Introduction

In this paper we propose a systematic classification of the types of of associations that can occur between any two relations in a relational data base [9, 14]. Many types of association can occur in relational data bases [12, 13]. Nevertheless, although associations are fundamental in data base theory [2, 4, 8], and practice [20, 21, 27, 31, 32], there appears to have been no systematic attempt in the literature to classify them. There are probably two explanations for this neglect. First, the basic relational database manipulation languages, such as DSL Alpha [12, 13], SQL [10, 20, 21], QUEL [27, 28], and relational algebra [14, 15] can be used with any association type without any requirement of distinction or classification. A second explanation is that there is a significant body of theoretical data base research devoted to elimination of the need for navigation between relations, that is, of the need to specify associations between relations in any way in language expressions [19, 22, 23, 29]. This research is based on the idea of manipulating a universal relation formed from the underlying relations of the data base by means of relational algebraic join operations [1, 3, 14, 18]. A possible motive for the universal relation approach is that it has proven difficult to specify many important associations using the basic relational languages and dialects [6, 31].

In the commercial world, where nowadays the files of non relational data bases have become relational in structure, associations between relations are used extensively [5, 15, 30]. In addition, associations (called relationships) are an integral part of the entity/relationship approach [11]. They are also necessary with natural quantifier relational languages [6]. Consequently, there is reason to believe that associations between relations are important, so that there are good grounds for research directed towards a better understanding of them, and the development of techniques that make complex associations more amenable to use with database manipulation languages.

There exists a class of higher level relational data base languages that make use of associations in a manner that is far more concise than with conventional data base langauges. They do this by permitting the use of the wide range of natural quantifiers of natural languages. An example of this type of langauge is SQL/N [4, 6, 7], a research language that is upward compatible with SQL. Languages of this type require that associations be specified outside of language expressions, either in the schema, in host programs, or interactively, and it is research into such specifications that has led the author to the classification proposed in this paper.

1.1 Notation

We use upper case letters for attributes or attribute concatenations, except where otherwise stated. Upper case bold is used for relation names, and subscriped lower case for attribute values within a tuple. The relation **T**($\underline{T}$, U, V, W) may thus have a tuple (t_1, u_1, v_1, w_1). The primary key attribute (or any canditdate key attribute) is underscored, and for reader convenience will always have the same upper case letter as the relation name.

1.2 The association concept with relational data bases

No formal definition of an association between relations appears to have been proposed in the literature, although the term is used frequently. The following definition is consistent with the largely intuitive use of the concept to date.

Definition 1

Consider a data base with a set of relations [**A**, **B**, **C**, ...]. There is an association between any arbitrary pair of relations (**A**, **B**) of this data base iff, by a process of relational algebraic operations, involving any of the relations in [**A**, **B**, **C**, ...], it is possible to generate a relation **R**(A, B, ...), with A, B as minimal attributes, and where **R** is not necessarily a relation of the database. **R** must be non empty.

We refer to **R** as an association relation, and essentially this paper deals with a systematic classification of the kinds of **R** that can be generated from relational data bases.

1.3 Overview of major classification components

The main classification is into primitive and composite (non primitive) associations. Briefly an association between **A** and **B** is primitive iff it as the direct result of a join (whether equality-based or not) on attributes of **A** and **B** alone. The major components of the classification are thus:

Class 1 Primitive associations

(a) Functional, or simple 1:n

(b) Multivalued

(c) Equivalence, or attribute/relation

(d) Non equality based

Class 2 Composite associations

(a) Cascade, cascaded functional, or composite 1:n

(b) Matrix, or n:m

(c) Cascaded equivalence

(d) Obscure

Note that under Class 1b, we are dealing with multivalued associations, and not multivalued dependencies [16, 17, 24, 25, 26]. They are not the same thing, although there are some similarities. (It is because of the similarities that we have used the term multivalued for the association.) Most of the above association types can be further classified into cyclic and non cyclic versions, as will be analysed in later sections. Thus it is legitimate, in this context, to refer to a cyclic multivalued association.

2 CLASS 1 - PRIMITIVE ASSOCIATIONS

Before looking at the diferent kinds of primitive association in detail, we need more formal definitions of primitive and composite associations.

Suppose that we use the symbol Π for algebraic projection, and the symbol * for any arbitrary join operation, whether equality based or otherwise [1, 3, 14]. Consider any two relations **A, B** from a database.

Definition 2 Iff two relations **A, B** are such that $\Pi_{A,B}(\mathbf{A} * \mathbf{B})$ generates a non empty relation **R**(A,B), then the association described by **R** is primitive.

The requirement that **R** be non empty is a natural one, for if **R** is empty then it will not be possible to pair an **A** and **B** tuple, so that the relations cannot be associated. From this is should be clear that an association is not a constant property of a pair of relations **A**, and **B**, but rather of instances of these relations. Suppose that we are using an equijoin on attributes F in **A** and G in **B**. Suppose that with * as an equijoin $\Pi_{A,B}(\mathbf{A} * \mathbf{B})$ generates an non empty **R**. Sometime later, following update of the two relations, it is possible that G and F will be disjoint sets, so that (**A** * **B**) and consequently **R** will be empty. In other words, a simple update to the data base can eliminate, or create an association.

Definition 3 An association that is not primitive is composite.

Theorem 1 There will be a composite association between **A** and **B** if

$$\Pi_{A,B}(\mathbf{A} * \mathbf{C} * \mathbf{D} * \ldots * \mathbf{B}) = \mathbf{T}(A,B, \ldots)$$

where **T** is a non empty association relation.

Proof Since **T** exists and is non empty, there is an association between **A** and **B**, by Definition 1. Since T is not generated by a projection of a join of **A** with **B**, by Definitions 2 and 3, **T** must describe a composite association.

Corollary There is a composite association between relations **A** and **B** if **A** and **B** are at either end of a chain of primitive associations.

Proof Let **X** $\otimes$ **X** denote the equijoin of any relation **X** with X as the join attribute.

$\Pi_{A,B}(\mathbf{A} * \mathbf{C} * \mathbf{D} * \ldots \mathbf{M} * \mathbf{B})$ can be rewritten as:

$\Pi_{A,B}(\mathbf{A} * \mathbf{C} \otimes \mathbf{C} * \mathbf{D} \otimes \mathbf{D} * \ldots \mathbf{M} \otimes \mathbf{M} * \mathbf{B})$, which can be rewitten as:

$\Pi_{A,B}[(\Pi_{A,C}(\mathbf{A} * \mathbf{C})) \otimes (\Pi_{C,D}(\mathbf{C} * \mathbf{D})) \ldots \otimes (\Pi_{M,B}(\mathbf{M} * \mathbf{B}))]$

which forms a chain of primitive associations, by Definition 2.

Theorem 2 There is a primitive association between **A** and **B** iff **A** and **B** each have a non disjoint attribute drawn on the same domain.

Proof If each of **A** and **B** has a common domain, then a join operation is allowed, otherwise it is not allowed. Since the join attributes are not disjoint, the computation $\Pi_{A,\ B}(\mathbf{A} * \mathbf{B})$ will yield a non empty relation **R**(A, B).

