A VIEW OF DIRECTIONS IN RELATIONAL DATABASE THEORY

Jeffrey D. Ullman†
Stanford University
Stanford, Calif., USA

ABSTRACT

We shall briefly survey what the author believes are some of the most fruitful directions in relational database theory. These directions include dependency inferences, support for the universal relation concept, null value semantics, and an exploration of the properties of acyclic database schemes.

I. Introduction

We shall assume the reader is familiar with the basic concepts of relational database theory, at least at the level covered by [U] or [BBG]. These concepts include

1. *Relation Scheme.* This is a set of *attributes*, which are names of columns for relations. We shall use $A, B, C, \ldots,$ for attributes and shall use concatenation for forming sets of attributes. Thus AB stands for $\{A, B\}$. We shall use $\ldots, X, Y, Z$ for sets of attributes; concatenation will also stand for union. Thus XY stands for $X \cup Y$.
2. *Relation.* A relation is a set of *tuples*, which, strictly speaking, are maps from the set of attributes (relation scheme) for that relation, to the corresponding *domains*, or data types for the attributes. It is normal to display relations as tables, where the attributes head the columns and the rows are tuples. We then fix an order for the attributes and represent tuples by the sequence of values that tuple has for each of the attributes in order. A relation scheme is assumed to remain constant over time, while the relation corresponding to it changes frequently. Thus we speak of the "current" relation for a relation scheme or talk about the set of possible relations for the scheme.
3. *Functional Dependency* (FD). A functional dependency $X \rightarrow Y$ is an assertion about a relation scheme. It asserts of any "legal" relation that two of its tuples t and s that agree on set of attributes X (written $t[X] = s[X]$) also agree on Y. A relation with that property is said to *satisfy* the dependency.
4. *Multivalued Dependency* (MVD). A multivalued dependency $X \twoheadrightarrow Y$ is, very informally, a statement that there is a set of Y-values associated with each X-value. More formally, if t and s are two tuples of a relation that satisfies $X \twoheadrightarrow Y$, and $t[X] = s[X]$, then the relation must have a third tuple u that takes the Y-value from one of the tuples, say t, and its value everywhere else from the other. That is, if Z is all attributes not in XY, then $u[Y] = t[Y]$ and $u[XZ] = s[XZ]$.
5. *Normalization.* Certain relation schemes have redundancy, in the sense that predictable values appear in certain tuples of the relations over that scheme. It is the dependencies, such as functional and multivalued dependencies, that we assume about "legal" relations over that scheme, that cause redundancy. The appropriate response to redundancy is *decomposition*, the replacement of one relation scheme

† Work partially supported by NSF grant MCS-80-12907.

by two or more (not usually disjoint) schemes. The theory of normalization relates dependency types to desirable properties of relation schemes; those properties involve the dependencies that hold for the relation scheme and can be construed as saying "there can be no redundancy caused by dependencies of the given type in the relations for this scheme." The normal forms that eliminate redundancy caused by functional dependencies, called *third normal form* and *Boyce-Codd normal form*, come from the seminal papers by Codd [C1, C2], while the normal form that responds to multivalued dependencies, *fourth normal form*, is from [F1].

Example 1: Consider a relation scheme whose attributes are E, S, D, and M, standing for employee, salary, department, and manager. Informally, we intend that the salary is the (unique) salary of the employee, the employee may work for several departments, and each department may have several managers. Put another way, a tuple *esdm* in the "current" relation over this scheme is an assertion that, at the current time, e is an employee, his salary is s, one of the departments he works for is d, and one of the managers of d is m. We can express the requirements on the "current" value of this relation by the functional dependency $E \rightarrow S$, and the multivalued dependency $D \rightarrow\!\!\!\rightarrow M$. It is interesting to observe that the MVD $E \rightarrow\!\!\!\rightarrow D$ is false, even though the intuitive notion of meaning for this relation is that "there is a set of departments for each employee." That is, just because $es_1d_1m_1$ and $es_2d_2m_2$ are tuples doesn't mean that $es_2d_1m_2$ is a tuple, since manager m_2 may not manage department d_1. The reader should thus be warned that the colloquial meaning of MVD's is very informal, and the definition must be checked carefully before asserting an MVD.

