

In many commercial, governmental, and scientific data banks, however, some of the relations are of quite high degree (a degree of 30 is not at all uncommon). Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering *supplier*, then *part*, then *project*, then *quantity* in the relation *supply*). Accordingly, we propose that users deal, not with relations which are domain-ordered, but with *relationships* which are their domain-unordered counterparts.² To accomplish this, domains must be uniquely identifiable at least within any given relation, without using position. Thus, where there are two or more identical domains, we require in each case that the domain name be qualified by a distinctive *role name*, which serves to identify the role played by that domain in the given relation. For example, in the relation *component* of Figure 2, the first domain *part* might be qualified by the role name *sub*, and the second by *super*, so that users could deal with the relationship *component* and its domains—*sub.part super.part, quantity*—without regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary).³ Even this information might be offered in menu style by the system (subject to security and privacy constraints) upon request by the user.

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, nonsimple domain) and establish some links with terminology currently in use in information systems programming. In the remainder of this paper, we shall not bother to distinguish between relations and relationships except where it appears advantageous to be explicit.

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called *part* is defined on the following domains:

- (1) part number
- (2) part name
- (3) part color
- (4) part weight
- (5) quantity on hand
- (6) quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part

names, and part numbers are. We shall call the set of values represented at some instant the *active domain* at that instant.

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (*n*-tuple) of that relation. Such a domain (or combination) is called a *primary key*. In the example above, part number would be a primary key, while part color would not be. A primary key is *nonredundant* if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called *the* primary key of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such cross-references. We shall call a domain (or domain combination) of relation *R* a *foreign key* if it is not the primary key of *R* but its elements are values of the primary key of some relation *S* (the possibility that *S* and *R* are identical is not excluded). In the relation *supply* of Figure 1, the combination of *supplier, part, project* is the primary key, while each of these three domains taken separately is a foreign key.

In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the *supply* relation). This distinction is difficult to maintain when one may have foreign keys in any relation whatsoever. In the user's relational model there appears to be no advantage to making such a distinction (there may be some advantage, however, when one applies relational concepts to machine representations of the user's set of relationships).

So far, we have discussed examples of relations which are defined on simple domains—domains whose elements are atomic (nondecomposable) values. Nonatomic values can be discussed within the relational framework. Thus, some domains may have relations as elements. These relations may, in turn, be defined on nonsimple domains, and so on. For example, one of the domains on which the relation *employee* is defined might be *salary history*. An element of the salary history domain is a binary relation defined on the domain *date* and the domain *salary*. The *salary history* domain is the set of all such binary relations. At any instant of time there are as many instances of the *salary history* relation in the data bank as there are employees. In contrast, there is only one instance of the *employee* relation.

The terms *attribute* and *repeating group* in present data base terminology are roughly analogous to simple domain

² In mathematical terms, a relationship is an equivalence class of those relations that are equivalent under permutation of domains (see Section 2.1.1).

³ Naturally, as with any data put into and retrieved from a computer system, the user will normally make far more effective use of the data if he is aware of its meaning.