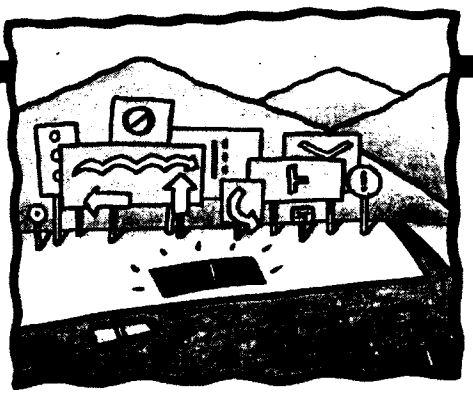


The Universal-Relation Data Model for Logical Independence

Moshe Y. Vardi, *IBM Almaden Research Center*

This relational model keeps access-path independence by removing the need for logical navigation among relations. One benefit is a simple yet powerful query-language interface.

Because early database-management systems required that application programmers and end users alike specify access paths, Codd introduced the relational model to free the programmer and user from what he called the "navigation problem."[1] He also wanted to eliminate the need for program modification to accommodate changes in the database structure (that is, to eliminate access-path dependence in programs).

But after a few years of experience with relational database-management systems, it became clear that — although it was a significant step forward — the relational model by itself failed to achieve complete freedom from user-supplied navigation and from access-path dependence. The relational model was successful in removing the need for *physical* navigation as no access paths need be specified in a relation's storage structure. Nevertheless, the relational model has not yet provided independence from *logical* navigation, since access paths among the relations must still be specified.

For example, consider a database that has relations *ED*(Employee, Department) and *DM*(Department, Manager). If you are interested in the relationship between employees and managers through departments, you must specify the natural join of the *ED* and *DM* relations, projected onto *EM*. This join is an access-path specification, without which the answer to the query cannot be computed. Thus, a user must know what relations are in the database and how they can be joined. Furthermore, all application programs that include the join are data-dependent. If the database were, for example, reorganized to have a single relation *EDM*, these programs would have to be modified accordingly. Of course, this problem may be overcome by defining a view on *EM*, but that approach may lead to an unwanted proliferation of views.

The *universal*-relation model tries to achieve *complete* access-path independence by letting you ask (in an appropriate language) "tell me about employees and their managers," expecting the system to figure out the intended access path for itself. Of course, you cannot expect the system always to select automatically the intended relationship between employees and manager

0740-7459/88/0300/0080/$01.00 ©1988 IEEE

because the user might have something other than the simplest connection (in this case, the one through departments) in mind — the user instead might be looking for the manager of the manager of the employee, or, for some obscure reasons, the managers of all departments that come alphabetically later than the department of the employee. In a universal-relation system, you must settle for eliminating the need for logical navigation along certain paths — those selected by the designer — while letting the user navigate explicitly in more convoluted ways.

The universal-relation model is not meant to replace the relational model; rather, it is meant to *supplement* it. There are several applications where a universal-relation interface to an existing database-management system is essential. The most obvious example is natural-language interfaces — indeed, it is hard to see how a natural-language interface could reliably use anything else, since it is unreasonable to make the user talk in terms of the database's logical structure. After all, you cannot expect a customer who is interrogating a point-of-sale application in a department store to know anything about the underlying database system.

Unlike the relational model, the universal-relation model was not introduced as a single, clearly defined model — it emerged and evolved during the 1970s through the independent work of several researchers and is still in development. While several experimental implementations have been built, I know of no commercial database-management system that supports it.

This article just introduces the universal-relation model. It sketches the fundamental ideas behind the model and describes a few examples. Other work offers deeper treatment.[2-4]

Fundamental assumptions

Consider the assumptions that are frequently made when talking about universal-relation systems. These assumptions must be satisfied by an application for the universal-relation model to be adequate for that application. All data models have such underlying assumptions, although they are very rarely made explicit.

