1991

ON THE EXPRESSIVE POWER OF THE RELATIONAL MODEL: A DATABASE DESIGNER'S POINT OF VIEW

R. J. Veldwijk
Vrije Universeteit Amsterdam

E. R. K. Spoor
Vrije Universeteit Amsterdam

M. Boogaard
Vrije Universeteit Amsterdam

M. V. van Dijk
Vrije Universeteit Amsterdam

ON THE EXPRESSIVE POWER OF THE RELATIONAL MODEL: A DATABASE DESIGNER'S POINT OF VIEW

R. J. Veldwijk
E. R. K. Spoor
M. Boogaard
M. V. van Dijk
MESDAG Research Group
Vrije Universeteit Amsterdam

ABSTRACT

The purpose of this paper is to introduce a framework for assessing the expressive power of data models and to apply this framework to the relational model of data. From a designer's point of view, a data model such as the relational model should not only be formally defined and easy to understand, but should also provide a powerful set of constructs to model real-world phenomena. The expressive power of a data model, defined as the degree to which its constructs match with constructs encountered in reality, can be judged by two complementary principles: the interpretation principle and the representation principle. It is asserted that database designers attempt to minimize the number of ad hoc database constraints, and that a data model faithful to the two principles supports this design strategy. Subsequently, this constraint minimization strategy is used to assess the expressive power of a particular data model, i.e., the relational data model. The authors take the position that the expressive power of the relational model is not optimal, due to a lack of adherence to both the interpretation principle and the representation principle. The paper amplifies this position by means of a number of examples, all based on publications by Codd and Date.

1. INTRODUCTION

The importance of capturing aspects of reality in data models is generally accepted among scientists and information systems designers. Codd's (1990) 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, notably normalization theory, have greatly influenced the way in which databases are designed.

Initially, 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 by means of a bottom up normalization procedure.

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, and the database design now has a place of its own in the overall application design. This shift in attitude is reflected in several publications by Date (see, for example, 1989, Chapter 19, and compare 1981, Chapter 14, with 1990a, Chapter 21).

Another development has been the recognition of the importance of database constraints in database design. Many program-independent aspects cannot be captured by the normalization procedure. It is quite difficult to explain why a business rule like "any employee works for at most one department" can be represented in a database design, while a rule like "any department employs at least one employee" cannot. This position is also taken by Codd (1990, p. 243), who asserts that constraints should be expressed in a relational language and that constraint enforcement is the responsibility of the DBMS, rather than the application programs.

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

The database design process is aided by a data model, which enables the designer to express the rules that apply to the user's environment. The expressive power of a data model is an important factor in the effective and efficient design and maintenance of applications. The data model's expressive power can be regarded as the degree to which its constructs match with the constructs that designers encounter in reality.

Paramount objectives of data models are formality and simplicity (see, e.g., Date 1990b, p. 134). However, from a database designer's point of view these objectives are necessary but insufficient. In their view, data models should emphasize expressiveness rather than formality and simplicity.

Although the preceding discussion can be applied to data models in general, the present paper focuses on one particular data model — the relational data model. This model has been chosen because it is highly popular in the field of database design and is dominated by formal considerations. The purpose of the paper is to introduce a framework for assessing the expressive power of data models and to apply this framework to the relational data model. In section 2, we take a closer look at data models as such, and identify two complementary principles by which the expressive power of a data model can be assessed. Section 3 examines the process of capturing reality in a database design using a data model and a database design strategy, directed at minimizing the number of application-specific constraints, is identified. Sections 4, 5 and 6 apply the constraint minimization strategy to aspects of the relational model. In section 4, we criticize certain design guidelines advocated by Codd and Date from a constraint-minimization viewpoint. In section 5, we discuss a number of constructs allowed by the relational model that do not occur in reality, and thus lead to cumbersome database design. In section 6, we discuss some constructs that do occur in reality, but cannot be represented elegantly by the relational model, again leading to cumbersome database design. Finally, in section 7, recommendations are made for remedying the identified shortcomings of the relational model. The central idea behind these recommendations is that research directed at extending the expressive power of a data model requires the research community to adopt not only a formal but also a database designer's orientation.