The requirement that **R**(A, B) be non empty for an association to exist is necessary, as explained above. However, in practice it is usually expedient and safe to ignore it. If **A** and **B** are significant relations with common domain attributes, without knowing the contents of these relations it is usual to assume that the common domain or join attributes will not be disjoint, so that **R** will be non empty. This tacit assumption will be made in most of the remainder of this paper, so that the requirement for a non empty **R** in practice can be replaced by the less stringest requirement that the operation $\Pi_{A,B}(\mathbf{A} * \mathbf{B})$ simply be allowable. This appears to be from most writers mean when they discuss an association in a relational data base. The possibility that **R** could be empty, in addition, is simply ignored.

In the light of the above discussion, in practice Theorem 2 can be usefully restated as:

There is a primitive association between any relations **A** and **B** iff **A** and **B** have attributes drawn on a common domain.

It is this somewhat flawed version of the theorem that appears to be assumed by most writers in dealing with associations between relations. However, readers should note that there is a fundamental distinction between primitive and composite associations. We now examine the different kinds of primitive associations.

2.1 Functional associations

In the foregoing, whenever we use the term attribute, it is to be assumed that the term attribute concatenation could also apply. A value of a concatenation of P and Q is simply the value of P concatenated to the value of Q.

Consider a pair of attributes from **A, B** drawn on a common domain.

Definition 4 If at least one of the common domain attributes in a primitive association is either a primary or a candidate key, that is, if one attribute at least has only unique values, then the association is functional (or simple one-to-many (1:n)).

This means that if A is one of the common domain attributes, and the other one is some non key attribute G in **B**, then the association relation R(A,B) is equal to $\Pi_{A,B}(\mathbf{A} *_{A,G} \mathbf{B})$, where $*_{A,G}$ denotes an equijoin on the attributes $_{A,\ G}$. Since G values are not unique in G in **B**, it follows from the mechanism of an equijoin that B values are unique, so that given a B value in an **R** tuple the A value is determined (but not vice versa). Accordingly **R** implies a function $r{:}B \rightarrow A$. It follows that the association between **A** and **B** is functional in the mathematical sense as well. Alternatively, for one **A** tuple there may be zero or more associated **B** tuples. Associations of this type are well known and are among the most common encountered in practice.

As we move to more complex associations, diagrammatic technique are useful for display purposes. The functional association between **A** and **B** is displayed in Figure 1, using a directed graph. A node shows a relation with its attributes, and the association is depicted by an edge that is directed from the key attribute A to the other join attribute G. This type of graphical depiction, and the arrow direction, is based more on tradition than anything else [4, 5, 15, 28, 31], but at least has the advantage of pointing out the attributes on which the association rests, as well as indicating which of the two possible functions $r:A \rightarrow B$, $r:B \rightarrow A$ applies.

The association can also be usfully depicted using a co-ordinate system, and the association relation **R**(A,B). The B values are assumed laid out along the x-axis (Figure 2) in ascending G value order, that is, the function implicit in **R**(A,B) is assumed derived from $\Pi_{A,B}(\mathbf{A} *_{A,G} \mathbf{B}^G)$, where $\mathbf{B}^G$ is simply **B** with the tuples sorted in ascending G value order.

A trivial case of the functional association occurs when G is also a (candidate) key attribute, so that G values, as well as A values, are unique. In such a case the function $r:B \rightarrow A$ implicit in **R**(A,B) is simply a trivial one-to-one function, not included in the classification as a distinct association type.

2.2 Multivalued associations

Another type of primitive association between any two relations **A**, **B** occurs when neither of the two attribute in the pair with a common domain are candidate key attributes.

Definition 5 A primitive assocaition between any two relations **A**, **B**, is multivalued, iff neither of the common domain attributes supporting the association is a primary or candidate key.

In other words, the association is supported by a pair of attributes, each with values are that are not unique. This type of association has similarities with the much investigated concept of a multivalued dependency.

Lemma 1 The association relation **R** for a multivalued association between any two relations **A**, **B** is partitioned by the values of the attributes supporting the association

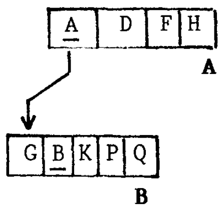

Figure 1

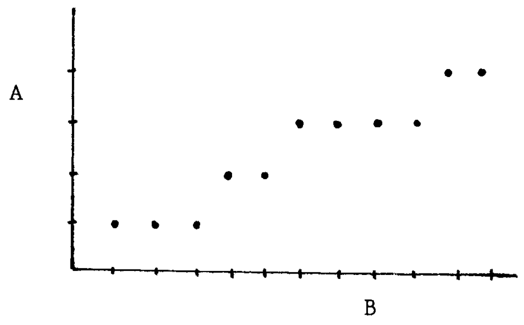

Figure 2

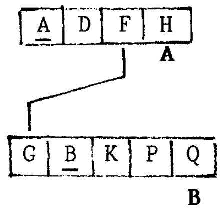

Figure 3

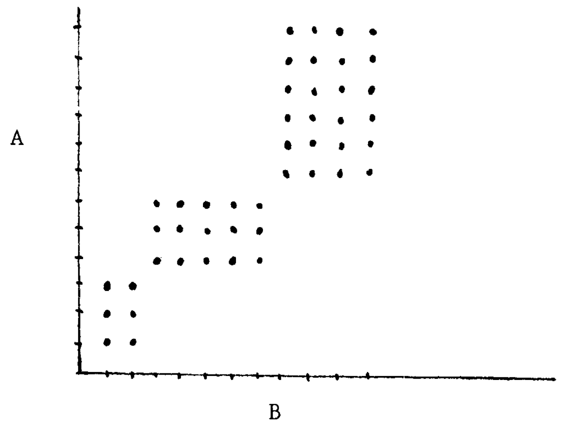

Figure 4

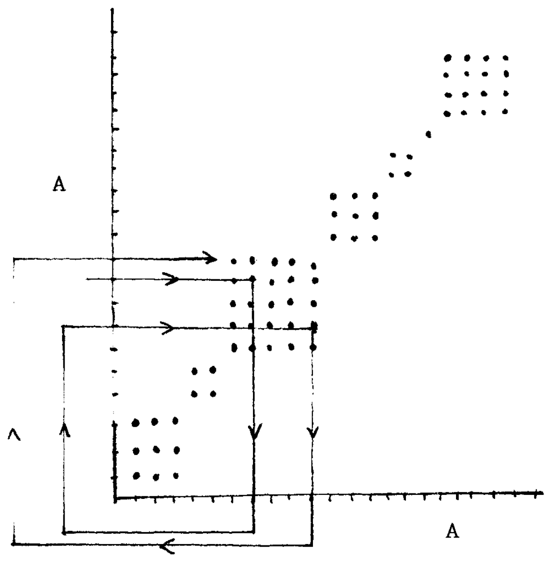

Figure 5

Proof Suppose that F in **A**, and G in **B** are the attributes supporting the association. Let [a_1, a_2 ... a_i] be the set of A values in all those **A** tuples with a F value g_k, and similarily [b_1, b_2, ... b_j] is the set of B values in all those **B** tuples with an equal G value g_k.