There is considerable redundancy in this relation scheme. For example, the salary of an employee is repeated once for each related department-manager pair for the employee, and the managers of a department are repeated for each employee. The normalization process would replace this one scheme by ES, ED, and DM. Each of these schemes are in fourth normal form, by the way. []

II. Universal Relations

A *database* is a collection of relations, and their relation schemes collectively form the *database scheme*. In order to make queries that involve data from more than one relation, it is convenient to imagine that the true world these relations represent is a single relation whose scheme consists of all the attributes mentioned in the scheme for any of the relations. We call this relation the *universal relation* and its scheme the *universal relation scheme*. It is this viewpoint that justifies the decomposition of one relation scheme into many during the process of normalization, although early papers on normalization saw themselves as removing redundancy from a relation scheme that could be one of many (unrelated) relation schemes in a database scheme.

Another, equally important justification for interest in this *universal relation assumption* is that it provides a simple user interface. If we allow the user to query about the universal relation, then he needs to know only about the attributes and their intuitive meanings and relationships, without concerning himself about the structure of the database. For example, natural language interfaces often make the universal relation scheme assumption tacitly.

A stronger form of assumption is that not only does the universal relation exist, but at all times, the relations in the database are the projections of this universal relation.

We *project* a relation r with scheme R onto a subset of the attributes in R, say S, by forming $\{ t[S] \mid t \text{ is in } r \}$, and we denote the result by $\pi_S(r)$. This *pure universal relation assumption* has been criticised on various grounds ([K], [BG1]), principally because of its inability to express all conceivable real-world situations. There is merit to that point of view, although the universal relation can be given meaning in far more cases than anticipated if we carefully define null values in tuples and the ways they operate and interact with dependencies.

Thus, in Example 1, we might imagine that there is a universal relation over attributes $ESDM$. If we had an employee with no salary recorded, we might believe that the universal relation assumption forced us to discard all information about this employee. In this case, the simple device of entering a "null salary" in the S-component of tuples for this employee will serve, provided we agree that the identical null value appears in all the employee's tuples (or else the FD $E \rightarrow S$ would be violated). A number of recent papers involve the use of nulls in defining the semantics of universal relations; these include [KU], [L2], [M], [S1], [V], and [W].

Whether or not the universal relation assumption can be made in all real-world cases is, perhaps, not the most important issue, anyway. Like the context-free grammar, for example, there is a great deal of power in the concept, and awkwardnesses that arise can be finessed much the way noncontext-free aspects of programming languages are finessed when one designs a parser for the language.

Moreover, the need to make the pure universal relation assumption is dubious. I prefer to see it as a guide to relational database design (discussed as the lossless join property, below) and to the interpretation of queries over the universal relation (see [FMU, KU]), while it disturbs me not in the least if one or another relation in the database contains a *dangling* tuple, one that cannot be explained in the light of the other relations as being the projection of any tuple in a universal relation. Thus, referring to Example 1, the presence of tuple (Jones, Toy) in the ED relation while no entry for Jones is in the ES relation merely means that Jones' salary is unknown, or null. There should be no problem with answering "Jones" when queried for the employees of the Toy Dept., even though had the query been "what is Jones' salary?", we could not have made a sensible response.

III. Decomposition Theory

We alluded to the fact that a principal activity in relational database design is the decomposition of (universal) relation schemes into sets of attributes that serve as schemes for the relations in the database and have certain nice properties (which we haven't defines here, and don't intend to—see [B, C1, F1, U], e.g.). There are also some requirements that the database scheme, or collection of relation schemes must have as a whole; for example, we cannot always choose the database scheme consisting of all pairs of attributes in the universal relation scheme, even though a two-attribute relation scheme always satisfies just about any normal form you can name.

One requirement on the database scheme is the *lossless join property*, the condition that when you take a legal universal relation and project it onto the relation schemes in the decomposition, then "join" the results (a term we'll define in a moment), you get back no more tuples than you started with (it is easy to show you never get less). If U is a universal relation scheme, and $R_1, \ldots, R_n$ a decomposition, then the *join* of relations $r_1, \ldots, r_n$ over schemes $R_1, \ldots, R_n$, respectively, denoted $r_1 \bowtie \cdots \bowtie r_n$, is the set of tuples t such that for all i, $t[R_i]$ is in r_i.