2. DATA MODELS

A data model is often regarded as a collection of concepts, well-defined by mathematics or formal logic, that help one to consider and express the static and dynamic properties of data-intensive applications. This definition stresses the need for a solid formal basis for any data model, but does not explicitly state that it should faithfully represent the relevant properties of reality. Nevertheless, both these objectives of database design are widely accepted, as is shown by Codd's (1979) efforts to extend the semantic content of the relational model. Moreover, both Codd and Date stress the semantic nature of relational constructs such as relations, domains, keys, etc. (Codd 1990, p. VII-VIII; Date 1989, Chapter 8).

The emphasis on formality and consistency is clearly demonstrated by Date's (1989, p. 145) **interpretation principle**, which states that "the data 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."

The interpretation principle, while useful, is not sufficient to judge the expressive power of a data model. For this reason we will introduce the **representation principle** as a complementary yardstick. This principle states for any data 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 and users of data models, it should be an important and explicit aim of data model design, as it is in database design.

We argue that the relational model does not fully conform to Date's interpretation principle in a broader sense, by which we mean that although the model uses only constructs that have real-world counterparts, it allows designers to devise database designs that cannot have real-world counterparts. Moreover, we argue that, because of the overemphasis on formality, the current relational model does not 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. DATABASE DESIGN AND DATA MODEL SUPPORT

In an abstract sense, most data models provide application designers with a view of the world in terms of objects, constraints on objects, and operations on objects. In designing a database, it is obviously crucial to decide which real-world phenomena are important enough to be represented as objects and which high-level operations on these objects should be supported. In addition, it is important to decide what constraints apply to the objects identified.

A data model provides database designers with inherent, explicit and implicit constraints (Brodie 1984). Inherent constraints are rules that can never be violated in the data model. In the relational model, examples of such rules (i.e., 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 using some combination of mechanisms provided by the data 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 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."

This constraint classification schema makes it possible to appraise the design decisions of competent database designers. Their design strategy seems to be directed at minimizing the number of explicit constraints. To put it another way, database design aims at expressing as many rules as possible in terms of inherent constraints provided by the data model. It is obvious that if the set of inherent constraints the data model supports is expressive, the database 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 constraint which asserts that "attributes are atomic," with the explicit constraint that "no two tuples in a relation have identical values for attribute A." Together these constraints imply the functional dependence of attribute B on attribute A.

Of course there is much more to database design than normalization. It is possible to conceive fully normalized but wrong databases. The general strategy directed at minimizing the number of explicit constraints provides a framework for distinguishing between good and bad database designs and expressive and inexpressive data models.

The rationale for this strategy must still be explained. At a low level of abstraction, the reward for designing good databases is decreased programming effort and improved maintainability of application programs. At a high level of abstraction, the reward for good design is improved understanding of the application in general, as constructs in the real world can easily be mapped onto constructs in the data model and vice versa. In other words, good database design applies both the interpretation principle and the representation principle. Hence, these principles are just as relevant to database designs as they are to data model design.

In the next three sections the framework presented above will be applied to the relational model. We shall demonstrate its applicability to discussions on relational database design guidelines (section 4) and to discussions related to constructs in the relational model itself (sections 5 and 6).

4. CONSTRAINT MINIMIZATION AND DATABASE DESIGN GUIDELINES

The preceding sections provide a frame of reference for assessing guidelines for database design advocated by authorities like Codd and Date. It appears that several of these are at odds with the constraint minimization strategy described above. The guidelines to be discussed are concerned with normalization, composite keys, and the classification of explicit constraints.

4.1 Normalization

The guideline to decompose relations into at least BCNF is generally accepted among database designers. Unfortunately, some authorities in the field now display a different attitude towards normalization. For instance, Date (1989, p. 439) 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." He suggests that many-to-one relationships are of 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 both kinds of relationships.

If the database designer chooses to view 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 be 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 in practice this design criterion is a very soft one, it is clear that this approach does not lead to a minimal number of explicit constraints. Since Date (1990b, p. 212) also supports this objective, the advice not to always normalize all the way is not only impractical but also questionable on theoretical grounds, on the basis of arguments Date himself