If we now take $\Pi_{A,B}(\mathbf{A} *_{F,G} \mathbf{B})$, then in order to form the association relation **R**(A, B), each of the set of A values [a_1 ... a_i] must be paired with each of the set of B values [b_1, ... b_j] to give us a relation [(a_1,b_1), (a_1,b_2), ... (a_i,b_j)] that is a subset of **R**(A,B). No other tuples of **R** can contain either [a_1, ... a_i] values or [b_1, ... b_j] values, so that **R** must contain a subset of tuples for each value common to both F and G, that is, for each member of the set $\Pi_F\mathbf{A} \cap \Pi_G\mathbf{B}$. It follows that each common F and G value gives rise to a block of **R** tuples, and thus a partition of **R**.

Theorem 3 For a multivalued association between relations **A** and **B**, within a block of the partition of **R**: Iff tuples (a_1, b_1) and (a_2, b_2) exist, then tuples (a_1, b_2) and (a_2, b_1) also exist.

Proof By Lemma 1, the attribute value supporting the association between **A** and **B** tuples within a block of a partition of **R** is a constant g_k. Accordingly, there exists **A** tuples (a_1, ... g_k), (a_2, ... g_k) and **B** tuples (b_1, ... g_k), (b_2, ... g_k). If we apply the operation for generating the association relation **R** to these tuples, namely $\Pi_{A,B}(\mathbf{A} *_{F,G}\mathbf{B})$, then we must get tuples (a_1, b_2) and (a_2, b_1) in **R**.

This is reminiscent of the condition for a multivalued dependency. In fact, if we add the common F and G attribute to **R**, then this ternary relation will contain a multivalued dependency, as proved in [7]. In otherwords, if we rename the F attribute in **A** as G, and form the relation $\mathbf{R_m}$ from $\Pi_{A,G,B}(\mathbf{A} *_G \mathbf{B})$, where $*_G$ denotes a natural join on the G attribute, then it can be shown that for $\mathbf{R_m}$ the following condition holds:

"Iff tuples (a_1, g_k, b_1) and (a_2, g_k, b_2) exist, then tuples (a_1, g_k, b_2) and (a_2, g_k, b_1) also exist."

This is the well known condition for a multivalued dependency.

Returning to the association relation **R** for a multivalued association between relations **A** and **B**, supported by non key attributes F and G, a graphical display, as in Figure 3, will later be found to be useful with more complex associations. Because of the defining condition for **R**(A, B), there does not exist an implicit function $r:A \rightarrow B$ or $r:A \rightarrow B$. However, if we order the tuples of **R** so that tuples of the same partition block are adjacent, and use a coordinate system to display $\Pi_A(\mathbf{R})$ versus $\Pi_B(\mathbf{R})$, we get an interesting geometry of rectangles (Figure 4). Each rectangle represents a block of the partition of **R**, and a point within a rectangle represents a pair of associated tuples.

Example Each tuple of **A** describe a carrier based aircraft, identified by A. The attribute F in **A** identifies the carrier on which the aircraft is based. Each tuple of **B** identifies a crew member of a carrier, and G in **B** identifies the carrier to which the crew member is assigned. If crew-member b_4 and aircraft a_2 are respectively assigned to and based on the same carrier g_k, then they are associated, as are the tuples describing them. The association is clearly not as "strong" as that of a functional association, but is clearly important nevertheless.

2.3 Equivalence association

An equivalence association is essentially an implicitly cyclic association with no corresponding non cyclic version. An association is cyclic if is between two identical relations.

Definition 6 A primitive association between two identical relations is an equivalence association iff a common attribute in each of the two relations supports the association, and that attribute is neither a primary nor candidate key.

<u>Theorem 4</u> The association relation **R**(A, A) for an equivalence association of the relation **A** supported by the attribute G is an equivalence relation, such that there is a block of a partition of the relation for each G value.

<u>Proof</u> We have $\mathbf{R}(A, A) = \Pi_{A,A}(\mathbf{A} *_G \mathbf{A})$

Suppose that we have the set of A values $[a_1, \dots a_i]$ for a given G value g_k. Then in forming **R** tuples, each of the set of A values $[a_1, \dots a_i]$ is paired with every member of the set, to give the relation:

$[(a_1,a_1), (a_1, a_2), \dots (a_i, a_i)]$

which is a subset of **R**(A, A). No other tuples can contain $[a_1, \dots a_i]$ values. It follows that each G value gives rise to a block of **R** tuples and thus a partition of **R.** It follows that **R** is an equivalence relation.

The equivalence association is really the special case of a multivalued association between two identical relations (**A, A**) on a common attribute (G). Although it is not usual to display an equivalence relation on a cordinate system, it is intructive to do so in this case, as it gives a geometrical perspective on the difference between multivalued and equivalence associations.

We order the tuples of the equivalence relation **R**(A, A) so that tuples with A values from **A** tuples with the same G value lie adjacent, and then display A versus A on the cordinate system (Figure 5). We see that we get a system of squares, with each square representing a block of the partition of **R.** A point with a square represents a pair of tuples from **A.** The cyclic nature of the association should also be apparent. If we take a given tuple with a block of the partition (or point in a square of the co-ordinate display), such as (a_2,a_5), then we can find another tuple (a_5, a_3), and another (a_3, a_9), and so on in an undending sequence, as illustrated in Figure 5. Note, however, that given the partition of **R**, the sequence remains within a block of the partition.

2.4 Cyclic functional associations

Definition. Where **B** = **A** in a functional association between **A** and **B**, then the association is cyclic functional.

In order to be able to distinguish the relations in a cyclic functional association, let us refer to the two associated relations as **A** and $\mathbf{A}_G$, where the association is supported by the attribute A in **A**, and G in $\mathbf{A}_G$. The primary key attribute in $\mathbf{A}_G$ is then A_G in conformance with the convention we are using for keys.

The association relation $\mathbf{R}(A, A_G)$ is then generated from

$$\pi_{A,\ A_G}(\mathbf{A} *_{A,\ G} \mathbf{A}_G)$$

and this association relation contains the function $r{:}A_G \rightarrow A$. Since there can be many associated $\mathbf{A}_G$ tuples for a single **A** tuple, the association is often said to be 1:n in nature.

The association can be displayed as a directed graph, as in Figure 6a, where we draw two (identical apart from the name) versions of **A**, and in Figure 6b, where we use only one node (with two names **A**, and $\mathbf{A}_G$). It is also useful to display the association relation using a cordinate system, as in Figure 7. Here we assume that **R** was generated, using the expression above, using a version of $\mathbf{A}_G$ where the tuples are arranged in ascending or descending G order.