Universal-relation scheme. Perhaps the most basic assumption is that there is a universal-relation scheme, a set of attributes about which queries may be posed. Further, attributes in this set are assumed to play only one role, and puns are not allowed. Thus, an attribute like Name cannot stand for names of employees, customers, suppliers, and managers in the same universal-relation scheme. This requirement forces the database designer to embed access paths in attribute names. You often hear claims that many attributes must then receive unintuitive names, but there is no evidence that this is true for typical databases — although occasional renaming is, of course, necessary.

The underlying assumption, called relationship uniqueness, strengthens the universal-relationship model because combinations of attributes — not just individual attributes — have unique meanings.

Relationship uniqueness. Another basic assumption is that for all attribute sets X (for example, {Employee, Manager}), there is a unique relationship on this set X that the user has in mind. That does not mean there can be only one relationship on X but that one relationship is the most basic one, so you can assume that this relationship is what the user intends unless it is explicitly specified otherwise. In the earlier example of employees, departments, and managers, you would expect the most basic relationship between managers and employees to be that of "manages," while the relationship "manages the manager of" you would feel intuitively is less basic.

This underlying assumption is called relationship uniqueness. It strengthens the universal-relation scheme assumption because not only are attributes expected to play unique roles, but combinations of attributes likewise have a unique meaning. This basic relationship on a set X of attributes is called a *connection* on X and is denoted $[X]$. (The connection on X has also been called the window on X.) Technically, $[X]$ is a function from database states db to relations $[X](db)$. Given a database db, $[X](db)$ is a relation on X that represents the basic relationship on X for that database.

Connections and query processing. The preoccupation of universal-relation systems with uniqueness of relationships is caused by a seldom-acknowledged assumption that underlies all universal-relation systems: Query processing consists of two steps. The steps are:

- Binding. Construct the connection $[X](db)$ for the set X of attributes in the query.
- Evaluation. Whatever operations must be applied to answer the query are then applied to $[X](db)$.

The binding and evaluation phases are independent. Different connection functions $[X]$ can be used to produce different relations over X without changing the way evaluation works on the resulting relation. (The answer may, of course, be changed.)

For example, the queries

```
retrieve (EMP)
where MGR = "Jones"
```

and

Universal-relation system sampler

Other systems that implement the universal-relation model include:

- Unix's "q" command, which has what is perhaps the simplest computational approach. This command supports the universal-relation model through a rel file, which is essentially a list of joins that can be taken if a query requires it; the first join on the list that covers all the needed attributes is taken. The rel file is set up by the database designer, which means that the designer must carry the extra burden of defining all connections.
- Pique, the PITS Query language, a universal-relation interface for the Pie in the Sky database system developed at the State University of New York at Stony Brook and the Oregon Graduate Center.[1] It is based on the concepts of associations and objects, which are closely related to System/U's objects and maximal objects.
- Aurical, the Universal-Relation Implementation via Codasyl, a universal-relation system at the University of Illinois at Urbana-Champaign.[2] It is implemented in Fortran using an interface to an existing Codasyl database system on a Prime computer. It is based on the weak universal-relation approach.
- DURST, the Datenbank mit Universalrelation-Scnittstelle, a universal-relation system at the University of Dortmund in West Germany.[3] Its approach is similar to FIDL's but does not include a physical implementation of the universal relation.

References

1. D. Maier, D. Rozenshtein, and D.S. Warren, "Window Functions," in *Advances in Computing Research, Vol. 3: The Theory of Databases*, P. Kanellakis and F.P. Preparata, eds., JAI Press, Greenwich, Conn., 1986, pp. 213-246.
2. S.M. Kuck and Y. Sagiv, "A Universal-Relation Database System Implemented via the Network Model," *Proc. First Symp. Princ. Database Systems*, ACM, New York, 1982, pp. 147-157.
3. J. Biskup and H.H. Bruggeman, "Universal Relations Views: A Pragmatic Approach," *Proc. Ninth Int'l Conf. Very Large Databases*, William Kaufmann, Los Altos, Calif., 1983, pp. 172-185.


```
retrieve (MGR)
where EMP = "Smith"
```

are each answered by forming from the database some relation r over (Emp, Mgr) in the binding phase. For the evaluation phase in the first case, you select from r those tuples where Mgr equals "Jones" and then project onto Emp. In the second case, you select for Emp equal to "Smith" and then project onto Mgr.