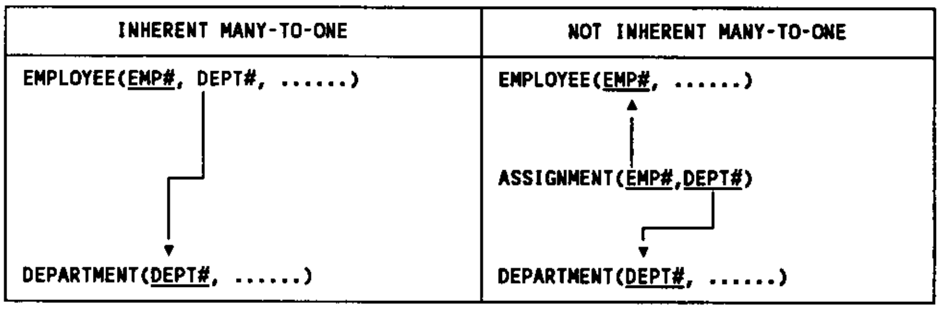


Figure 1. Any Employee Works for Exactly One Department

agrees with. If one takes the constraint minimization strategy seriously, it appears that Date's guideline is relevant for discussing the relational model, but not for discussing the database design process. If instability in the relationships between classes of objects is a normal phenomenon in reality, it follows that the relational model does not adequately support the representation principle and should therefore be extended or improved.[1] We feel that the process of database design is fuzzy enough as it is, and that following Date's guideline generally does not improve matters.

4.2 Composite Keys and Surrogates

The relational model permits the use of composite keys. Date (1990b, Chapters 5 and 6) is opposed to using such keys. Again his arguments include the possibility of future changes in the database design. Although it is good practice to minimize the use of composite keys, blindly following Date's advice leads to unnatural database designs and an increased number of explicit constraints. There are circumstances in which a composite key solution is the more expressive one. This is illustrated by Table 1, which gives an example of a database containing information about companies and their annual balance sheets.

The composite key alternative is intuitively much more appealing and intuition is right if the alternatives are judged by the constraint minimization objective. In some cases, noncomposite keys lead 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 database design 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 1, the fact that the Royal Dutch company 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 it is redundancy, never leads to consistency problems. The reason is that whenever consistency is violated, a referential integrity constraint is violated as well. 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 does not complicate the database design. The advantage of redundancy, easier retrieval, also applies to logical redundancy. Date's 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 (1979) introduced in his RM/T paper. Thus, on the basis of the preceding discussion, we conclude that the introduction of meaningless attributes does not provide opportunities to capture more meaning in our database designs, and that designers will therefore prefer the present version of the relational model.

4.3 Ad Hoc Versus Generalized Explicit Constraints

In the preceding two subsections, we have demonstrated that minimizing the number of explicit constraints using the inherent constraints of the data model is a sensible procedure. The data model must provide the database designer with the means to express these constraints. In the relational model, this can be done by means of a

COMPOSITE KEYS	NONCOMPOSITE KEYS
COMPANY(COMP#, NAME, ...)	COMPANY(COMP#, NAME, ...)
RD Royal Dutch	RD Royal dutch
UNL Unilever	UNL Unilever
... .	... .
BALANCE(COMP#, YEAR, DATE_APPROVED)	BALANCE(BAL#, COMP#, YEAR, DATE_APPROVED)
RD 1989 04/19/1990	567 RD 1989 04/19/1990
RD 1990 04/12/1991	568 RD 1990 04/12/1991
UNL 1990 03/28/1991	569 UNL 1990 03/28/1991
... .	... ...
BALITEM(COMP#, YEAR, ITEM, AMOUNT)	BALITEM(B_I#, BAL#, ITEM, AMOUNT)
RD 1989 LAND 2500	3531 567 LAND 2500
RD 1989 DEBT 250	3532 567 DEBT 250
RD 1989 CRED 300	3533 567 CRED 300
RD 1989 EQTY 4000	3534 567 EQTY 4000
... .	....

Table 1. Companies, Annual Balances and Balance Items

relational language, such as relational calculus or SQL. It is clear that while explicit constraints can be of any degree of complexity, the majority of explicit constraints fall into 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 database design in which referential integrity rules are not specified is considered unacceptable. Thus, a referential integrity constraint is not just another explicit constraint. Other examples are explicit constraints required to support the concept of "image domains" (see Smith and Smith 1977) and the quite common constraint category asserting that "any tuple in relation A must be referenced by at least one tuple in relation B."

It is interesting to note that Codd and Date differ about whether to classify the more common explicit constraints. Codd (1990, p. 244) takes the position that constraints should not be casted in the data structure, but should instead be expressed linguistically. Date (1990b, p. 208) takes the position that while it must be possible to specify any constraint in a relational language, identifying *generalized* constraints[2] is highly desirable for certain commonly occurring cases. We agree with Date for a number of reasons.