If we take any A value a_k, then, because of the functional nature of the association, there will in general exist a set of **R** tuples $[(a_k, a_1), (a_k, a_2), \ldots (a_k, a_j)]$, indicating a number of associated $\mathbf{A}_G$ tuples for any given **A** tuple. But for any of these $\mathbf{A}_G$ tuples, such as that identified by a_2, for example, if regarded as an **A** tuple, then in turn it will be further associated with a further set of $\mathbf{A}_G$ tuples, and so on in cyclic fashion, as is also illustrated in Figure 7. Note that if the association relation **R** contains the tuples (a_1, a_7) and (a_7, a_{13}), then the **A** tuple identified by a_1 will be associated

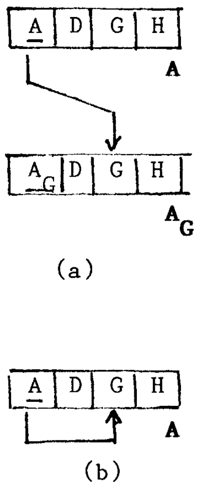

Figure 6

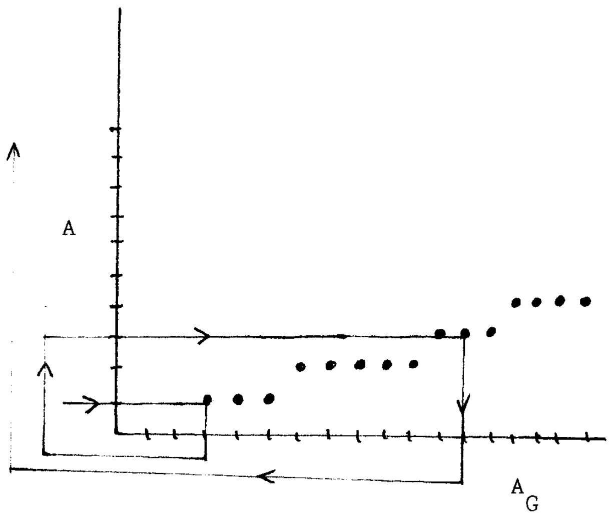

Figure 7

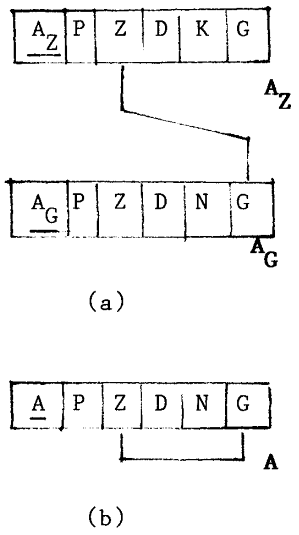

Figure 8

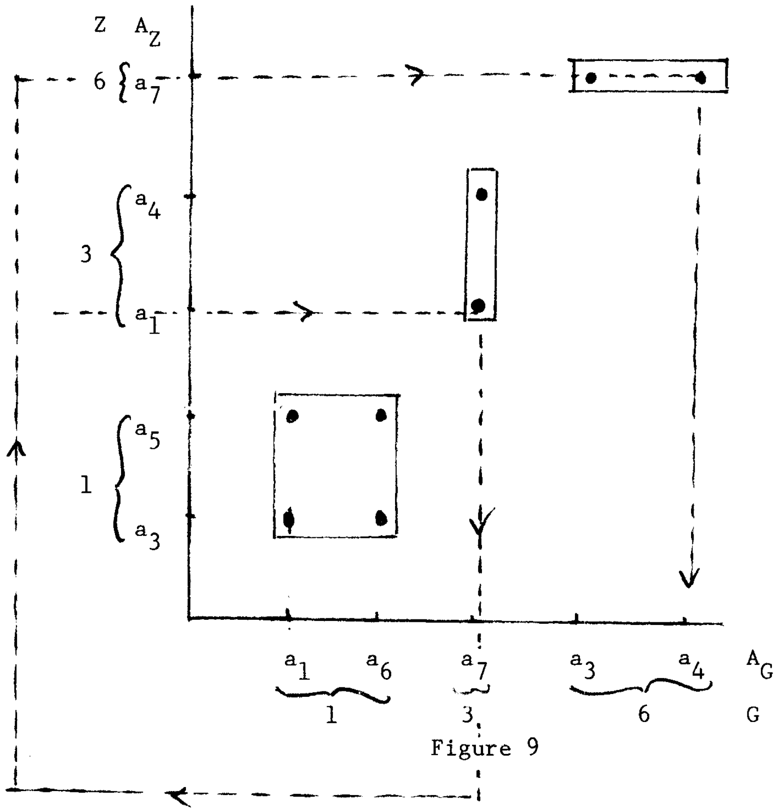

Figure 9

with the **A** tuple identified by a_{13}. However this association is not primitive, since the tuple (a_1, a_{13}) is not a member of $\mathbf{R}(A, A_G)$ generated from $\Pi_{A,A_G}(\mathbf{A} *_{A,G} \mathbf{A}_G)$. This tuple can be generated from:

$$\mathbf{R}(A, A_G) *_{A,A_G} \mathbf{R}(A, A_G)$$

confirming that the association involved (or sub or outer association) is non primitive, that is, it is composite. This is an admitted flaw in our so far "clean" classification of associations into primitive and composite associations. However, it is the only one, and is relatively minor. The difficulties resulting from other classifications attempted by the author are far greater.

Example The classic example is where an **A** tuple describes an employee in an organization, A gives the employee's identifying number and G give the identifying number of the employee's immediate superior. An employee is thus primitively (cyclic functionally) associated with his immediate superior, as are the corresponding tuples, but is compositely associated with his immediate superior's superior, as are the corresponding tuples.

2.5 Cyclic multivalued associations

Definition 8 When **B** = **A** in a multivalued association between **A** and **B**, then the association is cyclic multivalued, provided the attributes supporting the association are not the same.

It should thus be clear that we have a cyclic multivalued association between two identical relations **A, A** when the association is supported by two non key attributes Z and G drawn on the same domain. To distinguish the relations for analysis purposes we refer to the relations as $\mathbf{A}_Z$ and $\mathbf{A}_G$. An $\mathbf{A}_Z$ tuple is cyclic multivalued associated with an $\mathbf{A}_G$ tuple if the Z value in the $\mathbf{A}_Z$ tuple has the same value as the G value in the $\mathbf{A}_G$ tuple. The association relation **R** is formed:

$$\mathbf{R}(A_Z, A_G) = \Pi_{A_Z,A_G}(\mathbf{A}_Z *_{Z,G} \mathbf{A}_G)$$

From emma 1 it follows that **R** is also partitioned, with a block of **R** tuples for each common value of Z and G. From Theorem 3, within a block of the partition of **R**, that is, for a single value z_k of both Z and G, the following condition holds:

"Iff tuples (a_{z1}, a_{g1}) and (a_{z2}, a_{g2}) exist, then so do tuples (a_{z1}, a_{g2}) and (a_{z2}, a_{g1})."

And, despite the symmetry of the situation, if (a_{z1}, a_{g1}) is in a block of a partition of **R**, (a_{g1}, a_{z1}) is not necessarily in **R**. The existence of (a_{z1}, a_{g1}) means that the Z value of the **A** tuple identified by a_{z1} equals the G value of the **A** tuple identified by a_{g1}; however, there is no reason for the Z value of the tuple identified by a_{g1} to be also equal to the G value of the tuple identified by a_{z1}, the condition for the existence of (a_{i1}, a_{g1}) in **R**. This is best seen by the example of **A** and **R** below, which will also be useful again later for illustrating some further properties of this association.

A	Z	G
a_1	3	1
a_3	1	6
a_6	9	1
a_5	1	6
a_7	6	3
a_4	3	6

A, $\mathbf{A}_Z$, $\mathbf{A}_G$

A_Z	A_G
a_3	a_1
a_5	a_1
a_3	a_6
a_5	a_6
a_1	a_7
a_4	a_7
a_7	a_3
a_7	a_4

R(A_Z, A_G)

It should be clearly understood that a tuple (a_5, a_1) in **R**, for example, means that the Z value of the a_5 tuple in **A**, equals the G value of the a_1 tuple in **A**. In the example above, **R** has three partition blocks whose tuples all satify the existence rule given above, which followed from Theorem 3.