Example 2: Let $U = ABC$ be the universal relation scheme and $r = \{102, 304\}$ the universal relation. Consider the decomposition AB, BC. Then $\pi_{AB}(r) = \{10, 30\}$, and $\pi_{BC}(r) = \{02, 04\}$. Further, $\{10,30\} \bowtie \{02,04\} = \{102, 104, 302, 304\} \neq r$. We conclude that this decomposition does not satisfy the lossless join property, as long as whatever dependencies we believe about the universal relation are satisfied by the relation r above. []

Notice that whether the lossless join property holds for a decomposition is determined by what dependencies we assume hold for the universal relation, since it may be that all "legal" relations, projected and then rejoined, give back what you started with, while other relations that are not "legal" do not restore themselves after projection and join. Also note that the lossless join property does not require us to assume there is a meaningful universal relation; it merely says what must happen if we start with a universal relation and project. It is in this sense that the universal relation assumption is a guide to decomposition.

The motivation for the lossless join property is that without it, the projected relations cannot be said to represent a unique universal relation. If you believe the universal relation assumption, the absence of the lossless join property would surely imply a defect in the decomposition. But even if you do not make this assumption, when decomposing a relation scheme with redundancy, if we do not have the lossless join property, we can be certain that the relations in the decompsition will sometimes fail to represent the relation we started with. Thus, in Example 2, the projected relations over AB and BC could represent $r = \{102, 304\}$, or they could equally well represent $\{102, 104, 302, 304\}$, or several other relations; we can't tell which if we don't know the universal relation from which they were projected.

[ABU] first discussed the lossless join property as a goal for decomposition, although the need for the property was implicit as early as [C1].

A second goal for decomposition is the *dependency preservation property*, which requires that we be able to infer the dependencies that hold for the universal relation from their projections onto the schemes of the decomposition. That is, if D and E are two sets of dependencies (functional, multivalued, or whatever) that apply to relations over scheme U, we say D *logically implies* E, written $D \models E$, if every relation over U that satisfies D also satisfies E. Given set of dependencies D over U and subset R of U, the *projection* of D onto R, written $\pi_R(D)$, is the set of dependencies that are true for all relations $\pi_R(r)$ such that r satisfies D. A dependency over R can be considered to be a dependency over U by defining it to be satisfied by a relation r over U if and only if $\pi_R(r)$ satisfies the dependency. We can then define the decompostion $R_1, \ldots, R_n$ to have the dependency preservation property for a set of dependencies D if D is logically implied by $\bigcup_{i=1}^{n} \pi_{R_i}(D)$.

The motivation for this property is that without it, certain constraints that we believe hold in the universal relation cannot be checked in the relations of the database. Again, even if we deny the universal relation assumption, when doing the decomposition of a relation with redundancy, we shall be giving up our ability to check these constraints if the decomposition does not have the dependency preservation property.

The first discussion of dependency preservation as a property of decompositions is in [R2], although the idea is implicit in [B]. [H] gives an efficient test for the property when the dependencies involved are only functional. [ABU] discusses how the projections of dependencies are to be computed within the worlds of functional and multivalued dependencies, while [GZ] shows how functional dependencies project in the space of any dependencies whatsoever.

IV. Dependency Inference

The problem of dependency inference, that is, determining whether a group of dependencies implies another, can be perceived as the driving force behind a large fraction of recent work in relational theory. There are three reasons why the problem is important.

1. Dependency preservation is an imprtant goal of decomposition, and the test for the property evidently involves inference of dependencies.
2. We shall see that the lossless join property can also be viewed as an implication of dependencies.
3. When dealing with queries about a universal relation, and wishing to translate them into a query about certain relations in the database, a well-motivated approach is to find some subset of the relation schemes whose attributes include those queried about, and which have a lossless join ([MU1], [KU]). The losslessness of a join is, again, a dependency inference to be made.

We could, of course, add to the above the fact that there is considerable intellectual challenge in this quite natural problem of dependency inference, but let that slide.

