A Survey of Relational Database Management Systems for Microcomputers

by

Brian E. Smith

1984

A Survey of Relational Database Management Systems for Microcomputers

This paper investigates the relational features of several commercially available database management systems for microcomputers. The relational algebra, as described by ALDAT, is introduced to establish a common frame of reference for the analysis of the individual systems. MRDSA, a relationally complete database management system based on ALDAT, is presented as a model of a working DBMS implemented on a microcomputer. Eight commercial systems are examined in detail with particular emphasis on the nature and variety of the operations of the relational algebra offered by each system. While we find that each DBMS implements a subset of the relational operators, we also note the addition, in a rather ad-hoc fashion, of non-relational procedures. In this sense we feel that the power and flexibility of the relational algebra is not being used to full advantage in most commercial systems.

This research was undertaken in the School of Computer Science at McGill University.

ACKNOWLEDGEMENT

I wish to thank Professor T.H. Merrett of the School of Computer Science at McGill University for his patience and encouragement during many discussions about the contents of this paper.

TABLE OF CONTENTS

1. INTRODUCTION

Database management systems have been successfully implemented on mainframe computers for some time and are firmly established as an integral part of the applications software in the business environment. In this paper interest focuses on two relatively recent developments in this area. On the one hand we have seen the implementation of relational database management systems following the theoretical model first proposed by Codd in 1970. On the other hand there are now several systems available for use with microcomputers, thus making this powerful data processing tool readily accessible for small business and personal applications. This paper is concerned with the implementation of relational database systems on microcomputers with particular emphasis on the operations on relations which are provided by the various systems.

As a general guideline for the investigation of relational DBMSs we briefly recall the original discussion by Codd (1970). At the most elementary level a relation is defined to be a two dimensional table of data items. Thus any system which stores data as sets of fixed-length records may claim to be a relational database system. However Codd also defines a relational algebra which uses relations as the primitive units of data and provides a set of operations for manipulating relations. The relational algebra was originally defined in terms of two types of operations: the traditional set theoretic operations (Cartesian product, union, intersection, difference) and the special relational operations restriction (now usually called selection), join, projection and division.

In the remainder of this paper we shall understand a relational DBMS to refer to a system which not only stores its data in the appropriate tabular form, but also implements (a subset of) the relational algebra. Since our objective is to examine database systems primarily with respect to their relational features, we omit a discussion of systems which manipulate files and records, however successfully, using implementations which are clearly non- relational i.e. do not employ explicit operations drawn from the relational algebra. While such systems may be excellent single or multifile management systems they fall outside the domain of interest of the present study.

In order to relate the ensuing discussion to the usual file oriented terminology we note that a relation corresponds to a file with unique fixed-length records. Relations are two-dimensional tables in which a row (or tuple) corresponds to a record and a column (domain or attribute) corresponds to a field. The uniqueness property ensures that a relation cannot contain identical tuples.

We compare the relational operators provided by a selection of commercially available microcomputer database management systems with the relational algebra as extended by ALDAT (Merrett,1977). This provides us with a general framework and a theoretical yardstick against which to measure the extent and versatility of the implementation of the relational algebra in specific systems. In addition we shall describe the implementation embodied in MRDSA (Merrett and Chiu,1983) , a research system at McGill University on the APPLE II microcomputer using APPLE (UCSD) Pascal. MRDSA is not a commercial DBMS and makes no claim to user friendliness e.g. there is no English-like query language built into the system at the present time. However MRDSA is a fully relational DBMS supporting an impressive variety of commands in the relational algebra as well as an interactive relational editor and a single relation query facility. In the hands of a competent programmer MRDSA is a powerful and versatile data management system. ALDAT establishes a theoretical framework while MRDSA exemplifies a working system which is faithful to the relational philosophy.

The subsequent chapters are devoted to a review of eight commercially available DBMSs for microcomputers: Logix, Condor, dBASE II, RQL, RMS, Sequitur, MicroRim and RL-1. Due to the lack of uniformity in terminology and syntax we have found it necessary to establish a consistent scheme for describing the features of the systems. For instance we shall always use the standard relational terminology: database, relation, attribute, tuple. Table 1.1 shows the alternative names used in the various systems. It is also necessary to employ, as far as possible, a common syntax in describing the systems. Thus, for example, each system provides a method of specifying a condition for tuple selection; in one case it might be through a "where" clause and in another by using an "if" statement. While we retain the actual keywords employed by the individual systems, we note that in most cases the condition itself is a fairly general tuple selection condition permitting algebraic comparison of numeric attribute values, string comparisons and boolean combinations. We will adopt the convention that "clause" always refers to a condition of this type; for those systems which differ markedly from the norm the differences will be documented in the text.

The following meta-syntactical notation will be used:

{ }	- indicates a mandatory choice
[ ]	- indicates an optional choice
*	- indicates repetition zero or more times
R,S,T,..	- are relation names
A,B,C,..	- are attribute names.

In addition we note that keywords will be written in lower case and underlined; names of relations and attributes will always be upper case.

The concepts of key fields and sorting are handled differently in the various systems. Some general comments on these features are appropriate here. Sorting is not a relational operation - a sorted relation, i.e. a relation in which the tuples are ordered on a specified attribute (or set of attributes), has exactly the same tuples as the original unsorted version; thus, in terms of the relational algebra, sorting is merely the identity operation. Nevertheless, the end-user will want to view the relation in a specific order: this requirement may be satisfied by (1) implementing relational commands such as project and join with automatic sorting on specified or default fields (2) having an explicit sort command which permits the user to sort on specified fields prior to printing, or (3) embedding the sort routine in output commands such as list, print, display. As we shall see all three methods are used in practice. Notwithstanding the formally trivial nature of sort commands we will indicate how each system handles the sorting requirement, since it is closely related to the nature of the implementation and is a measure of the degree to which a system adheres to the relational philosophy.

In the relational context a key is defined to be a minimal subset of the attributes of a relation which can be used to identify uniquely each tuple. A key is minimal in the sense that if we omit any attribute of the key we can no longer uniquely identify each tuple, whereas if we include an extra attribute the resulting subset of attributes is no longer considered to be a key, even though it clearly provides a unique identification of tuples. This clarification of the concept of a relational key is necessary because several commercial systems use the word "key" in the context of sort key i.e. a favoured field (usually indexed) on which sorting can be done most efficiently. The concept of a favoured attribute is a clear violation of the relational model.

Database Terminology

MRDSA :	Database	Relation	Tuple	Attribute
Logix :	Database	Relation	Item	Column
Condor :		Database	Record	Data item
dBASE II :		Database or File	Record	Field
RQL :	Database	Table	Row	Column
RMS :	Database	Table	Row	Column
Sequitur :	Database	Table	Row	Column
MicroRIM :	Database	Relation	Row	Attribute
RL-1 :	Database	Relation	Row	Column

Table 1.1

2. ALDAT

2.1 INTRODUCTION

The objective of ALDAT is to provide as much power, convenience and flexibility as possible within the framework of the relational algebra. The facilities provided are described by Merrett (1984) and Merrett and Chiu (1983). We review them briefly here, introducing ALDAT syntax and emphasizing the simplicity of the conceptual foundations.

In order to illustrate the nature of relational algebra procedures in ALDAT, consider the following example of a relation which summarizes delivery schedules for a local trucking company.

The relation is

DELIVERY(DELNO,SUPPLIER,DEST,PRODUCT,QTY,UPRICE).

Each order, or delivery, is labelled with a unique number (DELNO) and consists of shipping goods from SUPPLIER to DEST. A delivery may involve several products, the quantities and unit prices of the goods being labelled QTY and UPRICE respectively.