The association is displayed graphically in Figure 8a, 8b. If the tuples of **R** are arranged so that all tuples belonging to the same block of the partition are adjacent, as in the example above, then when we use a co-ordinate system to display $\Pi_{A_Z}(\mathbf{R})$, or A_Z, versus $\Pi_{A_G}(\mathbf{R})$, or A_G, we once more get a system of rectangles (Figure 9), each rectangle representing a block of the partition of **R**.

The cyclic nature of the association is also displayed in Figure 9. Suppose that in **R** we have a tuple (a_5, a_1), as in the example above, within a particular block of the partition of **R**, or rectangle (Figure 9, beginning line linking points), then it is possible that there will exist a further tuple where the A_Z value is a_1, such as (a_1, a_7) in the example above, and then a tuple (a_7, a_4), and so on. This chain of associated tuples will not in general remain within the original block of the partition or rectangle, as shown in Figure 9, since Z and G values within a single **A** tuple are not necessarily the same, and will indeed normally be different. This leads to an interesting theorem.

<u>Theorem 5</u> A chain of associated **R** tuples in the association relation **R** of a relation **A** participating in a cyclic multivalued association, may be finite iff the set difference $\Pi_Z(\mathbf{A_Z}) - \Pi_G(\mathbf{A_G})$ is non empty.

<u>Proof</u> Suppose that $\Pi_z(\mathbf{A_Z}) - \Pi_G(\mathbf{A_G})$ is empty. Suppose that we have an arbitrary tuple (a_p, a_q) from **R**. There must therefore exist an **A** tuple with a G value equal to the Z value of the tuple identified by a_q. If this new **A** tuple is identified by a_r, then there exists an **R** tuple (a_q, a_r). In a similar manner there must exist an **R** tuple (a_r, a_s), and so on, so that the chain is infinite.

Now suppose that $\Pi_Z(\mathbf{A_Z}) - \Pi_G(\mathbf{A_G})$ is non empty, and that we have an arbitrary tuple (a_p, a_q). This time we cannot guarantee that there exists an **A** tuple with a G value equal to the Z value of the tuple identified by a_q, because, since the set of Z values in **A** does not match the set of G values in **A**, the Z value of the **A** tuple identified by a_q may not exist in the set of G values. Accordingly, a chain may be finite.

As an example, using the tuples of the instance of **R** given earlier, the chain

$(a_3, a_1); (a_1, a_7); (a_7, a_3); (a_3, a_6); (a_6, ?).$

is finite, since the **A** tuple identified by a_6 has a Z value (9) that does not occur as a G value. Removal of this tuple from **A** would make the sets of Z and G value equal, so that all chains would be infinite.

Cyclic multivalued associations occur reasonably frequently in data bases, although the associations are not normally very meaningful in practice.

Example A tuple of relation **A** describes a stolen car; A identifies a car, Z gives the (U.S.) state in which the car was stolen, and G gives the state of the dealership that **originally sold the car.** A car a_5 stolen in state 1 is associated with car a_6 originally sold in state 1. The association will frequently be little more than coincidental [7], although in police or other kinds of investigations the assocaition may be very meaningful.

2.6 Non equality based primitive associations

Clearly there can be many associations of this type. In general the association can be defined by an association formula $\emptyset$, that specifies the conditions the attributes must satisfy before two tuples are associated. Such associations do not appear to have much practical value, nor do they appear to have been much investigated. Preliminary work [7] indicates that they lead to generalized dependencies [17, 18, 24, 25, 26].

2.7 Equivalence associations and attribute/relation associations

In an equivalence association, all tuples in a relation **A** with common G values are associated. An equivalent functional association can be constructed from an equivalence assocaition, as described in [6]. Essentially a new relation T(G) is constructed equal to $\Pi_G(\mathbf{A})$, so that there is a functional association between **T** and **A**, supported by the primary key attribute G in **T** and the non key attribute G in **A.** This association between **T** and **A** exists because of the implicit functional dependence of G on A in **A**, so that given a A value in **A**, G is determined. Consequently, given an A value in **A**, a G value in **T** is also determined. This functional association between **T** and **A** has been called an attribute/relation association, and it has been shown that SQL/N expressions can be used with this type of association [6]. The equivalence between the equivalence association and the attribute/relation association becomes self evident, when it is considered that two **A** tuples are associated in an equivalence association if they have a common G value, while those same two tuples must each be associated to a common **T** tuple.

3 CLASS 2 - COMPOSITE ASSOCIATIONS

As stated by Theorem 2, a composite association between two relation **A** and **B** will involve a chain of primitive associations. Since we can have different kinds of primitive associations in the chain, different kinds of composite associations are possible.

3.1 Cascade associations

<u>Definition</u> There is a cascade association between relations **A** and **B**, with an implicit function $r:B \rightarrow A$, iff there exists a non empty set of relations $[\mathbf{C}_1, \mathbf{C}_2, \ldots \mathbf{C}_n]$, with primitive functional associations:

between **A** and $\mathbf{C}_1$, with implicit function $r_1:C_1 \rightarrow A$,

between $\mathbf{C}_1$ and $\mathbf{C}_2$, with implicit function $r_2:C_2 \rightarrow C_1$,

...

between $\mathbf{C}_{n-1}$ and $\mathbf{C}_n$, with implicit function $r_n:C_n \rightarrow C_{n-1}$, and

between $\mathbf{C}_n$ and **B**, with implicit function $r_{n+1}:B \rightarrow C_n$.

Cascade associations are quite simple in concept, are very common, and almost always meaningful. Readers can no doubt envisage many common examples. The association relation **R**(A, B) will clearly be given by:

$\mathbf{R}(A, B) = \Pi_{A,B}(\mathbf{A} * \mathbf{C}_1 * \mathbf{C}_2 * \ldots * \mathbf{C}_n * \mathbf{B})$

with the attributes supporting the individual functional associations as join attributes.

The cascade association is also a 1:n association, since each of the (primitive) functional associations is also a 1:n association. The association can be depicted by a directed graph, where any pair of connected nodes denotes a functional association, as in Figure 10.

3.2 Matrix associations

Definition 10 There is a matrix association between **A** and **B**, when there exists a relation **M**, such that there is either a functional or cascade association between:

(a) **A** and **M**, with implicit function $r{:}M \rightarrow A$, and

(b) **B** and **M**, with implicit function $s{:}M \rightarrow B$.

Matrix associations are well known, and are often called many-to-many or n:m associations. The Supplier-Parts matrix association due to Date [15] is almost the classic example. The assoociation is fundamentally n:m, since for a given **A** tuple there must be many associated M tuples and thus many associated **B** tuples, and the converse.

It is clear that we can distinguish between matrix associations where the associations between **A** and **M**, and between **B** and **M** are merely functional, and the many types where one or both of these underlying associations is a cascade association. This is illustrated by the graphs in Figures 11a and 11b. When we use a co-ordinate system to display the association the result is a matrix, no matter how the $\Pi_A(\mathbf{R})$, $\Pi_B(\mathbf{R})$ are ordered.