The problem of dependency inference has been made difficult by the fact that functional and multivalued dependencies are not the only kinds of dependencies we could conceive of, and more importantly, they are not the only ones that appear in the "real world."

Example 3: A canonical example of a believable dependency that is not in the classes we have discussed occurs in a universal relation with attributes C, S, P, and Y, standing for course, student, prerequisite, and year (the student took the prerequisite). The constraint we have in mind is that there is a set of students for the course and a set of prerequisites for the course, and every student took every prerequisite in some year (that year could even be null if the student doesn't belong in the course). In terms of tuples in the universal relation, we assert that if t and s are two tuples, with $t[C] = s[C]$, then there is some tuple u in the same relation with $u[CS] = t[CS]$ and $u[P] = s[P]$. Note that this constraint is not the MVD $C \twoheadrightarrow S$, since $u[Y] = s[Y]$ is not required, nor is it the MVD $C \twoheadrightarrow P$ for a similar reason; neither of these MVD's hold.

However, the above constraint means that if we project the universal relation onto set of attributes CSP, we do get the MVD's $C \twoheadrightarrow S$ and $C \twoheadrightarrow P$ in CSP. We call this constraint an *embedded* multivalued dependency and write it $C \twoheadrightarrow S \mid P$ to emphasize the fact that it is an MVD in CSP, not in $CSPY$. []

There is a horrible fact about embedded MVD's; it is not known whether their inference problem is decidable. Thus, if you want to do decomposition of relations into nice normal forms, or you want to support a universal relation by inferring convenient lossless joins to take in response to queries, you are faced with either pretending embedded MVD's don't exist, or doing an inaccurate job in certain circumstances (or solving an outstanding open problem).

There have been two different responses to the apparent undecidability of inference for real-world dependencies.

1. An attack on the decidability issue was mounted. The principal directions were to search for axiomatizations of a class of dependencies that included embedded MVD's, and to look for a larger class in which a decision procedure would be apparent, while the specialization of that algorithm to embedded MVD's was too murky to be visible without going to the larger class. These attempts have largely been failures, although much of intellectual interest was produced as a side effect.
2. A revision of the notion of what dependencies were appropriate was attempted. The idea here was to find a new class of dependencies that included the functional dependencies, the multivalued dependencies, and those of the embedded MVD's that we might realistically expect, yet for which inference of dependencies was tractable.

   We shall survey each of these approaches in turn.

V. Generalized Dependencies

The first attempt to generalize FD's and/or MVD's was the *join dependency* studied in [R1]. The join dependency $\bowtie(R_1, \ldots, R_n)$ is an assertion that if we project a relation satisfying this dependency onto the sets of attributes R_1 through R_n and rejoin, we get back what we would get if we had projected the relation directly onto $\bigcup_{i=1}^{n} R_i$. The embedded MVD $X \twoheadrightarrow Y \mid Z$ is equivalent to the join dependency $\bowtie(XY, XZ)$. Thus join dependencies generalize MVD's and embedded MVD's, although no nontrivial functional dependency is a join dependency. The properties of join dependencies were studied by [BV1], while a complete axiomatization (really an axiomatization for a slightly larger class) appears in [S2].

A second class, called *subset dependencies* was discussed by [SW]. This class also generalizes embedded MVD's, and the paper contains the important result that there can be no finite axiomatization for embedded MVD's.

A more general sort of dependency occurred at about the same time to many people; we shall follow [BV2] and call them simply *generalized dependencies.* These dependencies say that if we see tuples in a relation with a certain pattern, then we can expect to see something else, either another tuple or an equality between two symbols. We write a generalized dependency over a universal relation scheme of n attributes as $(r_1, \ldots, r_k)/r$, where $r_1, \ldots, r_k$ are called the *hypothesis rows* and consist of strings of length n, and r is called the *conclusion*; it is either another row of length n or an equality of the form $a = b$, where a and b are symbols appearing among the hypothesis rows.