DELIVERY(

DELNO	SUPPLIER	DEST	PRODUCT	QTY	UPRICE)
4	Adams	Brown	Eggs	20	1.20
6	Green	Daly	Apples	62	0.40
1	Fingal	O'Neill	Eggs	26	1.20
3	Green	Daly	Pears	42	0.50
5	Cowan	Stanley	Pears	31	0.50
4	Adams	Brown	Apples	50	0.40
6	Green	Daly	Pears	17	0.50
1	Fingal	O'Neill	Apples	23	0.40
2	Adams	White	Pears	35	0.50

Table 2.1.1

2.2 UNARY OPERATIONS

Projection creates a new relation by including only specified attributes i.e it takes all tuples in a relation and projects them on a given list of attributes. In ALDAT notation

ROUTE <- DELNO, SUPPLIER, DEST in DELIVERY

yields the new relation

ROUTE(	DELNO	SUPPLIER	DEST)
	4	Adams	Brown
	6	Green	Daly
	1	Fingal	O'Neill
	3	Green	Daly
	5	Cowan	Stanley
	2	Adams	White

Table 2.2.1

Note that ROUTE contains only six tuples since duplicates are eliminated in accordance with the uniqueness property of relations.

For future reference consider the projection

GOODS <- DELNO, PRODUCT, QTY, UPRICE in DELIVERY

GOODS(	DELNO	PRODUCT	QTY	UPRICE)
	4	Eggs	20	1.20
	6	Apples	62	0.40
	1	Eggs	26	1.20
	3	Pears	42	0.50
	5	Pears	31	0.50
	4	Apples	50	0.40
	6	Pears	17	0.60
	1	Apples	23	0.40
	2	Pears	35	0.60

Table 2.2.2

The relation GOODS identifies products, quantities and unit prices for each delivery.

While projection creates a new relation by restricting the attributes (or columns) to be included, selection works by selecting tuples (or rows) from a relation according to a specified condition. In ALDAT the condition is realized as a clause between the key words "where" and "in". For example

FRAGILE <- where PRODUCT = "Eggs" in DELIVERY

FRAGILE(	DELNO	SUPPLIER	DEST	PRODUCT	QTY	UPRICE)
	4	Adams	Brown	Eggs	20	1.20
	1	Fingal	O'Neill	Eggs	26	1.20

Table 2.2.3

The condition may be any logical expression which can be evaluated to true or false on any single tuple of the relation. ALDAT envisages completely general tuple conditions including comparison of attribute values and boolean combinations.

Combining projection and selection we obtain the T-selector, as illustrated:

```
EGGDEL <- SUPPLIER,DEST,QTY where PRODUCT="Eggs" in DELIVERY
```

EGGDEL(

SUPPLIER	DEST	QTY
Adams	Brown	20
Fingal	O'Neill	26

)

Table 2.2.4

The "T" in T-selector stands for "tuple" and signifies that each tuple in the original relation contains all the information necessary to determine whether or not the tuple will be included in the result relation. The T-selector is a special case of the more general QT-selector (Q for quantifier) which is the basis of ALDAT's single relation query system. The QT-selector is discussed briefly in section 3.2 (page 19); we do not dwell on it since a detailed discussion is not required for the present analysis. It is permitted to nest T-selectors. For example

```
where PRODUCT = "Eggs" in (SUPPLIER, DEST, QTY in DELIVERY)
```

is equivalent to EGGDEL.

The general notation for the T-selector is

```
A,B,C,... where condition in R
```

where A,B,C... is a subset of the attributes of relation R.

2.3 RELATIONAL EDITOR

The relational editor provides the user with the ability to change the data in individual tuples i.e. to add, delete, or update selected tuples. Any or all of the attribute values in the tuple may be changed at the user's discretion. This facility is essential to the end-user for whom each tuple, or file record, is the primary object of concern. The programmer-user, on the other hand, sees the initial and final relations as the basic units of data; to him or her the relational editor is simply an operation of the relational algebra which converts a source relation into a result relation. A unary relational operator is provided to allow the programmer-user to maintain this strictly relational point of view.

2.4 GLOBAL UPDATE

The relational editor permits the end-user to update tuples one at a time. However, applications may require that several tuples be updated at once; for example a customer may wish every instance of her maiden name to be changed to her married name. In this case a batch or global update facility is called for. ALDAT provides the ability to update sets of tuples according to specified criteria as a variant of the QT-selector (of which the T-selector is a special case). Briefly, the QT-selector incorporates quantifiers and adds a single relation query facility to the relational algebra. Using QT-expressions, the user can request the names of all types of widgets (the universal quantifier) or professors who teach at least one graduate course (the existential quantifier) and so forth.

Many of the commercial systems we have investigated provide some sort of batch update and some also include a posting operation which allows one to update one relation from the data in another; ALDAT does not incorporate this type of (binary) update.

2.5 BINARY OPERATIONS

In its simplest form the join command refers to the "natural join" which consists of concatenating tuples from two source relations when the tuples have the same value for a common attribute (or group of attributes). In ALDAT the natural join is denoted by ijoin. For example

DELIVERY <- ROUTE ijoin GOODS

reconstructs the relation DELIVERY by forming the natural join of ROUTE and GOODS (tables 2.1.1, 2.2.1, 2.2.2). The join is understood to be on those tuples from the two source relations which have the same value of the common attribute DELNO. ALDAT syntax allows the common attribute(s) to be specified explicitly e.g.

DELIVERY <- ROUTE (DELNO ijoin DELNO) GOODS.

In general, for binary relations R(A,B), S(B,C) and T(D,E), the natural join is defined by

$$R \text{ ijoin } S = \{(a,b,c) : (a,b) \in R \text{ and } (b,c) \in S\},$$

or

$$R(B \text{ ijoin } D)T = \{(a,b,d,e): (a,b) \in R \text{ and } (d,e) \in T \text{ and } b=d\}$$

Note that A,B and C can be groups of attributes so that the

definitions given here are quite general. The second form is required if the join is done on distinct attributes from the input relations.

In the above example each DELNO is associated with one and only one SUPPLIER - DEST pair and with a set of products. The natural join associates each such pair, via DELNO, with the set of PRODUCTs. In a more general setting the join attribute may be associated with a set of items in each of the two source relations. In the following example each office has two telephone lines, the usual extension (a four digit number) and an intercom number (two digit) connecting the office to the main switchboard:

OCCUPANTS(OFFICE, PROF)

OFFICE	PROF
606	Smith
606	MacKenzie
815	White
815	Black
815	Green
423	Jones
423	Adams

TELEPHONE(OFFICE, LINE)

OFFICE	LINE
606	5821
606	29
815	4823
815	31
522	5120
522	24

Table 2.5.1

Then the natural join

PROFDATA <- OCCUPANTS ijoin TELEPHONE

yields the result relation

PROFDATA(PROF, OFFICE, LINE)

PROF	OFFICE	LINE
Smith	606	5821
Smith	606	29
MacKenzie	606	5821
MacKenzie	606	29
White	815	4823
White	815	31
Black	815	4823
Black	815	31
Green	815	4823
Green	815	31

Table 2.5.2

Another type of join, though best avoided in practice, is the Cartesian product R x S of two relations R and S which is formed by concatenating every tuple of S to every tuple of R. In the case of databases with large relations it is evident that the Cartesian product will result in very unwieldy relations and will have a correspondingly slow execution time. Nevertheless some commercial systems implement join as a Cartesian product

followed by tuple selection on an appropriate logical condition. For instance with input relations R(A,B) and T(D,E), the T-selector

where B = D in R x T

is equivalent to the natural join

R(B ijoin D)T.

The natural join is, in fact, a generalization of set intersection (hence the "i" in ijoin). For relations R(A,B) and S(A,B) i.e. relations with identical attributes, the natural join R ijoin S is precisely the intersection of the set of tuples in R with the set of tuples in S. Other set operations (union, symmetric difference etc.) can also be generalized to form relational operators in a similar fashion. To facilitate the discussion of the various types of joins obtained in this way consider the two relations AGE and INCOME :

AGE(

NAME	YEARS)
B	41
F	27
G	32
A	19
C	24
D	36
E	21

INCOME(

NAME	SALARY)
B	28
G	18
P	26
A	14
R	36
C	20
D	40
E	25

Table 2.5.3

where SALARY is recorded to the nearest thousand dollars. The natural join AGEINC is given by

AGEINC <- AGE ijoin INCOME

AGEINC(

NAME	YEARS	SALARY)
B	41	28
G	32	18
A	19	14
C	24	20
D	36	40
E	21	25

Table 2.5.4

Note that AGEINC omits those persons for whom either YEARS or SALARY is unknown. If all available information is required in a single relation it could be obtained by using the union join (ujoin)

DETAIL <- AGE ujoin INCOME

DETAIL(

NAME	YEARS	SALARY)
B	41	28
F	27	*
G	32	18
A	19	14
C	24	20
D	36	40
E	21	25
P	*	26
R	*	36

Table 2.5.5

where an asterisk denotes a missing value. We note that the union join consists of three distinct types of tuples. For input relations R(A,B) and S(B,C) we identify the three types as follows:

Type 1: tuples from R which match no tuple from S

Type 2: tuples which belong to the natural join R ijoin S

Type 3: tuples from S which match no tuple from R.