Unlike other associations so far discussed, the association relation for a matrix association is a ternary relation of the form **R**(A, M, B). An association relation of the form **R**(A, B) is insufficient. The difficulty arises as follows. Suppose that we have associated **A** and **B** tuples, identified by a_1

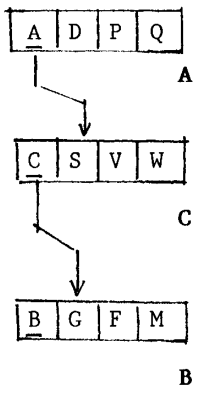

Figure 10

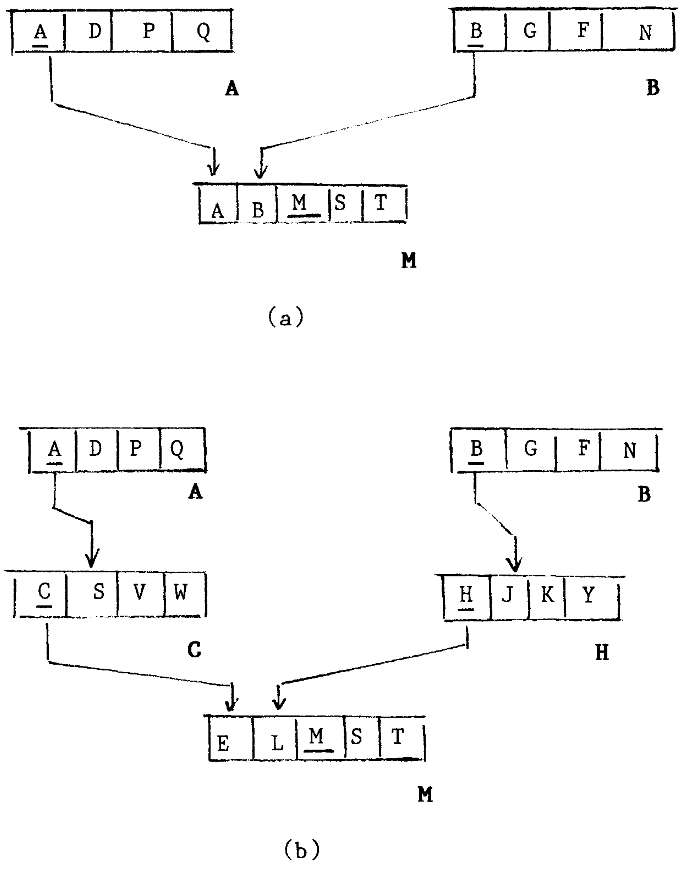

Figure 11

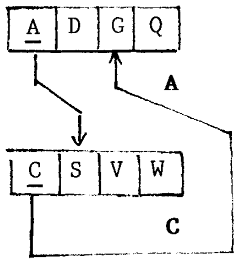

Figure 12

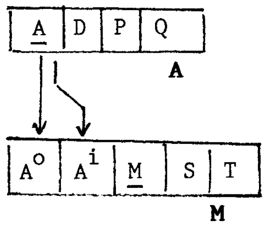

Figure 13

and b_1. For the a_1 tuple there can be many functional or cascade associated **M** tuples, for example, those identified by m_1, m_2, m_3, and m_4. Similarily, for the b_1 tuple in **B**, there can be many associated **M** tuples, for example those identified by m_2, m_3, m_4, m_5, and m_6. Consequently, since the intersection of these two sets of **M** tuples are the tuples identified by m_2, m_3, and m_4, it follows that the a_1 tuple in **A** is matrix associated with the b_1 tuple in **B** in three distinct ways, via m_2, via m_3, and via m_4. Each of these associations can be significantly diferent, and will need to be identified, so that the association relation **R** must contain the tuples:

$$(a_1,\ m_2\ b_1)$$
$$(a_1,\ m_3,\ b_1)$$
$$(a_1,\ m_4,\ b_1)$$

<u>Example</u> Suppose that an **A** tuple describes a supplier (of industrial part types), that a **B** tuple describes an industrial part type, and that an **M** tuple describes a shipment of a part by a supplier. Then supplier a_1 can ship part type a_1 in shipment m_2, and in shipment m_2 and in shipment m_3. There are thus three associations of supplier a_1 with part type b_1 and these are physically distinct and significant, and must be distinguished.

Because there can be more than one association between a given pair of **A** and **B** tuples, the term matrix association is something of a misnomer. In mathematics, given a matrix X_{nm}, there will be but one value of X_{nm} for given values of the n and m matrix coordinates (corresponding to A and B values), whereas we can have many associated **M** tuples, for given **A** and **B** tuples.

Obtaining the association matrix **R**(A, M, B) is a simple matter when there are only primitive functional associations between **A** and **M** and between **B** and **M**. Clearly, we must have:

$$\mathbf{R}(A,\ M,\ B) = \P_{A,M,B}\ (\mathbf{M})$$

In the case of cascade associations between either **A** and **M**, or between **B** and

M, or both, the generation is more complex. The method requires a sequence of joins, beginning with **A** and ending with **B**, followed by a projection on A , M and B, or:

$$\mathbf{R}(A, M, B) = \pi_{A,M,B}(\mathbf{A} * \mathbf{L_1} * \ldots * \mathbf{L_n} * \mathbf{M} * \mathbf{R_m} * \ldots \mathbf{R_1} * \mathbf{B})$$

The extensive computing needed to generate the assocaition relation for cases like this has an important corrollary for language designers. Suppose that we are dealing with a conventional SQL system, and have a retrieval expression involving the association between **A** and **B**, where the association is a matrix association involving cascade associations in the association chain. The entire chain will have to be specified in SQL, and where the chain takes the form **A**, $\mathbf{L_1}$, $\mathbf{L_2}$, **M**, $\mathbf{R_2}$, $\mathbf{R_1}$, **B**, with an easily expressed retrieval request:

> Retrieve the A value for each **A** tuple with a D attribute value 4, where most (matrix) associated **B** tuples have a P value of 10.

we need the following lengthy SQL expression:

```
    SELECT A  FROM A, XA
    WHERE D = 4
    AND [SELECT COUNT(*) FROM B
         WHERE P = 10 AND
          B IN [SELECT B  FROM R_1
                 WHERE R_1 IN [SELECT R_1 FROM R_2
                     WHERE R_2 IN [SELECT R_2 FROM M
                         WHERE L_2 = [SELECT L_2 FROM L_2
                             WHERE L_1 = [SELECT L_1 FROM L_1
                                 WHERE A = XA.A]]]]]]
          >
          [SELECT COUNT(*) FROM B
           WHERE P ≠ 10 AND
           B IN [SELECT B FROM R_1
```

```
                    WHERE R_1 IN [SELECT R_1 FROM R_2
                      WHERE R_2 IN [SELECT R_2 FROM M
                        WHERE L_2 = [SELECT L_2 FROM L_2
                          WHERE L_1 = [SELECT L_1 FROM L_1
                             WHERE A = XA.A]]]]]]
```

In addition, each time a query involving this type of association is submitted a computation needed for generating an association relation **R**(A, D, M, P, B), must be carried out. Since we have: **R**(A, D, M, P, B)

$$= \Pi_{A,D,M,P,B}(\mathbf{A} * \mathbf{L}_1 * \mathbf{L}_2 * \mathbf{M} * \mathbf{R}_2 * \mathbf{R}_1 * \mathbf{B}) ,$$

with the above SQL expression equivalent to:

```
    SELECT A FROM R, XR
      WHERE D = 4
        AND [SELECT COUNT(*) FROM R
             WHERE P = 10
             AND A = XR.A]
             >
            [SELECT COUNT(*) FROM R
             WHERE P ≠ 10
             AND A = XA.A]
```

the computation will be lengthy. (It is here that the universal relation approach has something to offer [19, 23], since the required result is also obtianed from the short SQL expression above with the universal relation formed by a join of **A**, $\mathbf{L}_1$, $\mathbf{L}_2$, **M**, $\mathbf{R}_2$, $\mathbf{R}_1$ and **B** substituted for **R**(A, D, M, P, B).)