We require that no symbol appear in more than one column (the term *untyped dependencies* has been applied to dependencies satisfying this assumption). The symbols appearing in the conclusion, in the case it is a row, rather than an equality, need not appear anywhere else. Symbols of the conclusion not appearing elsewhere are called *unique.* A generalized dependency with no unique symbols is called *full*; otherwise it is *embedded.*

The meaning of such a dependency is that whenever there is a symbol-to-symbol mapping that maps each hypothesis row into a tuple of a particular relation, then

1. If the conclusion is a row, then we can extend the mapping to include the unique symbols in such a way that the conclusion is also mapped into a tuple of the relation.
2. If the conclusion is an equality $a = b$, then the mapping must have mapped a and

b to the same symbol; no other mappings of all the hypothesis rows into tuples of the relation are possible.

Example 4: Consider the $ESDM$ universal relation of Example 1. If we assume the attributes are in that order, the FD $E \rightarrow S$ can be written

$$(es_1d_1m_1, es_2d_2m_2)/s_1 = s_2$$

The two hypothesis rows can be mapped to any two tuples, as long as the tuples agree on E, because e is the only symbol appearing in both hypothesis rows. Thus, this dependency says that whenever we find two tuples that agree in their E-component, it turns out that they also agree in their S-component.

The MVD $D \rightarrow\!\!\rightarrow M$ can be written $(e_1s_1dm_1, e_2s_2dm_2)/e_1s_1dm_2$, while the embedded multivalued dependency $D \rightarrow\!\!\rightarrow E \mid M$ can be expressed

$$(e_1s_1dm_1, e_2s_2dm_2)/e_1s_3dm_2$$

Note that the embedded MVD has a unique symbol, s_3, while the other examples do not have unique symbols. []

The generalized dependencies can have more than two hypothesis rows. For example, a join dependency with k components will have k hypotheses. The generalized dependencies with two hypothesis rows and another row for the conclusion are exactly the subset dependencies. The generalized dependencies with rows as conclusions (call them *tuple generating*) were considered by [P, PJ] and called generalized dependencies, and by [SU1, SU2] and called "template dependencies." A class larger than the generalized dependencies, in that they allow multiple related conclusions, was considered by [F2], where they were called "implicational dependencies," and by [YP], where they were called "algebraic dependencies."

Complete axiomatizations were given in a number of these papers: [BV2], [SU1], and [YP]. [F2] also contains the interesting notion of an *Armstrong relation* for a set of dependencies, that is, a relation that satisfies the dependencies, those dependencies that follow logically from it (as it must), but no other dependencies. It is shown there that generalized dependency sets always have Armstrong relations.

There are partial decision procedures for all these classes of dependencies. An axiom system provides such a method, since we may systematically search for proofs of the inference, but we cannot be sure that we shall find the proof at any particular time. Another approach is to apply the "chase" process of [ABU, MMS, SU1], in which we start with the hypothesis rows and infer additional rows and equalities among symbols. If we ever infer the conclusion, we are done, but embedded dependencies, with their ability to cause new symbols to be generated for the new rows (every occurrence of a unique symbol must be replaced by a distinct value when generating tuples) allow the generation process to go on forever. However, if the dependencies are full, we are constrained within the set of symbols of the original hypothesis rows, and thus we have a decision algorithm for inferences of full dependencies.

Recently, [CLM] has proven the inference problem undecidable for tuple-generating generalized dependencies. This paper effectively puts an end to efforts to solve the inference problem for embedded MVD's by considering larger classes of dependencies, but we should note that the decidability of inference for embedded MVD's is still an open problem.

VI. Description by Functional and Join Dependencies

The other direction responding to the problem of an undecidable inference problem for real-world dependencies is to develop a substitute class of dependencies for which inference is decidable. In [FMU] it was proposed that a suitable assumption about the dependencies obeyed by a universal relation is that there is an arbitrary collection of functional dependencies and one full join dependency. The connection between join dependencies and common types of database descriptions was also perceived by [L1].

The reason we see a full join dependency as sufficient to replace collections of MVD's is that the intuitive meaning one ascribes to a universal relation directly implies that the universal relation satisfies a particular join dependency. That is, as long as you believe the universal relation makes sense, you may as well assume this join dependency holds. Moreover, it appears in practice that all the MVD's we assert about a given universal relation follow logically from this join dependency, although we cannot, of course, prove that to be the case. More of this motivation is contained in [FMU], but here, let us be content with an example.