The following table summarizes the joins which are obtained as generalizations of set theoretic operations.

TYPE OF JOIN	ALDAT NOTATION	RESULT	EXAMPLE
intersection	R ijoin S	The natural join Type 1 only	AGEINC (table 2.3.4)
union	R ujoin S	The union join Types 1,2 and 3	DETAIL (table 2.3.5)
left	R ljoin S	The natural join augmented by tuples from R which match no tuple from S Types 1 and 2	All but the last two tuples of DETAIL
right	R rjoin S	The natural join augmented by tuples from S which match no tuple from R Types 2 and 3	All tuples of DETAIL except (F,27,*)
left difference	R dljoin S	All tuples from R which match no tuple from S Type 1 only	(F,27,*)
right difference	R drjoin S	All tuples from S which match no tuple from R Type 3 only	(P,*,26) (R,*,36)
symmetric difference	R sjoin S	All tuples from either R or S which do not match any tuple in the other relation Types 1 and 3	(F,27,*) (P,*,26) (R,*,36)

Table 2.5.6

Since there is no need to have both left and right difference joins (one can be derived from the other by interchanging the positions of R and S) we limit ourselves to the "difference join" djoin defined by

R djoin S = R dljoin S.

The family of joins defined here is collectively called the "μ-joins".

Another family of operations, called the "σ - joins" is defined in terms of "set selectors", which are a generalization of relational division (Codd 1972).Since operations of this type are rarely implemented in commercial database systems we restrict this discussion to the division operator. Consider the relations R(A,B) and S(B):

R(A	B)
x	1
y	1
z	1
x	2
y	2
t	2
y	3
t	3
x	4
t	4

S(B)
2
3

Table 2.5.7

In R, each value of A is associated with a set Ra of values of B. In the example

$Rx = \{1,2,4\}$ $Ry = \{1,2,3\}$ $Rz = \{1\}$ $Rt = \{2,3,4\}$

Since S is a set of values of attribute B we can make set comparisons between Ra and S, such as $Ra \supseteq S$, $Ra = S$ etc. Then relational division $R \div S$ is defined by

$$R \div S = R \supseteq S = \{(a):Ra \supseteq S\} .$$

In the example

$$R \div S = \{y,t\}$$

since $Ry = \{1,2,3\} \supset \{2,3\} = S$

and $Rt = \{2,3,4\} \supset \{2,3\} = S.$

To interpret division, suppose that attribute A refers to the names of professors while B gives course numbers. Then $R(A,B) \div S(B)$ yields the names of all professors who teach all of the courses specified in S. In fact $Q \leftarrow R \div S$ gives the relation

Q(A)
y
t

.

In ALDAT division is extended by defining relations using other set comparisons in an analogous manner. For future reference we note the following notation:

Ra ⍝ S is equivalent to Ra ∩ S = ∅
Ra ⍛ S is equivalent to Ra ∪ S = U

where U is the universe.

The remaining family of joins, the "θ - joins" can be derived by selecting tuples from the Cartesian product R x S of two relations. We have already observed that

where B = C in R x S

is equivalent to the natural join R(B ijoin C)S and it is called the "equi-join" in this context. The "less-than" and "greater-than" joins are defined analogously by replacing "=" by "<" and ">" respectively in the T-selector. For the example relations

R(A	B)
a	50
b	10
c	20

S(C	D)
30	x
15	y
10	z

Table 2.5.8

R x S = P

A	B	C	D
a	50	30	x
a	50	15	y
a	50	10	z
b	10	30	x
b	10	15	y
b	10	10	z
c	20	30	x
c	20	15	y
c	20	10	z

Table 2.5.9

so that

where B < C in P = T

A	B	C	D
b	10	30	x
b	10	15	y
c	20	30	x

Table 2.5.10

where B = C in P = U

A	B	C	D
b	10	10	z

Table 2.5.11

where B > C in P = V(A B C D)

A	B	C	D
a	50	30	x
a	50	15	y
a	50	10	z
c	20	15	y
c	20	10	z

Table 2.5.12

2.6 DOMAIN ALGEBRA

The domain algebra operates on attributes (domains) to give new attributes as results. Operations are classified as horizontal or vertical. Horizontal operators are used to create a new attribute as a combination of values in existing attributes of a tuple. Vertical operators may be thought of as aggregation operations on attributes e.g. totals, subtotals and averages of all the values in a specified attribute may be obtained and used to create a new attribute. Result attributes of the domain algebra are "virtual" i.e they are defined by expressions of the domain algebra but their values are not stored explicitly. When the result values are required in a relation it is necessary to use a command of the relational algebra to create a relation containing the desired attribute. For example consider the domain algebra statement

let TOTPRICE be QTY*UPRICE

which creates a new (virtual) attribute TOTPRICE representing the total price associated with each order in DELIVERY (table 2.1.1). Then the relational algebra command

CHARGES <- DELNO, PRODUCT, TOTPRICE in DELIVERY

yields the result relation CHARGES. Note that it is the projection that actualizes the new attribute TOTPRICE:

CHARGES

DELNO	PRODUCT	TOTPRICE
4	Eggs	24.00
6	Apples	24.80
1	Eggs	31.20
3	Pears	21.00
5	Pears	15.50
4	Apples	20.00
6	Pears	8.50
1	Apples	9.20
2	Pears	17.50

table 2.6.1

While the horizontal operations of the domain algebra create a new attribute by combining values in tuples, one tuple at a time, the vertical operations combine values from more than one tuple. Although ALDAT incorporates a wide variety of vertical operations we restrict our attention to the straightforward cases which are appropriate in the present study.

The "reduction" operation produces a single result from the values from all tuples of a single attribute in the relation. Summation is an example.

let TOTAL be red + of TOTPRICE

will produce the new attribute TOTAL which is the sum of the prices (TOTPRICE) for all orders in CHARGES. In this notation red indicates a reduction operation and + specifies the actual arithmetic operator to be employed. Any operator which is both commutative and associative may be used with "red". The statement

AMOUNT <- TOTAL in CHARGES

creates the relation

```
AMOUNT (TOTAL )
        171.70.
```

Notice that although the result consists only of the single value 171.70 nevertheless it is actualized as a relation. This is consistent with the requirement that the relational algebra operates on relations to produce relations. To find the average total price per order we can write

let AVPRICE be (red + of TOTPRICE)/(red + of 1)

where "red + of TOTPRICE" yields TOTAL and "red + of 1" gives the number of tuples (i.e. adds 1 for each tuple in the relation). Then

AVERAGE <- AVPRICE in CHARGES

yields

```
AVERAGE (AVPRICE)
          19.08    .
```

Equivalence reduction is similar to simple reduction but it provides a different result for different sets of tuples in the relation. Each set is identified by all tuples having the same value for a specified attribute. An important example is subtotalling. For example

```
let AMT be equiv + of QTY by PRODUCT

AMOUNT <- PRODUCT,AMT in DELIVERY
```

yields

AMOUNT(PRODUCT	AMT)
Eggs	46
Apples	135
Pears	125

Table 2.6.2

Clearly the keyword 'equiv' denotes equivalence reduction while the attribute specified after 'by' is used to group the tuples into equivalence classes. As in the case of simple reduction the virtual attribute is actualized by using a command of the relational algebra. ALDAT incorporates a second type of vertical domain algebra called 'functional mapping'. Since this feature is not commonly found in commercial applications it is not necessary to go into details here other than to observe that this type of domain algebra is useful for scientific applications involving statistical calculations. A simple example of functional mapping is the creation of an attribute containing cumulative frequencies.

This completes our discussion of ALDAT and establishes the frame of reference for investigating actual computer systems. We next consider MRDSA.