An alternative is the SQL/N approach, with an expression close in syntax to the original English-language expression:

```
        SELECT A FROM [EACH] A  [TUPLE]
        WHERE D = 4
        AND FOR MOST  M-ASSOCIATED B [TUPLES] (P = 10)
```

Here words surrounded by [] can be omitted, FOR MOST is a natural quantifier, and M-ASSOCIATED is a schema specified name or mode [7] for the association relation

R(A, M, B) for the matrix asscciation between **A** and **B.** Execution of this SQL/N expression can be made very fast if **R** is stored in the storage data base. Otherwise **R** will have to be computed, a lengthy process, using information in the schema that specifies exactly what **R** is.

3.3 Cascade equivalence associations

Definition 11 The association between two identical versions of a relation **A** is a cascade equivalence association if there is a relation **E** participating in an equivalence association, where, in addition, there is a cascade or functional association between **E** and **A**, with implicit function $r: A \rightarrow E$.

In other words, two **A** tuples, identified by a_1 and a_2, are cascade equivalence associated if the **E** tuple the a_1 tuple is associated with has the same Q value as the **E** tuple the a_2 tuple is associated with, where Q is the attribute supporting the equivalence association in **E.** Cascade equivalence associations are thus quite straightforward. When displayed using a co-ordinate system, we get a sequence of squares, as with an equivalence association. Note that in such a display, the A values must be ordered according to:

$$\Pi_A (\mathbf{A} * \mathbf{R}_1 * \mathbf{R}_2 * \ldots * \mathbf{R}_n * \mathbf{E})$$

with **E** ordered in ascending or descending Q order.

The association relation **R**(A, A) is clearly given by:

$$\Pi_{A,A}([\mathbf{A} * \mathbf{R}_1 * \ldots \mathbf{R}_n * \mathbf{E}] *_Q [\mathbf{E} * \mathbf{R}_n * \ldots * \mathbf{R}_1 * \mathbf{A}])$$

Example The Forestry data base [5] contain an illustrative example. Suppose that we have three relations **E,** $\mathbf{R}_1$ and **A.** An **E** tuple describes a forest, and Q in **E** gives the owner of the forest (not a candidate key.) An $\mathbf{R}_1$ tuple describes a tree in a forest, and an **A** tuple a measurement on a tree. Two cascade equivalence associated **A** tuples describe measurements carried out, not necessarily on the same trees, nor on trees in the same forest but in forests with the same owner.

The association can also be viewed in terms of attribute/relation associations (see Section 2.7 and [6]). In this view we would consider $\Pi_Q(\mathbf{E})$ to be cascade associated with **A**.

3.4 Cyclic cascade associations

Definition 12 When $\mathbf{B} = \mathbf{A}$ in a cascade association between **A** and **B**, the association is cyclic cascade.

The assocaition is very similar to the simpler cyclic functional association (Section 2.4). To be able to distinguish the two identical relations, we refer to them as **A** and $\mathbf{A_G}$, where the association is supported by A in **A** and G in $\mathbf{A_G}$. The primary keys are then A in **A** and A_G in $\mathbf{A_G}$.

The association relation $\mathbf{R}(A, A_G)$ is then generated from:

$$\Pi_{A,A_G}(\mathbf{A} *_{A,G_1} \mathbf{R_1} *_{R_1,G_2} \mathbf{R_2} * \cdots \mathbf{R_n} *_{R_n,G} \mathbf{A_G})$$

where in relation $\mathbf{R_p}$, R_p is the primary key attribute, and G_p is an attribute drawn on the same domain as the primary key attribute R_{p-1} of $\mathbf{R_{p-1}}$. The assocaition has thus an implicit function $r{:}A_G \rightarrow A$. It is depicted as a directed graph in Figure 12.

A given **A** tuple identified by a_k will be associated with many **A** (that is, $\mathbf{A_G}$) tuples, and each of these in turn with many **A** tuples, exactly as with cyclic functional associations.

Unfortunately, while of some interest theoretically, genuine cyclic cascade associations (not contrived) are rare indeed in practice, so the utility of this type of association is marginal at best.

3.5 Cyclic matrix associations

Definition 13 When $\mathbf{B} = \mathbf{A}$ in a matrix association between **A** and **B**, the association is cyclic matrix.

Cyclic matrix associations, in contrast to cyclic cascade assocaitions, are very important in practice, but only those cases where the associations between **A** and **M** are functional. The author has encountered no useful examples of a cyclic matrix association involving cascade associations between **A** and **M**.

Using $\mathbf{A}^o$ and $\mathbf{A}^i$ to distinguish two otherwise identical instances of **A**, for the practical case where there are only (primitive) functional associations between **A** and **M**, the association relation **R** is given by:

$$\mathbf{R}(A^o, M, A^i) = \Pi_{A^o,M,A^i} (\mathbf{A}^o * \mathbf{M} * \mathbf{A}^i)$$
$$= \Pi_{A^o,M,A^i} (\mathbf{M}(A^o, M, A^i, \ldots))$$

A unique feature of this association is that it generates two different cascades of associated tuples, for any given **A** tuple. By convention, in the direction $\mathbf{A}^o$ to $\mathbf{A}^i$, this cascade of associated relations is called an <u>explosion</u>, and in the opposite direction an <u>implosion</u>.

An explosion is generated as follows. Given an **A** tuple, identified as a^o_1, it will be associated with many $\mathbf{A}^i$ tuples, identified by the set of keys $[a^i_1, a^i_2, \ldots a^i_n]$. If we now consider each of these $\mathbf{A}^i$ tuples as a $\mathbf{A}^o$ tuple, then in turn each of these will be associated with many $\mathbf{A}^i$ tuples, and so on in a cascade of associations. Conversely, if we considered that original **A** tuple as an $\mathbf{A}^i$ tuple, identified by a^i_1 (so that a^o_1 and a^i_1 are identical), then there are many $\mathbf{A}^o$ tuples associated with this tuple, and, taking these $\mathbf{A}^o$ tuples as $\mathbf{A}^i$ tuples, there are many further $\mathbf{A}^o$ tuples associated with them, and so on in a cascade. This is the implosion. Figure 13 gives a directed graph of the association.

<u>Example 1</u> Each tuple of **A** describes a part type, assembly type, subassembly type, subsubassembly type, and so on. Each tuple of **R** shows how an A^o (outer) identified part encloses or contains an A^i (inner) identifed part type. The explosion of a part type gives the part types it contains, the part types those part types contain, and so on. Conversely, the implosion of a part type give the part types that contain it, the part types containing those part types and so on.