Example 5: Consider the universal relation from Example 1. There we defined the set of tuples in the current universal relation as

$$\{\, esdm \mid P_1(e,s) \text{ and } P_2(e,d) \text{ and } P_3(d,m)\,\}$$

The predicate P_1, for example, means "s is the salary of e." It is easy to show that the set of relations that can be so defined, for some P_i, $1 \leq i \leq 3$, is exactly the set of relations that satisfies the join dependency $\bowtie(ES, ED, DM)$.

In general, the components mentioned in the predicates defining the universal relation become relation schemes mentioned in the join dependency. The only additional semantics we need in this case is the FD $E \rightarrow S$; the MVD $D \twoheadrightarrow M$ can be shown to follow logically from the join dependency. []

The paper [MU1] discusses some of the problems that arise with this sort of universal relation semantics. It also shows how embedded MVD's can frequently have their constraint expressed by a concept called "maximal objects," which are not dependencies, but rather, influence the way queries over the universal relation are interpreted. Thus, much of the descriptive power of embedded MVD's can be captured, yet lossless joins are easy to deduce from the full join dependency and a collection of functional dependencies.

VII. Acyclic Join Dependencies and Hypergraphs

An interesting development from the above hypothesis about appropriate universal relation semantics is that a class of join dependencies called "acyclic" has been defined. These dependencies, it can be argued, are the only ones that should appear in real-world universal relation descriptions, if we take the trouble to split a single attribute that plays two or more roles in its relationships with other attributes. Without taking a stand whether this argument is valid, we can profitably explore and enjoy the theory of acyclic join dependencies for its naturalness, the many interesting properties it exposes, and its contribution to hypergraph theory.

To begin, for any join dependency we can construct a hypergraph whose nodes are the attributes and whose hyperedges are the components of the join dependency. For example, the hypergraph for the join dependency mentioned in Example 5 has node set

$\{E,S,D,M\}$ and set of edges $\{\{E,S\},\{E,D\},\{D,M\}\}$. This hypergraph is thus an ordinary graph, since the edges are all doubletons, but that will not be the case for general join dependencies.

An *articulation set* in a hypergraph is the intersection of two edges, whose removal increases the number of connected components of the hypergraph. For example, in the above hypergraph, $\{D\}$ is the intersection of edges $\{E,D\}$ and $\{D,M\}$, and its deletion breaks the hypergraph into two components, $\{E,S\}$ and $\{M\}$. Thus, D by itself is an articulation set; such a set need not have only one node, of course.

A *node-generated set of edgès* of a hypergraph is formed from some set of nodes N, by intersecting each edge with N, then throwing away edges that become subsets of other edges. In our running example, the set $\{E,S,D\}$ yields the set of edges $\{\{E,S\},\{E,D\}\}$, since each of those two edges is contained in the set, while $\{E,S,D\} \cap \{D,M\} = \{D\}$, which is contained in the edge $\{E,D\}$ and thus does not appear.

We say a hypergraph is *acyclic* if every node-generated set of more than one edge has an articulation set. Otherwise it is *cyclic*.

Example 6: The hypergraph with edges $\{A,B,C\}$, $\{C,D,E\}$, and $\{E,F,A\}$ is cyclic, as we would expect. The entire hypergraph is a node-generated set of edges with no articulation set. However, if we add the edge $\{A,C,E\}$, the graph suddenly becomes acyclic, pointing out some of the subtlety in this concept. For example, $\{A,C\}$ is an articulation set for the whole hypergraph. []

There are a number of interesting properties of acyclic hypergraphs, summarized in [B*]. We shall point out a few of them here.

1. A join dependency is logically equivalent to a set of MVD's if and only if it is acyclic [FMU].
2. The *Graham reduction* [G] of a hypergraph is obtained by applying the following two operations, in any sequence, until they can no longer be applied (the result can be shown unique). (a) Delete a node that appears in only one edge. (b) delete an edge that is a subset of another edge. Then a hypergraph is acyclic if and only if its Graham reduction is empty [BFMY].
3. We say a set of relation schemes has the *LCIGC property* (local consistency implies global consistency) if we can test whether the current relations for these schemes are "globally consistent" (are the projections of a universal relation) by testing "local consistency" (whether the projections of any two relations, say with schemes R and S, onto the set of attributes $R \cap S$ are the same). Then a set of relation schemes has the LCIGC property if and only if the hypergraph with those schemes as edges is acyclic [BFMY].