3. MRDSA

3.1 INTRODUCTION

MRDSA is a relational database management system which is implemented on the Apple II microcomputer using Apple (UCSD) Pascal. Relational operators are implemented as Pascal procedures which can be called from Pascal programs to execute algorithms for the relational commands join, project etc. Although MRDSA is intended for use by programmers, the MRDSA procedures themselves are masked from the user so that it is not necessary to be familiar with the programming details of the system subroutines in order to use MRDSA effectively.

It is not our intention here to present a complete description of the actual details of program structure and syntax, which can be found in the MRDSA programmer's manual (Chiu, 1982). The emphasis is rather on the nature and versatility of MRDSA procedures, so that only those syntactical elements which are necessary for an understanding of the present discussion will be introduced.

3.2 UNARY OPERATORS

In order to project the relation ROUTE (table 2.2.1) from DELIVERY (table 2.1.1) we would write the following short program:

```
PROCEDURE EXAMPLE;
VAR DOMLIST: DLIST;
BEGIN (*EXAMPLE*)
  DOMLIST(0) := 'DELNO';
  DOMLIST(1) := 'SUPPLIER';
  DOMLIST(2) := 'DEST';
  PROJECT('DELIVERY',DOMLIST,3,'ROUTE');
END (*EXAMPLE*)
```

In general the syntax for the procedure project is

project ('R',DOMLIST,N,'S')

where R is the source relation, DOMLIST is an array of N names specifying the attributes of R to be projected, N is the number of attributes to be projected, and S is the result relation. The names of the attributes in DOMLIST are specified via assignment statements in the MRDSA procedure. Implementation of projection in MRDSA yields a result relation which is sorted lexicographically.

In discussing other examples of MRDSA procedures we omit program details and specify only the syntax of the actual procedure call i.e. the relational operator. The reader will readily understand the underlying structure.

The MRDSA procedure QTEXPR provides a quantified query facility on single relations; it is based on the QT-selector which is a combination of projection and selection including quantifiers (cf ALDAT). The QT-selection syntax is

```
QTEXPR ('R',VALCNT,QSYMB,N,FCN,QTPRED,QTCOUNT,'S')
```

where R and S refer to the operand and result relations, respectively. The expressions VALCNT, QTPRED and QTCOUNT are an essential part of the syntax, but are irrelevant in the case of QT-selection in the absence of quantifiers i.e. they are necessary to specify the number of quantifiers, the nature of the quantifiers, etc. but can be ignored when simple selection and projection are required. QSYMB is an array which is used to specify the QT-expression to be evaluated; in the case of simple tuple selection each value of QSYMB will be S (for selection). N stands for the number of symbols in QSYMB. FCN is an integer and specifies which function to use in procedure QPRED, the procedure which actually defines the query or queries to be answered. Thus we see that QTEXPR always requires an associated QPRED command. The syntax for QPRED is

```
QPRED (FCN,I,Q,ATTRI)
```

where FCN indicates which QPRED function is to be used, I indicates which predicate within a QPRED function is to be used; I = 0 for the predicate associated with the first attribute (or attribute group) etc. Q is the quantity used in Q-PREDICATE and ATTRI is an array of attribute values of the current tuple. As an example consider relation P(A,B,C,D) in table 2.5.9. Suppose we want to answer the query

A,B where B < C in P.

Then the appropriate segments of the MRDSA program would be

```
      .
      .
      .
QSYMB(0) := S;
QSYMB(1) := S;
QTEXPR('P',VALCNT,QSYMB,2,0,QTPRED,QTCOUNT,'W');
      .
      .
FUNCTION QPRED(FCN,I,Q,ATTRI);
BEGIN (*QPRED*)
  CASE FCN OF
     0: CASE I OF
        0: QPRED := ATTRI(1) < ATTRI(2);
        END;
  END;
END;
```

The result of this query is relation

```
W(A   B )
  b   10
  c   20        .
```

For a more complete discussion of QTEXPR and QPRED see the MRDSA user's manual.

3.3 RELATIONAL EDITOR

In MRDSA the relational editor is an interactive interface. The procedure EDIT, like other relational operators, is implemented as a Pascal procedure. Unlike the other operators EDIT provides a high level interactive interface for the end-user. The editor provides the user with four operations: access, delete, insert and update. The user supplies a search key (one or more attributes of a given relation) and MRDSA presents one tuple at a time for editing. The editor is invoked with the command

```
EDIT ('R',DOMLIST,DOMLEN,KEYNO,N,'S',1,1)
```

where R and S are the input and output relations, respectively, DOMLIST is an array whose N attributes appear in an order which specifies the sort order and is used as a search key, DOMLEN is an array of N lengths corresponding to attributes of DOMLIST, KEYNO is the number of attributes in the search key. Note that if R = S then no relation is created but R will be changed by the edit session. In this context a search key consists of one or more attributes of a given relation and it is used to identify one or more tuples ot the input relation. Editor commands are insert, delete, update, find and scan. Details on

these and other aspects of the editor can be found in the Relational Editor user's manual.

3.4 GLOBAL UPDATE

This type of operation is not implemented in MRDSA.

3.5 BINARY OPERATIONS

The procedure merjoin in MRDSA implements the family of joins which we refer to as μ - joins in our discussion of ALDAT. Thus merjoin is a generalization of both the natural join and the Cartesian product, as well as the set operations. Merjoin has six modes: I,L,R,U,D and S corresponding to the intersection (natural), left, right, union, difference and symmetric difference joins respectively. Furthermore if N (the number of attributes on which the join is to be performed) is zero, the Cartesian product is assumed for modes I,U,L and R, while an empty relation is returned for modes D and S. The usual set theoretic operations intersection, union, difference and symmetric difference are implemented by merjoin, in the appropriate mode, where DOMLIST specifies the complete set of attributes for both input relations. In this case both relations must be defined on exactly the same set of attributes. The syntax of merjoin is

```
MERJOIN ('R','S',DOMLIST1,DOMLIST2,N,'T',MODE).
```

For example the program segment

```
DOMLIST1(0) := 'DELNO';
DOMLIST2(0) := 'DELNO';
MERJOIN ('ROUTE','GOODS',DOMLIST,DOMLIST,1,'DELIVERY',I)
```

implements the natural join of ROUTE and GOODS to reconstruct the relation DELIVERY (tables 2.1.1, 2.2.1, 2.2.2). By specifying the different modes of merjoin we can implement all of the cases discussed in ALDAT (cf table 2.5.5).

The σ - joins described in chapter 2 are also available in MRDSA. The procedure sigjoin has 12 modes corresponding to different set comparisons (subset, superset, empty intersection, etc.). Since we have not seen any attempt to implement division, or other modes of sigjoin, in the commercial systems which we consider, it is not necessary to go into details of this command. Nevertheless we mention briefly two useful applications

of sigjoin.

The general syntax for sigjoin is

```
SIGJOIN ('R','S',DOMLIST1,DOMLIST2,N,'T',MODE).
```

We observe that mode CIO (complement of empty intersection) implements the natural composition. The following program segment shows how to obtain the natural composition of the relation AGE and INCOME (table 2.5.3):

```
DOMLIST(0) := 'NAME'
SIGJOIN ('AGE','INCOME',DOMLIST,DOMLIST,1,'YRSAL',CIO);
```

The resulting relation is

YRSAL (YEARS SALARY)

YEARS	SALARY
19	14
41	28
24	20
36	40
21	25
32	18

Table 3.5.1

Note that the same result could be obtained using the two commands:

```
MERJOIN ('AGE','INCOME',DOMLIST,DOMLIST,1,'AGEINC',I)
PROJECT ('AGEINC','DOMLIST',2,'YRSAL')
```

where DOMLIST in MERJOIN is the same as in SIGJOIN, while DOMLIST in PROJECT specifies the (non join) attributes YEARS and SALARY.

Our second example of sigjoin refers to the division example of table 2.5.7. The MRDSA command

```
DOMLIST(0) := 'B';
SIGJOIN ('R','S',DOMLIST,DOMLIST,1,'Q',GE);
```

will yield the result relation Q = R(A,B) ÷ S(B) i.e.

Q(A)

A
y
t

.

Note that mode GE (superset) of sigjoin implements relational

division.