Defining connections

The crux of a universal-relation system is its connection function. This function is the bridge between the database as a collection of relations and the database as a semantic whole. There are two basic approaches to defining the connection function.

In the first approach, data is treated as if it were all in a single relation over all the attributes. This relation, called the universal relation, is in effect the user view, and $[X](db)$ is taken to be the projection of this relation onto X. In the above example, the universal relation is a relation over the attributes {Emp, Dpt, Mgr}, and the connection on Emp and Mgr is computed by projecting the universal relation onto these attributes.

The second approach deals with how $[X](db)$ is to be computed, without explicit reference to the universal relation. In the above example, the connection on Emp and Mgr is computed by joining the relation ED with the relation DM and projecting the result on {Emp, Mgr}.

The relationship between the two approaches is important because the first approach defines the connection semantically while the second approach, which defines the connection computationally, provides the algorithm needed to compute it.

Semantic definitions. Historically, the first approach was expressed first as the pure universal-relation assumption, which restricted the model to cases where a unique relation u exists over the set U of all attributes such that u satisfies the integrity constraints of the application and to cases where, in every relation scheme R in the database scheme, the current relation for R is the projection of u onto R. In this case, u is the user view and $[X](db)$ is the projection of u onto X.

This approach lets you view the database as a physical representation of a single universal relation, and it helps you deal with the integrity constraints on the database. Clearly, for this approach to work, you must ensure that the universal relation is unique and that you have an effective way of computing it. Nevertheless, even with these issues solved, the pure universal-relation approach is not applicable to a sufficiently broad set of applications because it is too constraining. For example, in the example database of employees, departments, and managers, you cannot store information about the manager of a new department that does not yet have an employee assigned to it.

A modified version of the pure universal-relation approach, which has become known as the weak universal-relation approach, has overcome the difficulties of the pure universal-relation approach while retaining its advantages. While the weak universal relation approach is theoretically very appealing, it leaves open the question of how to efficiently compute the connection function.

Computational definitions. The other approach to the universal-relation model is that the user may query about any set of attributes X, and the system will perform some computation on the database relations to compute the connection on X. Almost all systems that support the universal-relation model take this approach, although they vary in the algorithm they use for the computation. A major issue in database theory is how well the various computational definitions (algorithms) approximate the semantical definitions.

Universal-relation systems

Two universal-relation database-management systems are good examples of the model: System/U,[5] developed at Stanford University, and FIDL,[6] developed at International Computers Ltd. in Britain. The examples here are adapted from articles about the two systems.[5,6] Other implementations are outlined in the box above.

System/U. System/U is implemented in C on Unix. The system's data model is based on the concepts of attributes, objects, and maximal objects.

Objects are minimal sets of attri

that have collective meaning; each object is assumed to be contained in one relation scheme. For example, in terms of the entity-relationship data model, an attribute or attributes that form a key for an entity set will be found in one object for each of the properties of that entity set; the object includes only the key and the property. Relationships are represented by objects consisting of the keys for the related entity sets.

Maximal objects are maximal sets of objects that let the system navigate. The following example illustrates the notions of objects and maximal objects.

The database here describes the operations of banks. The attributes are Bank, Acct (account), Bal (balance of account), Loan, Amt (amount of loan), Cust (customer), and Addr (customer's address). Customers are related to branches by being the owner of an account or the holder of a loan. Both accounts and loans can be shared by several customers, but each account is at a single branch. The database scheme is shown in Figure 1. The objects are Acct-Bal, Acct-Cust, Acct-Bank, Cust-Addr, Cust-Loan, Loan-Bank, and Loan-Amt.