It is interesting to note that [Z] and [G] had searched for a notion of "acyclic hypergraph," principally with an eye toward testing the LCIGC property; however, their definitions are too strong, failing, for example, to recognize that the second hypergraph in Example 6 is acyclic.

Another way in which acyclic hypergraphs play a role is in the notion of "tree queries" of [BG2]. These queries, which were investigated because of their nice properties for execution in a distributed database system, are equivalent to the concept of acyclic hypergraph [BFMY]. Interestingly, [H] showed the relationship between tree queries and the LCIGC property, and [YO] claimed that a variant of Graham reduction served to identify the tree queries, without relating any of these concepts to acyclic hypergraphs.

The reader should consult [Y], [B*] and [BFMY] for additional properties of acyclic databases and hypergraphs. Finally, [MU2] gives a notion of "paths" in a hypergraph and shows that, in a sense, acyclic hypergraphs are exactly the hypergraphs for which paths between nodes are unique. That result has a significance in database theory, in that it implies unique connections between attributes in the database, whenever the database consists of a collection of relations that form an acyclic hypergraph.

References

[ABU] Aho, A. V., C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems* 4:3 (1979), pp. 297–314.

[BBG] Beeri, C., P. A. Bernstein, and N. Goodman, "A sophisticate's introduction to database normalization theory," *Proc. International Conference on Very Large Data Bases,* pp. 113–124, 1978.

[BFH] Beeri, C., R. Fagin, and J. H. Howard, "A complete axiomatization for functional and multivalued dependencies," *ACM SIGMOD International Symposium on Management of Data*, pp. 47–61, 1977.

[BFMY]Beeri, C., R. Fagin, D. Maier, and M. Yannakakis, "On the desirable properties of acyclic database schemas," manuscript in preparation.

[BV1] Beeri, C. and M. Y. Vardi, "On the properties of join dependencies," *Proc. Workshop on Formal Bases for Databases,* Toulouse, Dec., 1979.

[BV2] Beeri, C. and M. Y. Vardi, "Complete axiomatizations for generalized dependencies," Hebrew Univ., 1980.

[B*] Beeri, C., R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman, and M. Yannakakis, "Properties of acyclic database schemes," to appear in *Proc. 1981 ACM Symposium on Theory of Computing.*

[B] Bernstein, P. A., "Synthesizing third normal form relations from functional dependencies," *ACM Transactions on Database Systems* 1:4 (1976), pp. 277–298.

[BG1] Bernstein, P. A. and N. Goodman, "What does Boyce-Codd normal form do?," *Proc. International Conference on Very Large Data Bases,* 1980.

[BG2] Bernstein, P. A. and N. Goodman, "The theory of semijoins," TR CCA-79-27, Computer Corp. of America, Cambridge, Mass., 1979.

[CLM] Chandra, A. K., H. R. Lewis, and J. A. Makowsky, "Embedded implicational dependencies and their inference problem," Harvard Univ., 1980.

[C1] Codd, E. F., "A relational model for large shared data banks," *Comm. ACM* 13:6 (1970), pp. 377–387.

[C2] Codd, E. F., "Further normalization of the data base relational model," in *Data Base Systems* (R. Rustin, ed.), Prentice Hall, Englewood Cliffs, N. J., 1972.

[F1] Fagin, R., "Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systems* 2:3 (1977), pp. 262–278.

[F2] Fagin, R., "Horn clauses and database dependencies," *Proc. Twelfth Annual ACM Symposium on the Theory of Computing,* pp. 123–134, 1981.

[FMU] Fagin, R., A. O. Mendelzon, and J. D. Ullman, "A simplified universal relation assumption and its properties," RJ2900, IBM, San Jose, Calif., 1980.

[G] Graham, M. H., "On the universal relation," technical report, Univ. of Toronto, Sept., 1979.