3.6 DOMAIN ALGEBRA

At the present time the domain algebra is not available in MRDSA; however a full set of both horizontal and vertical operations of the domain algebra, as described in ALDAT, has been implemented on the IBM PC version of MRDSA (Van Rossum, 1983).

3.7 CONCLUSION

MRDSA provides the programmer with a powerful and versatile tool for creating, editing, and manipulating a relational database. MRDSA has been successful in implementing many of the relational operations which are discussed in the theoretical literature on relational databases, and in some cases has added useful extensions of the relational algebra as defined by Codd (e.g. quantified queries).

ALDAT	:	MRDSA	:	SYNTAX AND COMMENTS
QT-SELECTORS				
PROJECT	:	=	:	project('R',DOMLIST,N,'S')
SELECT ON GENERAL TUPLE CONDITION	:	=	:	QTEXPR('R',VALCNT,QSYMB,N,0,QTPRED, QTCOUNT,'S')
QUANTIFIERS	:	=	:	General QTEXPR syntax cf user manua
UPDATES	:	NO	:	Tuple updates provided by EDIT
μ-JOIN				
$\cap$	:	= mode I	:	merjoin('R','S',DOMLIST,DOMLIST, N,'T',MODE)
$\cup$	:	= mode U	:	
+	:	= mode S	:	
-	:	= mode D	:	
LEFT	:	= mode L	:	
σ-JOIN				
$\subseteq$ $\not\subseteq$	:	mode LE CLE	:	sigjoin('R','S',DOMLIST,DOMLIST, N,'T',MODE)
$\supseteq$ $\not\supseteq$	:	GE CGE	:	
$\cap$ $\not\cap$	:	IO CIO	:	
DOMAIN ALGEBRA				
HORIZONTAL REDUCTION	:		:	The domain algebra has been implemented on the IBM PC version of MRDSA
EQUIVALENCE REDUCTION	:		:	
EDIT	:	<	:	Edit('R',DOMLIST,DOMLEN,KEYNO, N,'S',1,1)
	:		:	The MRDSA editor can find only tuples which match values given for a pre-defined search key. There is no integrity checking.

4. LOGIX

4.1 INTRODUCTION

Logix is a relational DBMS for UNIX and UNIX - like operating systems. Since Logix commands are shell-level programs in UNIX, a user may freely alternate between Logix and UNIX commands. Before we proceed to the relational algebra in Logix we introduce some important definitions and terminology. A database is defined to be a set of relations residing in one UNIX directory; a relation in Logix consists of attributes which are either "key" or "residual" attributes: a key is a set of attributes whose values uniquely identify a tuple (relational key). Whenever an order for the tuples in a relation is not specified, the order of the key attributes is assumed. In creating a relation at least one attribute must be designated as the key, though the user may specify any contiguous set of attributes as the key. Details may be found on page 3 of the Logix tutorial. Most Logix commands do not change the input relations; the result is usually a new relation which is given the default name RESULT unless a name is explicitly specified in the command.

4.2 UNARY OPERATIONS

The projection command in Logix has the syntax

$project R by A,B,....,K .

The attributes listed in the "by" clause are the key attributes of the result relation; these need not be the same as the key of the input relation. However, because key attributes must contain unique values for every tuple, only one tuple from the input relation with a particular key value will be placed in the RESULT relation. In the Logix implementation duplicate tuples are automatically eliminated and the remaining tuples are sorted on the (new) key order. Additional (non-key) attributes of the input relation may be included in the project command as residual attributes, by listing them in the command line after a colon. For example in the DELIVERY relation (table 2.1.1) we could write

$project DELIVERY by SUPPLIER, DEST: PRODUCT, QTY

and the RESULT relation would contain five tuples, one for each unique SUPPLIER - DEST pair, with values of PRODUCT and QTY arbitrarily selected from the set of tuples for each SUPPLIER - DEST combination.

In conjunction with, and as an extension of, projection, Logix supports the statistical operations of finding maxima, minima, sums, averages and counts of groups of tuples. The commands min and max are similar to project but rather than selecting an arbitrary tuple they will select the tuple with minimum or maximum values, respectively, in the residual attributes. The commands sum, avg and count perform some of the functions of the domain algebra which we discuss in section 4.6.

The command select creates a new relation which contains some of the tuples of the source relation. The condition for tuple selection is entered interactively in response to prompts (->) and terminated by a null line. Conditions for inclusion may be formed using the usual order comparisons, negation and arithmetic operations. For attributes with text types the pattern matching operators "matches", "begins with", "ends with" and "contains" are available. String concatenation and Boolean combinations are also permitted. In order to illustrate select, and other Logix commands, we introduce the relations FIRST(NAME,YEAR) and LAST(NAME,YEAR), representing respectively the year in which a customer first placed an order and the year of the last order placed. Note that there are some customers for whom either the first or last date is not given.

FIRST(NAME	YEAR)
A	1976
B	1968
C	1982
D	1970
E	1974
F	1981
G	1974

LAST(NAME	YEAR)
A	1982
B	1980
C	1982
D	1970
E	1977
G	1978
P	1976
R	1965

Table 4.2.1

```
select NEWCUST from FIRST
-> YEAR > 1979
->
```

yields the relation

NEWCUST(NAME	YEAR)
F	1981
C	1982

It is noteworthy that the Logix implementation of select uses an interactive approach instead of the more usual conditional clause (e.g. "if" or "where") which is employed by most systems.

Logix supports the use of "command modifiers" to add flexibility to the relational algebra. Most Logix commands can be modified using either "by" or "if" clauses. "By" clauses can be used to specify an order for the tuples other than the default order by key columns. We illustrate the use of command modifiers by giving examples using the list command which is a non-relational command used strictly for output. For example

$list ROUTE by SUPPLIER

will result in the tuples of relation ROUTE (table 2.2.1) being listed with SUPPLIER as the first attribute and supplier names being sorted in alphabetical order, thus:

SUPPLIER	DELNO	DEST
Adams	2	White
Adams	3	Daly
Cowan	5	Stanley
Fingal	1	O'Neill
Green	3	Daly
Green	6	Daly

Any number of attributes may be specified after "by". The other type of command modifier, the "if" clause, allows the user to select only those tuples which satisfy a given condition to be included in the result of a command. This produces the effect of a select. For example

$list ROUTE if "SUPPLIER = 'Green'"

will yield the listing

DELNO	SUPPLIER	DEST
3	Green	Daly
6	Green	Daly

.

The "if" and "by" clauses can be combined in a single command, as in

$list ROUTE by SUPPLIER if "DELNO > 3"

which causes the following listing

SUPPLIER	DELNO	DEST
Adams	4	Brown
Cowan	5	Stanley
Green	6	Daly.

Another feature of Logix which increases the versatility of the relational operators is the concept of "segments". A segment of a relation is defined to be any contiguous sequence of attributes of that relation which has been designated as a

segment. Each segment has a name and most operations on attributes can also be applied to segments. We note that in MRDSA the concept of a segment is not explicitly defined but one can achieve a similar effect by appropriately choosing the attributes to be included in the DOMLIST arrays of the MRDSA commands.

4.3 RELATIONAL EDITOR

Logix provides a relational editor red for interactively viewing, updating, inserting and deleting tuples in a relation. red is a line editor, similar to the UNIX line editor ed. Relations may be edited tuple-by-tuple or global commands may be used to change several tuples. The editor is invoked with the command red R. A full-screen editor fred enables users to define forms and interactively update relations, providing the same features as red.

4.4 GLOBAL UPDATE

Several commands are available for adding tuples to an existing relation or changing tuples according to specified criteria. The command addto will add tuples from a text file to a relation, while append will add tuples from one relation to another using a statement of the form append R to S. Append may be used even if the two relations do not have identical structures - "Logix will look for columns with identical names and map the items (tuples) of one relation onto the other". However "attributes with the same name must have the same domain". The update command is used to update one relation based on the contents of another; this type of binary update may be used to "post" master files from transaction files. As already mentioned, interactive updating is possible using the line or full-screen editors.

4.5 BINARY OPERATIONS

Logix has a join command with syntax

$join R,S to T

This command corresponds to the left join operation in ALDAT (MERJOIN, mode L in MRDSA) though it is more limited than the corresponding MRDSA command because the user cannot specify explicitly in the join syntax the attributes on which the join is to be performed; thus the join in Logix is limited to joins

on keys i.e. it is assumed that the join will be performed on matching key values in the two input relations. (A communication from the manufacturers of Logix asserts that "the general join facility" is being implemented. We have received no further details so we cannot expand further, thus we may be underestimating the join capability of Logix by the time this paper appears.)