In System/U, navigation follows dependencies between the attributes. In this example, Bank and Bal are functionally dependent on Acct. Every account is related to a unique branch and has funds in it. Similarly, Bank and Amt are functionally dependent on Loan. Finally, Addr is functionally dependent on Cust. These dependencies give rise to two maximal objects, shown in Figure 2. There is no basic relationship between, for example, Loan and Acct because these attributes do not occur in the same maximal objects.

Objects and maximal objects are crucial to the definition of connection in System/U. For example, in the banking example, we could ask

```
retrieve (BANK)
where CUST = "Jones"
```

System/U will first figure out what the relevant attributes to the query are; in this case, they are Bank and Cust. It will then figure out in what maximal objects these attribute appear; in this case, they appear in both maximal objects. Finally, the system will take the join of the objects in each relevant maximal object, project onto the relevant attributes, and take the union of the results to be the connection that the query is applied to. In this case, the effect will be to find all banks at which Jones has an account or a loan.

System/U has a data-definition facility that accepts as input declaration of attributes, relations, functional dependencies, and objects. It computes the maximal objects from this facility. After entering the declarations, the user asks the system to list all maximal objects. Figure 3a shows the input for the banking example; Figure 3b shows the system's response.

System/U's query language is essentially Quel, with the following important difference: There is no need for tuple variables because they all range over a virtual universal relation. Thus, an occurrence of attribute A is deemed to stand for $b.A$, where b is the blank tuple variable, a tuple variable that never appears explicitly. Of course the user can also use explicit tuple variables.

FIDL. FIDL, the Flexible Interrogation

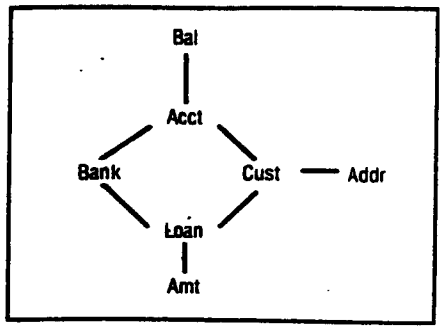


Figure 1. Banking example.

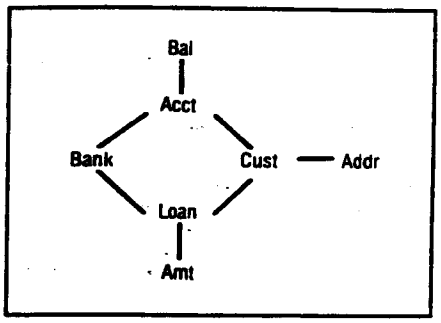


Figure 2. Maximal objects in the banking example.

```
                integer loan account;
                float amount balance;
                char [20] bank;
                char [25] customer;
                char [50] address;
                relation rcust = customer, address;
                relation rloan = customer, bank, loan, amount;
                relation racct = customer, bank, account, balance;
                object ocust in rcust = customer, address;
                object oloancust in rloan = customer, loan;
                object oloanbank in rloan = bank, loan;
                object oloanamt in rloan = loan, amount;
                object oacctcust in racct = customer, account;
                object oacctbank in racct = account, bank;
                object oacctbal in racct = account, balance;
                account->bank;
                account->balance;
                loan->bank;
                loan->amount;
(a)             customer->address;

                symbol: 2  type: maximal object
                objects: oloanbank oloanamt ocust oloancust
                symbol: 1  type: maximal object
(b)             objects: oacctbank oacctbal ocust oacctcust
```

Figure 3. (a) Input declarations for the banking example; **(b)** system response to the input.

Table 1.
Jobs example specification.