[GZ] Ginsburg, S. and S. M. Zaiddan, "Properties of functional dependency databases," Dept. of C. S., USC, 1980.

[H] Honeyman, P., "Properties of the universal relation assumption," Ph. D. thesis, Princeton Univ., Princeton, N. J., 1980.

[K] Kent, W., "Consequences of assuming a universal relation," IBM technical report, Dec., 1979, to appear in *TODS*.

[KU] Korth, H. F. and J. D. Ullman, "SYSTEM/U: a database system based on the universal relation assumption," *Proc. XP1 Conference*, Stonybrook, N. Y., June, 1980.

[L1] Lien, Y. E., "On the equivalence of database models," private communication, June, 1980.

[L2] Lien, Y. E., "Multivalued dependencies with null values in relational data bases," *Proc. International Conference on Very Large Data Bases*, pp. 61–66, 1979.

[M] Maier, D., "Discarding the universal instance assumption: preliminary results," *Proc. XP1 Conference*, Stonybrook, N. Y., June, 1980.

[MU1] Maier, D. and J. D. Ullman, "Maximal objects and the semantics of universal relation databases," TR–80–016, Dept. of C. S., SUNY, Stonybrook, N. Y., 1980.

[MU2] Maier, D. and J. D. Ullman, "Paths in acyclic hypergraphs," manuscript in preparation.

[MMS] Maier, D., Y. Sagiv, and A. O. Mendelzon, "Testing implications of data dependencies," *ACM Transactions on Database Systems* 4:4 (1979), pp. 455–469.

[P] Paredaens, J., "A universal formalism to express decompositions, functional dependencies, and other constraints in a relational database." To appear in *TCS*. To appear, *JACM*.

[PJ] Paredaens, J. and D. Jannsens, "Decomposition of relations: a comprehensive approach," in *Formal Bases for Databases* (H. Gallaire and J.-M. Nicolas, eds.), CERT workshop, Toulouse, 1979.

[R1] Rissanen, J., "Theory of joins for relational databases—a tutorial survey," *Proc. Seventh Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in CS, 64, Springer-Verlag, pp. 537–551.

[R2] Rissanen, J., "Independent components of relations," *ACM Transactions on Database Systems* 2:4 (1977), pp. 317–325.

[S1] Sciore, E., "Null values, updates, and normalization in relational databases," doctoral dissertation, Princeton Univ., Princeton, N. J., 1980.

[S2] Sciore, E., "A complete axiomatization for full join dependencies." To appear in *JACM*.

[SW] Sagiv, Y. and S. Walecka, "Subset dependencies as an alternative to embedded multivalued dependencies." To appear in *JACM*.

[SU1] Sadri, F. and J. D. Ullman, "A complete axiomatization for a large class of dependencies in relational databases," *Proc. Twelfth Annual ACM Symposium on the Theory of Computing*, pp. 117–122, 1980.

[SU2] Sadri, F. and J. D. Ullman, "The interaction between functional dependencies and template dependencies," *ACM SIGMOD International Symposium on Management of Data*, pp. 45–51, 1980.

[U] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, Potomac, Md., 1980.

[V] Vassilou, Y., "Null values in database management—a denotational semantics approach," *ACM SIGMOD International Symposium on Management of Data*, pp. 162–169, 1979.

[W] Walker, A., "Time and space in a lattice of universal relations with blank entries," *Proc. XP1 Conference*, Stonybrook, N. Y., June, 1980.

[Y] Yannakakis, M., "Properties of acyclic databases," unpublished memorandum, Bell Laboratories, Murray Hill, N. J., 1981.

[YO] Yu, C. T. and M. Z. Ozsoyoglu, "An algorithm for tree-query membership of a distributed query," *Proc. Compsac-79*, pp. 306–312, 1979.

[YP] Yannakakis, M. and C. H. Papadimitriou, "Algebraic dependencies," *Proc. Twenty First Annual IEEE Symposium on Foundations of Computer Science*, pp. 328–332, 1980.

[Z] Zaniolo, C., "Analysis and design of relational schemata for database systems," Ph. D. thesis, UCLA, 1976.