The commands compare, conflicts, diffa, diffb and merge are the "comparison operators" in Logix. These operations are similar to set operations in that the two operand relations are defined on the same set of attributes, but a new feature is introduced in the form of a "tag" attribute which is appended to the result relation and which keeps track of which input relation contributes each tuple of the result. Thus the comparison operators combine set theoretic and domain algebra features. We illustrate these operators using the relations FIRST and LAST (table 4.2.1).

The command

$merge FIRST LAST

yields the relation

RESULT(NAME	TAG	YEAR)
A	1	1976
A	2	1982
B	1	1968
B	2	1980
C	c	1982
D	c	1970
E	1	1974
E	2	1977
F	a	1981
G	1	1974
G	2	1978
P	b	1976
R	b	1965

Table 4.5.2

The meanings of the five possible tag values 1,2,a,b,c are given in the following table; the expressions "attrib1" and "attrib2" refer to the complete set of attributes in the first and second input relations respectively.

TAG VALUE	MEANING	ALDAT EQUIVALENT
1	:tuple comes from first :relation - key value in :both relations	:attrib1 in (rel1 key1 ijoin key2 rel2
2	:tuple comes from second :relation - key value in :both relations	:attrib2 in (rel2 key2 ijoin key1 rel1
a	:tuple comes from first :relation - key value not:in second relation	:attrib1 in (rel1 key1 djoin key2 rel2
b	:tuple comes from second :relation - key value not:in first relation	:attrib2 in (rel2 key2 djoin key1 rel1
c	:tuple is common to both :relations	:rel1 ijoin rel2

Table 4.5.3

The Aldat equivalent in table 4.5.3 ignores the additional tag attribute. If it seemed desirable to include such an attribute it could be defined via ALDAT's domain algebra. The following table indicates which tag values are included in the RESULT relations created by the comparison operators.

COMMAND	TAGS
merge	1,2,a,b,c
compare	1,2,a,b
diffa	1
diffb	2
conflicts	1,2

table 4.5.4

The conflicts operator is used when residual attributes are included to select tuples with common keys but different residual columns. Note that the tag attribute actually appears in the RESULT relations for the comparison operators.

The set operations in Logix are union and common. The union command yields a relation which includes only those tuples which would be tagged a,b or c in a merge command. However no tag attribute is included in the result of union. Tuples whose keys are common to both input relations but whose residual attributes differ are omitted. For example

$union FIRST LAST

yields the relation

RESULT(NAME	YEAR)
C	1982
D	1970
F	1981
P	1976
R	1965

Table 4.5.5

Note that union in Logix is not what we commonly refer to as union in set theory. However the command common yields the set theoretic intersection (i.e. tuples tagged with 'c' in merge but without the tag attribute). Thus

$common FIRST LAST

yields

RESULT(NAME	YEAR)
C	1982
D	1970.

4.6 DOMAIN ALGEBRA

In section 4.2 we mentioned the statistical operations of Logix. We have already discussed the functions of the commands min and max. The remaining statistical operators are count, sum, and avg. The command count is a form of projection which not only projects on the specified attributes but also appends an extra attribute, headed TALLY, which specifies the number of occurrences of each unique combination of attribute values in the key attributes named in the count command. For example, referring to table 2.1.1

$count DELIVERY by SUPPLIER, DEST

yields

RESULT(SUPPLIER	DEST	TALLY)
Adams	Brown	2
Adams	White	1
Cowan	Stanley	1
Fingal	O'Neill	2
Green	Daly	3

Table 4.6.1

The command sum will calculate the sum of the values for a specified residual attribute and will append an extra attribute, called COUNT, which records the number of tuples contributing to the sum. Note that COUNT is an attribute in this situation and is actually the same as the attribute TALLY defined above. The command

$sum DELIVERY by SUPPLIER, DEST: QTY

creates

RESULT(SUPPLIER, DEST, QTY, COUNT)

SUPPLIER	DEST	QTY	COUNT
Adams	Brown	70	2
Adams	White	35	1
Cowan	Stanley	31	1
Fingal	O'Neill	49	2
Green	Daly	121	3

Table 4.6.2

The attribute QTY now specifies the total number of objects being shipped from each supplier to each destination (we overlook the fact that in this example we are adding eggs to apples !). We see that sum implements subtotalling, a function of equivalence reduction in ALDAT. The command avg is similar to sum except that it calculates the average of the values of a given residual column, while again appending the COUNT attribute. Thus the statistical operations min, max, count, sum and avg are seen to perform some of the functions of the domain algebra.

ALDAT	: LOGIX	: SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	: >	:$project R by A,B,..(residual columns)
SELECT ON GENERAL: TUPLE CONDITION	: = :	:select R from S :-> condition (entered interactively)
QUANTIFIERS	: NO	:
UPDATES	: YES :	:addto, append, $update R from S :line and full screen editors
μ - JOIN		
$\cap$	: <	:$common R S (set intersection only)
$\cup$	: <	:$merge R S (set union plus tags)
+	: <	: compare, diffa — restricted join plus tag - limited to relations with same structure.
-	:	: diffb, conflicts
LEFT	: <	:$join R,S to T (restricted to join on keys)
σ - JOIN		
$\subseteq$ $\not\subseteq$	: NO NO	:
$\supseteq$ $\not\supseteq$	: NO NO	:
$\cap$ $\not\cap$	: NO <	:
DOMAIN ALGEBRA		
HORIZONTAL REDUCTION	: <	: {min, max, count, sum} R by A,B,....:P,Q,..
EQUIVALENCE REDUCTION	: <	: {avg}
EDIT	:	: $red R , $fred R (line and full-screen)

NOTES: (1) implicit select is possible via 'if' clauses (cf command modifiers).
(2) LOGIX manufacturers have communicated that the 'general join facility' is being implemented.

5 CONDOR

5.1 INTRODUCTION

CONDOR Series 20/rDBMS is a relational DBMS for Z80 microcomputers. Some preliminary remarks are in order before discussing the relational algebra. First we consider the way CONDOR handles sorting; the following comment on "usage strategy" is taken from the user's manual (CONDOR,1981 Page 35). "Sorting is not required nor done automatically by CONDOR Series 20/rDBMS commands . However if files are sorted response time will improve by as much as 90%". The syntax for the explicit sort command is

sort R by A,B,.....[D] ,

where up to 32 attributes may be specified with A as the major sort field, followed by B and so on. The optional switch 'D' when specified will cause the relation to be sorted in descending order. In implementing the sort a temporary work file is created on the current disk drive and the relation is rewritten in the sorted order; the work file is erased at the end of the sort.

Another important aspect of CONDOR is the "matching fields" feature. The binary relational operators require that the attributes on which the operation is to be performed be explicitly specified as matching fields; these fields (attributes) must be defined identically (i.e. have the same name, size and type) in the two operand relations. Recall that in Logix the join command was limited to predefined key attributes; in CONDOR the philosophy is quite different - to quote again from the user's manual "one of the primary features of a relational database is that there is no requirement that specific key fields be defined". Thus in the join, and other, commands attributes must be specified explicitly as matching fields.

5.2 UNARY OPERATIONS

The project command

project R by A,B,.....

is equivalent to the project operator in MRDSA and the simple

project in Logix. The command

select R where clause

is used to create a RESULT relation consisting of those tuples which satisfy the where clause. The selection condition permits numeric or character comparison of attribute values with constants or other attribute values. Boolean combinations are permitted though both 'and' and 'or' may not be used together in the same command. The usual comparison operators, as well as several synonyms , are available. For example <>, NE, IS NOT, and ARE NOT may be used to express 'not equal'.

5.3 RELATIONAL EDITOR

The command update with syntax

update R where clause

selects the set of tuples which are presented interactively for editing by the user. We emphasize that update is an interactive command which permits the user to change one tuple at a time. The where clause is the same as in the select command.