Relations	Represents
S[Sup-No:Town]	Suppliers and the supplier's town
P[Part-No:PD,QOH]	Parts, part description, and quantity on hand
J[Job-No:JD]	Jobs with the job description
PJ[Part-No, Job-No:QC]	Parts used by jobs and quantity committed
PS[Part-No, Sup-No:Price]	Catalog of suppliers with price
PJS[Part-No, Job-No, Sup-No:Qty]	Parts used on jobs and from which suppliers

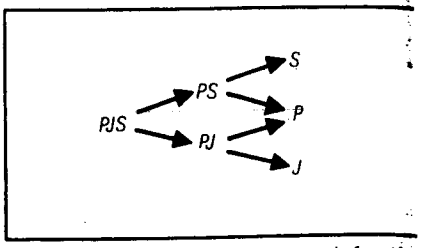


Figure 4. Implication network for the jobs example.

and Declaration Language, is a universal-relation system at International Computers Ltd. in Britain. In most universal-relation systems, the universal relation is a virtual relation, but FIDL is based on a physical implementation of the universal relation. The universal relation, called JNF (for Joint Normal Form), is stored on specialized hardware called CAFS, the Content-Addressable File Store. This store is designed to scan rapidly through a file and retrieve selected records and send them to the mainframe.

The system's data model is based on the concept of an implication network, which is a graphical description of the dependencies between the relations of the database. To obtain information that represents the actual database state, relations can be joined directly only with those lying on the same directed path in the implication network.

The jobs application shows which jobs (*J*) use which parts (*P*) and who supplies them (*S*). Table 1 shows the application's specification; Figure 4 shows the implication network for the jobs example. The result of taking the join of the relations *PS* and *PJ* shows which suppliers could provide parts for certain jobs, rather than the suppliers that actually do. This shows why joins between relations that are not on the same path should not be taken.

The implication network is specified by the database designer in a data-definition language. This specification is used by Flin, the Flexible Language Interpreter, to create a universal relation by performing

P[Part#:Part-Des]

1	NUT
2	BOLT
3	SCREW

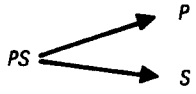
S[Supplier#: Name]

304	SMITH INC.
362	WESTLEY & MORTON
424	CRYOGENICS CO.

PS[Part# Supplier#:Price]

1	304	6
1	362	7
2	304	18
3	362	10

(a)



(b)

Figure 5. (a) Database for parts and supplies; **(b)** its implication network.

all possible joins of the application's relations. The relations are joined through common attributes along the arcs of the implication network. The tuples that are lost because of an absence of common values in shared attributes are restored by joining them to dummy tuples with null values in the unspecified attributes. The result is then appended to the universal relation, which is put on CAFS in a compressed form.

Consider the database for parts and suppliers shown in Figure 5a, whose implication network is shown in Figure 5b. The universal relation is the union of the join *P*S*PS*, a relation describing the parts that are not linked to suppliers, and a relation describing the suppliers not linked to parts (shown in Table 2).

FIDL lets the user query the database and modify it by inserting or deleting information. The user's commands are translated by Flin to CAFS commands. A sample query is

```
←LIST ALL ORDERS FOR WHICH
PART IS SCREW OR BOLT
```

Figure 6 shows the creation of a new relation that relates salesmen to customers. Figure 7 shows the deletion of a relationship.

Table 2.
Universal relation for the jobs example.

Header			Attributes				
P	*S*	*PS*	Part #	Part-Des	Supplier #	Name	Price
1	1	1	1	NUT	304	SMITH INC.	6
0	1	1	1	NUT	362	WESTLEY & MORTON	7
1	0	1	2	BOLT	304	SMITH INC.	18
1	0	1	3	SCREW	362	WESTLEY & MORTON	10
0	1	0	—	—	424	CRYOGENICS CO.	—

```
← CREATE CUSTOMER SALESMAN
    The user types in the requirements
ODB5 MAKE CUSTOMER SALESMAN
    The system responds by specifying the file accessed
THE ASSUMED RELATION IS CUS•SAL (CUSTOMER SALESMAN)
    The root relation is deduced
CUSTOMER ← 123456
    The essential key attributes are requested
SALESMAN ← 654321
  CHECKING CUS•SAL WHERE CUSTOMER = "123456" SALESMAN
= "654321"
NONE FOUND
    The tuple to be inserted does not exist and, since it contains no attributes
    of its own, none are requested
  CHECKING SAL WHERE SALESMAN = "654321". OK.
    The salesman is already recorded
  CHECKING CUS WHERE CUSTOMER - "123456". OK.
    The customer is already recorded
OK. 1 CREATED
    The complete JNF record, tagged and containing all implied informa-
    tion, is added to the ODB5 CAFS file
```