First, as we have seen above, the distinction between inherent constraints and generalized explicit constraints is quite fuzzy. Just as inherent constraints are preferable to explicit constraints, generalized explicit constraints are preferable to other one-of-a-kind assertions. While referential integrity is a very important generalized explicit constraint, there are 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 substantial 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 database design. Sophisticated systems must be able to access this information in order to display smart or flexible behavior. Representing constraints by means of constraint classes, each with its own specific meaning, is important in preventing RDBMSs from becoming nothing but complex trigger mechanisms (Date 1990b, p. 127).

5. CONSTRAINT MINIMIZATION 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 data model. In this section, we shall 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 poor database designs and 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

According to Codd's definition of the relational model, 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 (1990b, Chapters 5 and 6) 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 of a case where such a facility might be useful. We take the stronger position that a database in which a foreign key references tuples in more than one relation is *always* proof of poor design.

In his latest book, Codd (1990, p. 25) presents two examples in which multiple-target relations occur. In the first example a SUPPLIERS-relation is split up horizontally, separating domestic suppliers from foreign suppliers. Figure 2 elaborates this example.

We agree with Date that the single-target solution is a much cleaner one. In any case, it explicitly distinguishes 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 for performance reasons is decomposed into two relations with the same primary key (see Figure 3).

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 such as indexes and clusters. Otherwise, Codd's (1990, p. 34) proposition that the ANSI term "conceptual schema" corresponds to the set of base relations does not hold true. Decomposing a relation for performance reasons clearly 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 we must conclude 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 model.[3] Even if it is possible to conceive a non-contrived example, it is questionable whether the relational model should support a construct that rarely occurs and creates numerous opportunities for bad database design.

5.2 Self-Referencing Relations

The relational model permits foreign key references within a single relation. In practice, such self-referencing relations often occur, especially when database designers attempt to create more generalized database designs. Figure 4 gives an example of a relation representing employees and their managers.

It would seem that the constraint expressing that "cycles must never occur" always holds true. We have 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 into the application programs in the form of screening routines. The question is whether realistic examples in which the constraint does not apply exist. If not, an argument can be made for incorporating it into the relational model, in the same way referential integrity[4] is incorporated.

A possible argument against extending the relational model is based on the fact that observations like this rely heavily on induction. But suppose the constraint applies *almost* always; then there is still an argument for incorporating it into the relational model on the basis of the representation principle. In the rare instances 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 almost always occurs.

Of course there is a compromise possible 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 4.3. We believe that the problems described in this section warrant a serious discussion.

6. CONSTRAINT MINIMIZATION AND THE REPRESENTATION PRINCIPLE

In the previous section, we demonstrated that the relational model under-constrains its users, resulting at best in unnecessary work for both designers and programmers. Unfortunately, there are also situations in which the relational model over-constrains its users. This happens whenever a real-world object cannot be represented by means of one tuple, as in the case of generalization hierarchies, historical data, and missing data. A fundamental discussion of these aspects is beyond the scope of this paper. The problems we intend to discuss are concerned with object identification and the integration of data and functional aspects.

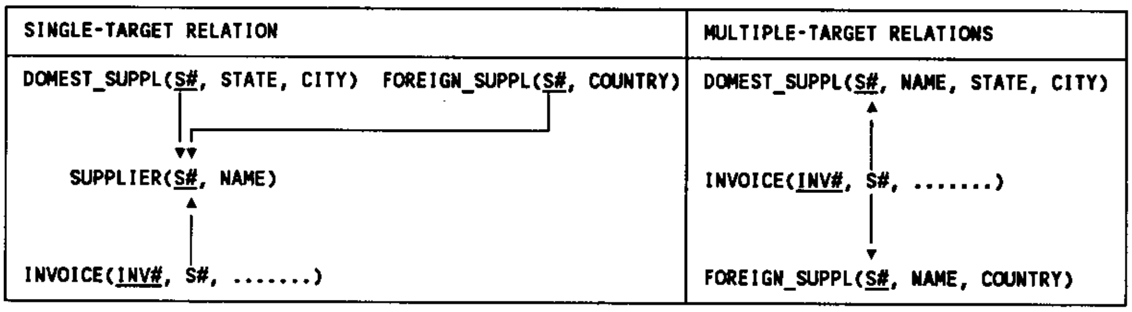

SINGLE-TARGET RELATION
DOMEST_SUPPL(S#, STATE, CITY)
FOREIGN_SUPPL(S#, COUNTRY)
SUPPLIER(S#, NAME)
INVOICE(INV#, S#, ...)
MULTIPLE-TARGET RELATIONS
DOMEST_SUPPL(S#, NAME, STATE, CITY)
INVOICE(INV#, S#, ...)
FOREIGN_SUPPL(S#, NAME, COUNTRY)