5.4 GLOBAL UPDATE

A general feature of CONDOR is that it employs a business oriented terminology. For example

post R S matching A,B,.... {{opr C,D...} ...}

where opr can be any of the following operators:
REPLACE (REP) , ADD (SUM) , SUBTRACT (SUB).
The ADD and SUBTRACT operators apply only to attribute values declared as numeric (N) or dollar ($) data types. The post command is used to update attribute values in one relation with those from another. In effect post is a combination comparison - updating operation which matches attribute values from the two input relations and, when a match occurs, the tuple from R is updated from the matching tuple in S, as specified by the operators in the command: thus attribute values may be added to, subtracted from, or replaced by values of the second relation.The result relation consists of updated instances of all tuples R that matched tuples of S. A typical application of post is to update a master file from a transaction file. The following illustration is taken from the user's manual:

post TASK TIME matching TASKNO and add HOURS.

The result of this command is as follows - for each matching TASKNO in the input relations TASK and TIME, the hours specified in the TIME relation are added to the hours already in the TASK relation. If no attributes or operators (add, subtract, replace) are specified in the syntax, the command will default to a complete replacement operation. The RESULT relation created by post may be used to maintain an audit trail and for verification purposes: for further details and examples the reader is referred to the user's manual (page 36).

5.5 BINARY OPERATIONS

The join command in CONDOR has syntax

join R S matching A,B,C,.... [D]

where up to 32 attributes may be specified. The join implemented by this command is the natural join unless the optional switch 'D' is specified; in this case if no match occurs between any one tuple in R with the tuples of S, a tuple will be forced to appear in the RESULT relation with default values of S joined to the unmatched tuple from R. This is the left join (MERJOIN - mode L in MRDSA).

The comparison operator compare may be used to match tuples of one relation with tuples of another by specified attribute(s) and create a RESULT relation consisting of tuples from the first input relation which match (or do not match) the tuples of the second relation. The syntax is

compare R S { matching / not matching } A,B,....

We illustrate this command by referring to table 2.5.3 where relations AGE and INCOME are defined. The command

compare AGE INCOME matching NAME

creates the relation

RESULT(NAME	YEARS)
B	41
G	32
A	19
C	24
D	36
E	21

Table 5.5.1

In ALDAT terminology this would be

RESULT <- NAME,YEARS in (AGE ijoin INCOME) .

Thus, in general, CONDOR's compare matching procedure is equivalent to a natural join followed by a projection on the attributes of the first relation in the join. Likewise the command

compare AGE INCOME not matching NAME

yields

RESULT(NAME	YEARS)
F	27

Again, the ALDAT equivalent would be

RESULT <- NAME,YEARS in (AGE djoin INCOME)

i.e a (left) difference join followed by projection on attributes of the first relation.

The set theoretic union of two relations which have identical structures (attribute order and type) may be implemented using the command

combine R S

The result relation is the union of R and S, both of which remain unchanged. Set intersection is obtained by using compare on all attributes of the two relations, again assuming identical structures.

5.6 DOMAIN ALGEBRA

CONDOR provides some of the effects of the domain algebra through the stax command, with syntax

stax R by A,B,..... option

where the available options are A,C,M,T and P which result in averages, count, maxima and minima, totals and printer output, respectively. Note that the option M results in both minimum and maximum values being displayed i.e. one cannot have one without the other. If no option is specified all statistics are displayed on the screen. The stax command will display the statistics specified in 'option' for the attributes included in the attribute list. Stax implements some of the simple reduction operations in ALDAT.

Features of the horizontal domain algebra are implemented through the compute command with syntax

compute R st C = {{A opr B} ....}

where C is the name of the new attribute and opr may be + , - , * or / . The user's manual states that "using this command properly requires a careful understanding of the syntax; otherwise unexpected results may be obtained" and further "caution must be exercised in using this command because of its global effects". An example offered in the manual is

compute ORDERS st PROFIT = PRICE - COST.

This command will cause a new attribute, PROFIT, to be added to the ORDERS relation. In ALDAT this result would be achieved by the statement

let PROFIT be PRICE - COST

followed by a projection on all the attributes of ORDERS plus the new attribute PROFIT.

This comletes our survey of the relational features of the CONDOR Series 20 relational database management system.

ALDAT	CONDOR	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	=	project R by A,B...
SELECT ON GENERAL TUPLE CONDITION	<	select R where clause (AND and OR cannot be used in the same clause)
QUANTIFIERS	NO	
UPDATES	YES	post R S matching A,B,... {{oprB,C..} ..}
μ-JOIN		
$\cap$	=	join R S matching A,B,...(natural join) compare R S matching A,B,...(natural join plus projection on R - set intersection if all attributes are specified)
$\cup$	<	combine R S (set union only)
+	NO	
-	<	compare R S not matching A,B,... join R S matching A,B,... D
LEFT		
σ-JOIN		
$\subseteq$ $\nsubseteq$	NO NO	
$\supseteq$ $\not\supseteq$	NO NO	
α $\not\alpha$	NO =	
DOMAIN ALGEBRA		
HORIZONTAL	<	compute R st C= {{A opr B}....}
REDUCTION	<	stax R by A,B,... option (max,min,count, avg,total)
EQUIVALENCE REDUCTION	NO	
EDIT	<	update R where clause

6 dBASE II

6.1 INTRODUCTION

This DBMS runs on a variety of microcomputers and is available for the more widely used operating systems, such as CP/M, MS-DOS etc. The system is relational in that data are stored and managed according to the relational model. Several relational commands are implemented directly while others may be effected in a less obvious fashion. For example, at first sight it appears as if the project command is completely overlooked, yet on further investigation it is evident that one can in fact realize projection by use of the copy command which permits the user to copy any subset of the attributes of a given relation.

The following information is necessary to understand the way dBASE II works. The system maintains two work areas named PRIMARY and SECONDARY; the relation residing in either work area at any time is called the USE relation for that area. In this context we note that the command select has an entirely different meaning in dBASE II from the usual selection operator of the relational algebra. Here select is used to identify either the primary or secondary work area as supplying the operand relations in dBASE II commands. The select command has the form

select { primary / secondary } .

6.2 UNARY OPERATIONS

Relational algebra selection is implemented in dBASE II via the locate command

locate scope for clause

where scope is an optional command modifier which specifies the set of tuples to be included in the command; scope may be all, to search the entire relation, next n, to search only the next n tuples (including the current one), or record n which means examine only the n th record (or tuple). If scope is omitted it defaults to all. The for clause may include numeric comparison,

string and substring matching and Boolean operators. Locate corresponds to selection on a general tuple condition as described in ALDAT.

dBASE II provides another approach to tuple selection via the concept of 'keys' and indexed files. We note that key, in this context, refers to a sort key and not a key in the relational sense of unique tuple identifier. In order to use the find command it is necessary to create an indexed relation using the command

index on A,B,..... to IR

where IR indicates an indexed relation which is said to have the specified attribute list as its key. Notice that the relation to be indexed is not explicitly named since the command automatically works on the USE relation in the current work area. Several indexes may be created for the same relation; the index command creates a new indexed relation which is stored as a B-tree and which contains pointers to the tuples in the USE relation. The USE relation itself is unchanged. Searching and sorting relations is more efficient when the indexed relation technique is used. The command

find char string

causes the computer to find the first tuple in an indexed relation (in USE) whose key is the same as 'char string'. The user's manual states that "a typical find time is two seconds on a floppy diskette system". Find may only be used with indexed relations. In many cases the index command need only be used once since other commands will automatically adjust the index when new tuples are added to the relation.

The command

sort on A to R { [ascending] / [descending] }

creates a new relation sorted on the specified attribute. The source relation is unaltered and remains in USE. The manual states "to sort on several keys (fields), start with the least important key, then use a series of sorts leading up to the major key". If neither ascending nor descending is specified, ascending is assumed. The manual further states "the index command is compared with the sort command in this way: index, when done, performs nearly all of sort's duties. Also, index generally allows greater freedom and greater speed than sort".

dBASE II does not have an explicit projection operator. However there is a copy command which may be used to duplicate an entire relation or part of a relation. Thus by setting up an empty relation with the appropriate structure (attribute names, order and type) specifying only a subset of the attributes in the operand relation we can effectively produce a projection. The manual warns that "when you copy to an existing filename, the file is written over and the old data is destroyed". The command copy to temp creates a new relation called temp.DBF. The general syntax for copy is

copy to R [structure][fields A,B,...].