Example 2 Each tuple of **A** describes a joint stock company. Each tuple of **M** shows how a A^o identified (parent) company owns stock in an A^i identified company (immediate subsidiary). For a given company, an explosion shows the immediate subsidiaries, their immediate subsidiaries, and so on. Conversely, an implosion for a given company gives the immediate parents, their parents, and so on.

Cyclic matrix associations are difficult to manipulate using conventional non procedural languages, such as SQL. The major difficulty is the large number of implicit sub-associations involved. For example, in the case of the joint stock companies, we can be interested in the association between a company and its immediate subsidiaries, which is one type of subassociation, or between a company and the subsidiaries of its immediate subsidiaries, which is another type of subassociation, and so on. The conventional languages require that the user fully specify, in terms of matching attribute specifications, any such subassociation within a language expression, which can be very difficult. A full account of this matter is beyond the scope of this paper, which concentrates on the the classification of associations. In a separate report [8], a complete analysis of these subassociations is carried out, together with a demonstation of the ease of applicability of natural quantifier languages, such as SQL/N.

3.6 Other composite associations

Many diferent types of association chains between relations **A** and **B** are possible, but the author has found no others that are likely to be of any use in practice. We are inclined to call all of them obscure associations. Neverthless, a few of these obscure assocaitions may be of theoretical interest. For examples, it is possible to define composite multivalued associations, and even cyclic composite multivalued associations. Prelimary investigations strongly suggest no practical application. These associations will be analysed in a separate report.

4 CONCLUSIONS

The number of different types of associations that can occur between relations in a relational data base is surprizingly large, and attests to the great flexibility of the relational approach. In the CODASYL approach [5], even where the conceptual files are relations, only two types of association are allowed, namely the equivalent of the functional and cyclic functional associations, although it has to be admitted that functional associations are easily the most common, and are the building blocks of many of the other types defined in this paper.

The classification proposed in this paper is not exhaustive, in that in that it is possible that further useful classification of both the primitive non-equality based associations and the composite obscure associations will prove useful in the future. However, we believe that the present classification should be of use in practice to designers of data base manipulation languages and systems, particularily SQL/N-like systems. But even those concentrating on the universal relation approach could find the classification useful. It may well be the case that absolute universal relation manipulation languages will prove unweildy, or even impossible, so that proponents will settle for an approach based on an appropriate collection of subuniversal relations. Such subuniversal relations could be designed so that association relations (**R**) defined in this paper are obtainable from them by single elementary projection operations.

REFERENCES

1. Aho, A. V., Beeri, C., and Ullman, J.D. The theory of joins in relational databases, ACM Trans. Database Syst., 4(3), 1979, 317-314.

2. Armstrong, W.W. Dependency structures of database relationships, Proc. IFIP 74, North Holland, Amsterdam, 1974, 580-583.

3. Bernstein, C.W., and Chiu, C.W. Using semijoins to solve relational queries, J. ACM, 28(1), 1981, 25-40.

4. Bradley, J. An extended owner-coupled set data model and predicate calculus for database management, ACM Trans. Database Syst., 3(4), 1978, 385-416.

5. Bradley, J. File & Database Techniques, Holt, Rinehart & Winston, New York, 1982.

6. Bradley, J. SQL/N and attribute/relation assocaitions implicit in functional dependencies, Int. J. Computer & Information Science, 12(2), 1983

7. Bradley, J. SQL/N and modes of association in relational databases, Research Report No. 84/143/5, Univ. of Calgary, Calgary, Alberta, Canada, 1984, 39 pages.

8. Bradley, J. Application of SQL/N and association modes to data bases containing recursive associations, Research Report No. 85/202/15 Unin. of Calgary, Calgary. Alberta, Canada, 1985, 22 pages.

9. Chamberlin, D.D. Relational database management systems. Comput. Surv., 8(1), 1976, 43-66.

10. Chamberlin, D.D., et al. SEQUEL 2: A unified approach to data definition, manipulation and control, IBM J. Res. & Dev., 20(6), 1976, 560-575.

11. Chen, P.P., The entity-relationship model: Towards a unified view of data, ACM Trans. Database Syst. 1(1), 1976, 9-36.

12. Codd, E.F. Further normalization of the database relational model, In "Database Systems", Courant Computer Science Symposium, 6, R. Rustin, Ed., Prentice-Hall , Englewood Cliffs, N.J., 1971, 33-74.

13. Codd, E.F. Relational completeness of database sub-languages, In "Database Systems", Computer Science Symposium 6, R. Rustin, Ed., Prentice-Hall, Englewood

Cliffs, N.J., 1971, 65-98.

14. Codd, E.F. Relational database: A practical design for productivity, CACM, 25(2), 1982, 109-117.

15. Date, C.J. Introduction to database systems, Addison-Wesley, Reading, Mass., 1981.

16. Fagin, R. Multivalued dependencies and a new normal form for relational databases, ACM Trans. Database Syst., 2(3), 1977, 262-278.

17. Fagin, R., Mendelzon, A.O., and Ullman, J.D. A simplified universal relation assumption and its properties, ACM Trans. Database Syst., 7(3), 1982, 343-360.

18. Fagin, R. Horn clauses and database dependencies, J. ACM 29(4), 1982, 952-985.

19. Forth, H. et al. System U: A database based on the universal relation assumption, ACM Trans. Database Syst. 9(3), 1984, 331-347.

20. Kim, W. On optimizing an SQL-like nested query, ACM Trans. Database Syst., 7(3), 1982, 443-469.

21. Kim, W., Gajski, D., Kuck, D.J. A parallel piplined relational query processor, ACM Trans. Database Syst., 9(2), 214-242.

22. Maier, D., et al, Towards logical data independence: A relational query language without relations. Proc 1982 Int. Conf. on Management of Data, Orlando, Fla., June 1982, ACM, New York, 51-60.

23. Maier, D., Ullman, J.D., Vardi, M.Y., On the foundations of the universal relation model, ACM Trans. Database Syst. 9(2), 1984, 283-309.

24. Sadri, F, Ullman, J. D.. Templete dependencies: a large class of dependencies in relational data bases and its complete axiomatization. J. ACM 29(2), 363-372.

25. Sagiv, Y., et al. An equivalence between relational database dependencies and a fragment of propositional logic, J. ACM 28(3), 1981, 435-453.

26. Sagiv, Y. and Walecka, S.F. Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies, J. ACM 29(1), 1982, 363-372.

27. Stonebraker, M., Wong, E., Kreps, P., and Held, G. The design and implementation of INGRES, ACM Trans. Database Syst., 1(3), 1976, 189-222.

28. Ullman, J.D., Principles of Database Systems, Computer Science Press, Rockville, MD, 1983.

29. Wald, J.A., and Sorenson, P.G. Resolving the query inference problem using Steiner trees, ACM Trans. Database Syst. 9(3), 348-368, 1984.

30. Weiderhold, G. Database Design, McGraw-Hill, New York, 1983.

31. Welty, D. and Stemple, D.W. Human factors comparison of procedural and non procedural query languages, ACM Trans. Database Syst., 6(4), 1981, 626-649.

32. Wilmot, R.B. Foreign keys decrease adaptibility of database designs, CACM, 27(12), 1984, 1237