Figure 2. Single Versus Multiple-Target Relations: Generalization

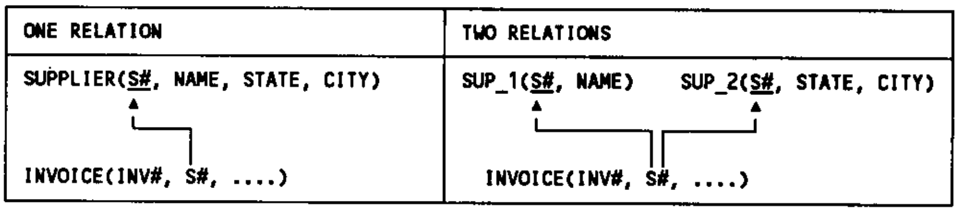
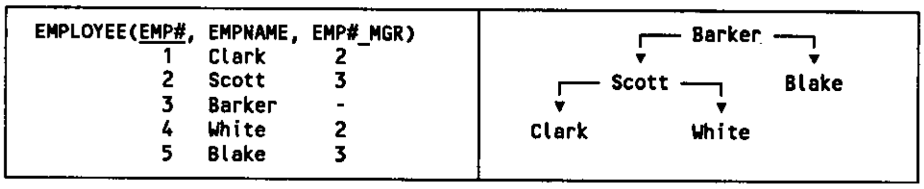


Figure 3. Single Versus Multiple-Target Relations: Performance Optimization



EMPLOYEE(EMP#,	EMPNAME,	EMP#_MGR)
1	Clark	2
2	Scott	3
3	Barker	-
4	White	2
5	Blake	3

Figure 4. A Self-Referencing Relation and Its Graphical Representation

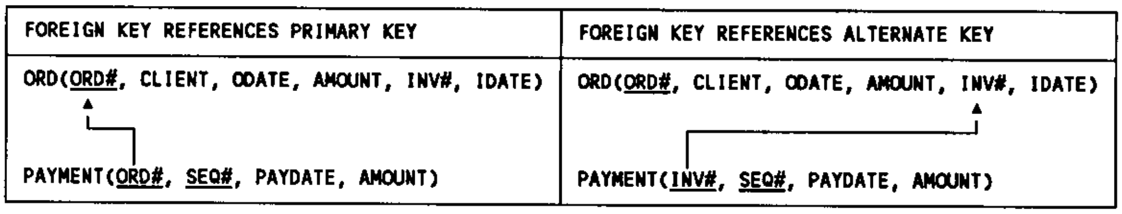

FOREIGN KEY REFERENCES PRIMARY KEY
ORD(ORD#, CLIENT, ODATE, AMOUNT, INV#, IDATE)
PAYMENT(ORD#, SEQ#, PAYDATE, AMOUNT)
FOREIGN KEY REFERENCES ALTERNATE KEY
ORD(ORD#, CLIENT, ODATE, AMOUNT, INV#, IDATE)
PAYMENT(INV#, SEQ#, PAYDATE, AMOUNT)

Figure 5. Orders, Invoices and Payments

6.1 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 database design can then always be identified by means of its relation name, together with a set of values for its primary key attributes.

Difficulties occur when one encounters classes that contain at most one object, in which case the relation name is sufficient identification. The relational model requires the database designer to arbitrarily specify a primary key and to express an explicit constraint to the effect that the relation must not contain more than one tuple. If it were allowed to assign the empty set as the primary key of any such relation, as suggested by Warden (1990), no extra constraint would be required and no meaningless primary key would have to be assigned to the relation.

