td>PAYMENT(INV#, SEQ#, PAYDATE, AMOUNT)</td>
</tr>
</tbody>
</table>

Figure 7: Orders, invoices and payments
There is obviously a trade-off between representational power on the one hand and simplicity and consistency on the other. What is needed is a discussion on the basis of examples such as the ones in figures 6 and 7 to determine the price we pay for simplicity or representational power.

6.3 Integration of data and functional aspects
As we have seen in section 2 not only static (database) concepts but also dynamic (programming) concepts should ideally be part of a meta model. Meta models typically describe static concepts only. This holds for the relational model too although this model does provide its users with operators for data manipulation. The model contains concepts like 'relation', 'attribute' and 'tuple' but lacks concepts like 'program' or 'transaction'. Support for such concepts is left to RDBMS vendors or to data model designers.
This omission may pose problems whenever extensions to the relational model are suggested. Take for example the discussion about the extension of the relational model with the foreign key rules 'Cascade', 'Delete' and 'Nullify' [11]. The introduction of such rules in RDBMSs would greatly aid database designers and application programmers but it would be even better if these rules could be specified (or overruled) per application program or even per transaction. For example, it is perfectly feasible for one program to reject an attempt to delete a CLIENT-tuple referenced by one or more ORDER-tuples and for another not to reject such an operation.

7 Conclusions and recommendations
The three preceding sections demonstrate that the relational model in many respects fails to support the interpretation and representation principles introduced in section 2. The result of this failure is always the introduction of ad hoc explicit constraints. By their nature the meaning such constraints add to a data model is not accessible by a DBMS or an application program.
One possible way to tackle this problem is to introduce more generalized constraints like referential integrity and to extend the relational model to support these. Because there is a trade off between expressive power and formal elegance choices will have to be made. One way to find out what types of constraint occur frequently is to start up empirical research with regard to existing data model design. In any case the inherent fuzziness of such extensions requires communication between meta model designers and the database designers community.
Another recommendation would be to express meta models in their own terms. It is perfectly feasible to express the relational model, at least the structural and integrity parts, in relational terms. As demonstrated in table 1 the objects of interest to the relational meta model do not differ significantly from those of a relational data model.
In addition it would be a good idea to refrain from using unrealistic examples when discussing meta models. If it is impossible to find realistic examples to support an argument the argument is probably worthless.

A question that remains is whether it is wise to extend the representational power of the relational model on informal grounds. It may be a far better idea to use the formal relational model as a basis for higher level meta models that are semantically more expressive. In either case the problems discussed in this paper will have to be addressed.


[14] Warden A, Table_Dee and Table_Dum, in [8]