Figure 6. Example creation of a FIDL relation. Comments are italicized; user entries are in reverse video.

```
← DELETE ALL FOR EQUIP EQUALS "123450" OR EQUIP EQUALS "123457" OR EQUIP EQUALS "123465"
    "All" or some limit of JNF records to be deleted must be given as a
    precaution; the assumed limit is 1
ODB5 DELETE ALL FOR EQUIP EQUALS "123450" OR EQUIP
EQUALS "123457" OR EQUIP EQUALS "123465"
THE ASSUMED RELATION IS EQP (EQUIP : PMARK MACHINE
BAR)
    The EQP relation is deduced
THE DEPENDENT RELATIONS ARE CUS•EQP SYS•EQP
ORDAM}ITEM
    The affected relations are given
2 FOUND. DO YOU WISH TO CONTINUE. ← PLEASE OK
    The number of JNF records found to be deleted is given and, as a fur-
    ther precaution, some affirmative reply is required
OK. 2 DELETED
    The appropriate tags are removed in the JNF records
```

Figure 7. Example deletion of a FIDL relation. Comments are italicized; user entries are in reverse video.

The universal-relation model gives users a more succinct language for expressing queries, frees them from concern with the database's logical structure, and protects them from the errors that creep into queries when complicated access paths must be specified.

To be fair, these advantages do not come for free. The desire that certain logical navigation be done automatically by the system may place some subtle constraints on the data structure and may make unusual access paths harder to specify. Nevertheless, I believe that it is worth building universal-relation interfaces to existing database-management systems because such an interface is essential to several applications, such as natural-language interfaces.

The universal-relation model stands today where the relational model stood a decade ago. It is clearly defined, has robust theoretical foundations, and has several experimental implementations. Perhaps 10 years from now it will be as commercially successful as the relational model is today. ⊹

Acknowledgments

Figures 1-3 are copyright 1984 by ACM and are reproduced from the September 1984 issue of *ACM Transactions on Database Systems*. Figures 4-7 and Tables 1-2 are copyright 1982 by ACM and are reproduced from the June 1982 issue of *ACM Transactions on Database Systems*. All materials are reproduced with permission.

References

1. E.F. Codd, "A Relational Model for Large Shared Data Banks," *Comm. ACM*, June 1970, pp. 377-387.
2. D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville, Md., 1983.
3. D. Maier, J.D. Ullman, and M.Y. Vardi, "On the Foundations of the Universal-Relation Model," *ACM Trans. Database Systems*, Sept. 1984, pp. 283-308.
4. J.D. Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, Md., 1983.
5. H. Korth et al., "System/U: A Database System Based on the Universal-Relation Assumption," *ACM Trans. Database Systems*, Sept. 1984, pp. 331-347.
6. T.R. Addis, "A Relation-Based Language Interpreter for a Content-Addressable File Store," *ACM Trans. Database Systems*, July 1982, pp. 125-163.

Moshe Y. Vardi is a research staff member at the IBM Almaden Research Center, where he is researching system fundamentals. His research interests include database theory, protocol specification and verification, and knowledge theory in distributed systems and artificial intelligence. Before joining IBM in 1985, he was a research associate at Stanford University.

Vardi received a PhD in computer science from Hebrew University in Jerusalem. He is a member of ACM.

Address questions about this article to Vardi at IBM Almaden Research Center, MS K53/802, 650 Harry Rd., San Jose, CA 95120.