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 (Date 1990b, p. 135). However, this argument does not hold true in the case where alternate keys designate different statuses of objects, and references to these objects are made depending on their status. Consider Figure 5, which shows part of a database for an order entry application. It is assumed that all orders received will at some time be delivered and an invoice sent to the customer. The customer pays for the delivery later, possibly in several installments. Orders received and invoices sent must be numbered consecutively.

If the foreign key in PAYMENT references the primary key in ORD, the database 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 captured better because the existence of the alternate key expresses the fact that the order has a status in which payments are possible.

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 one in Figure 5, in order to determine the price we pay for simplicity.

6.2 Integration of Data and Functional Aspects

Ideally, not only static (database) concepts but also dynamic (programming) concepts should be part of a data model. Existing data models typically describe static concepts only. This also holds true for the relational model, although this model does provide its users with operators for data manipulation. The model contains concepts such as relation, attribute and tuple, but lacks concepts such as program or transaction. Support for such concepts is left to RDBMS vendors or to database designers.

This omission may present problems when extensions to the relational model are suggested. Consider the discussion about the extension of the relational model to include the foreign key rules Cascade, Delete and Nullify (Date 1990b, Chapter 5). The introduction of such rules into RDBMSs would be a great help to 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 ORD R tuples and for another not to reject such an operation.

7. CONCLUSIONS AND RECOMMENDATIONS

The three preceding sections demonstrate that in many respects the relational model fails to support the interpretation and representation principles introduced in section 2. This failure always results in the introduction of ad hoc explicit constraints. By their very nature, the meaning that such constraints add to a database design is not accessible by a DBMS or an application program.

One possible way to tackle this problem is to introduce more generalized constraints 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 constraints occur most frequently is to initiate empirical research focusing on existing database designs. In any case, the inherent fuzziness of such extensions requires communication between data model designers and the database designers.

Another recommendation would be to express the relational model in relational terms. It is perfectly feasible to express the relational model, at any rate its structural and integrity parts, in relational terms. Not only would this make the relational model more comprehensible to database designers, but it would also provide us with a yardstick against which attempts to improve the expressive power of the relational model could be measured.

In addition, it is advisable to refrain from using unrealistic examples when discussing data models. If it is impossible to find realistic examples to support an argument, then the argument is probably worthless.

One 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 data models that are semantically more expressive. In either case the problems discussed in this paper will have to be addressed.

8. ACKNOWLEDGMENTS

The authors are members of the MESDAG Research Group, sponsored by the N.V. Netherlands Railways, RAET N.V., and the Vrije Universiteit Amsterdam. MESDAG is an acronym for MEta Systems Design And Generation.

9. REFERENCES

Brodie, M. L. "On the Development of Data Models." In M. L. Brodie (Ed.), *On Conceptual Modelling*, New York: Springer-Verlag, 1984, pp. 19-47.

Codd, E. F. "Extending the Database Relational Model to Capture More Meaning." *ACM TODS*, Volume 4, Number 4, December 1979, pp. 397-434.

Codd, E. F. *The Relational Model for Database Management: Version 2*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.

Date, C. J. *An Introduction to Database Systems* (Third Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

Date, C. J. *Relational Database: Selected Writings*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1989.

Date, C. J. *An Introduction to Database Systems, Volume I* (Fifth Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1990a.

Date, C. J. *Relational Database: Writings 1985 — 1989*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1990b.

Shneiderman, B., and Thomas, G. "An Architecture for Automatic Relational Database System Conversion." *ACM TODS*, Volume 7, Number 2, June 1982, pp. 235-257.

Smith, J. M., and Smith, and D. C. P. "Database Abstractions: Aggregation and Generalization." *ACM TODS*, Volume 2, Number 2, June 1977, pp. 105-133.

Veldwijk, R. J.; Boogaard, M.; Dijk, M. V. van; and Spoor, E. R. K. "EDSOs, Implosion and Explosion: Concepts to Automate a Part of Application Maintenance." *Information and Software Technology*, Volume 33, Number 5, June 1991, pp. 1-8.

Warden, A. "Table_Dee and Table_Dum." In C. J. Date, *Relational Database: Writings 1985 — 1989*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.

10. ENDNOTES

1. More specifically, the extent to which the relational model supports logical data independence is insufficient. See Shneiderman and Thomas (1982) and Veldwijk et al. (1991) for a further discussion of this problem.

2. Date uses the misleading term *"special case constraint."*

3. 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.

4. Note that, unlike the incorporation of 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.