For example to project relation ROUTE (table 2.2.1,page 6) from DELIVERY (table 2.1.1) we would write:

```
use DELIVERY
copy to ROUTE structure fields DELNO,SUPPLIER,DEST
```

As we shall see in section 6.5 the join command in dBASE II also permits the projection of the result of a join on specified attributes.

The copy command causes the creation of a new relation but does not automatically display it. The display command has syntax

display [scope] [off][for clause]

Scope has already been explained with locate. Normally record numbers are included with the display. To suppress the printing of record numbers the option 'off' may be specified. Displaying selected tuples is possible by including the optional 'for' clause.

6.3 RELATIONAL EDITOR

Three commands are available for interactive editing. The edit command:

```
use R
edit n
```

may be employed to edit the nth tuple in relation R. Note that the use command 'opens' the relation while 'edit n' causes the tuple to be displayed for full screen editing.

Another command,change, may be used to select tuples for

editing according to specified criteria.

change [scope] A[, B,....] [for clause]

finds the first tuple that meets the condition in the 'for' clause and displays the tuple for editing; when you are finished with that tuple the next one is presented and so on until all tuples have been examined or the 'escape' key is pressed. Full screen editing allows the user to change, add or delete tuples.

A third editing command which is very useful in practice is browse which provides the user with a 'window' into the relation. The view of the relation may be changed from the keyboard by means of scrolling the screen up or down, left or right. Changes may be made to the relation while in the browse mode.

6.4 GLOBAL UPDATE

The command

update from R on key { add / replace } A,B,....

may be used to modify the USE relation with data from another relation. The options in update are add or replace, meaning that we can update attribute values in the USE relation by adding values from the other or by replacing them with values from the other relation. Update is implemented by comparing tuples in the 'from' and USE relations and performing the add or replace function if the keys match. Note that in order to use this command the USE relation must be presorted or indexed on the key, while the 'from' relation must be presorted on the key.

The command

replace [scope] A with exp [,B with exp]* for clause

is used to replace data in attribute(s), where the 'exp' can be "specific new information (including blanks) or it could be an operation such as deducting sales tax from all your bills because you have a resale number". The specified attribute(s) may already exist in the operand relation, in which case the command amounts to a global replacement of attribute values, or may be new attribute(s) in which case the replace operation becomes a command of the horizontal domain algebra (cf section 6.6). The 'for' clause allows the user to replace attribute

values in selected tuples.

6.5 BINARY OPERATIONS

The join command is implemented as a Cartesian product followed by tuple selection (the θ -joins in ALDAT). This implementation may cause problems with execution time and data storage if the condition for tuple selection is too loose. From the description of join syntax usage it is evident that by specifying only a subset of the attributes in the attribute list one can produce the effect of a project on the result relation. The following example, taken from the user's manual, illustrates the sequence of instructions to be used when performing a join:

```
use INVENTORY
select secondary
use ORDERS
join to NEWREL on P.PART:NUMBER = PART:NUMBER;
        field CUSTOMER,ITEM,AMOUNT,COST
```

This program segment creates a new relation (NEWREL) with four fields with the same data types as in the two source relations. Notice that using the 'P.' prefix in P.PART:NUMBER calls a variable from the work area (primary) which is not currently in USE. We observe that the join syntax permits the user to effect an implicit projection in conjunction with the join. We also note that by varying the conditional expression and attribute list in the join command it is possible to construct other types of operation such as natural composition, less-than join etc., though the family of μ -joins could not be derived from this implementation. The general syntax for join is

```
join to R on condition field A,B,....
```

We close this section by noting that the update command discussed in section 6.4 is in fact a binary operation, namely a binary update or 'post' command.

6.6 DOMAIN ALGEBRA

In section 6.4 it was noted that the replace command may be used to implement the horizontal domain algebra function of creating a new attribute and appending it to a relation. For example to add a new attribute TOTPRICE to the DELIVERY relation (table 2.1.1) we would write

use DELIVERY
replace all TOTPRICE with QTY*UPRICE.

Since the general replace syntax supports the simultaneous creation of multiple attributes and also permits the user to specify the scope as well as a 'for' clause, we conclude that dBASE II implements a very general version of the horizontal domain algebra.

Vertical operations of the domain algebra are count, sum and total. The command

count [scope] [for clause] [to memory variable]

will count the number of tuples in the USE relation. If the 'for' clause is included then only the tuples which satisfy the condition are counted. If the 'to' clause is included, the integer count is stored in a memory variable which will be created at this time if it was not defined prior to this command. The command

sum A[,B] *[to memory var[,memory var] *] [scope] [for clause]

adds numeric expressions involving the USE relation according to the scope and 'for' clauses. Up to five attributes may be summed simultaneously. If the 'to' clause is present the sums will be saved in memory variables. The command

total on key to R [A[,B]*] [for clause]

may be used to calculate totals for specified attributes and place the totals in a new relation. The USE relation must be either presorted or indexed on the key.

We conclude that dBASE II incorporates several useful features of both horizontal and vertical domain algebra. Totals and subtotals are accessible via the sum and total commands in conjunction with the optional 'for' clauses. While averages and other statistical functions are not explicitly provided, we note that a general feature of dBASE II is the availability of built-in programming features, specifically the 'if...else...endif' and the 'do while ..... enddo' structures. The 'do case' structure is also available. This programming capability may be used to create "command files" thus enabling the user to define his or her own custom designed commands.

ALDAT	:dbase II:	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	: =	: copy to R [structure][fields A,B,...]
SELECT ON GENERAL TUPLE CONDITION	: <	: locate [scope] for clause
QUANTIFIERS	: NO	:
UPDATES	: YES	: update from R on key { add A,B,... / replace A,B,... }
		: replace [scope] A with exp [,B with exp]*[for clause]
μ-JOIN		
$\cap$	: >	: join to R on cond field A ,B *
$\cup$	: NO	:
+	: NO	:
-	: NO	:
LEFT	: NO	:
σ-JOIN		
$\subseteq$ $\nsubseteq$	: NO NO	:
$\supseteq$ $\not\supseteq$	: NO NO	:
$\cap$ $\not\cap$	: NO =	:
DOMAIN ALGEBRA		
HORIZONTAL	: >	: replace [scope] A with exp [,B with exp]* [for clause] (allows multiple attribut comparison, also scope and for clause)
REDUCTION	: <	: count [scope] [for clause] [to mem var]
		: sum A[,B]*[to mem var[,to mem var]*] [scope] [for clause]
EQUIVALENCE REDUCTION	: <	: total on key to R[A[,B]*] [for clause]
EDIT	: <	: edit n (fetch tuple by number)
		: change [scope] A[,B]*[for clause](fetch tuple using for clause)

7. RQL

7.1 INTRODUCTION

RQL is described as a relational algebraic query language. It is a relational DBMS for the Apple microcomputer and it requires at least 48K RAM, the DOS operating system (version 3.3 or greater), the APPLESOFT language and at least two floppy diskette drives. The user's manual is well written and contains many examples. First time users of a database system should be able to teach themselves how to use RQL with relative ease. The syntax is easy to follow and is clearly intended to permit the user to formulate queries and requests in a 'natural' manner.

It appears that RQL operators such as project and select do not automatically sort the result relations. In order to sort a relation prior to displaying it, the command

sort R by A

may be used to sort the tuples in ascending order on the specified attribute. After execution a new relation R.SRT is created. The name is generated automatically. Since relation names cannot exceed eight characters in length, the input relation name must be no longer than four characters to accommodate the extension .SRT. The rename command can be used to give a shorter name to a relation prior to issuing a sort command. It seems that both the original relation R and the sorted version R.SRT are stored in the database.

7.2 UNARY OPERATIONS

Projection in RQL is thought of as attribute selection. Thus

let R be select A,B,... from S

will project S on the named attributes and create the new relation R. If no attributes are specified the operation simply copies the input relation. Duplicate tuples are eliminated in RQL's implementation of projection.

The selection operation is visualized as a tuple selection from a specified relation with a general tuple selection condition given in the form of a 'where' clause. In RQL syntax two types of 'where' clause are permitted, clausel and clause2. The general form of clausel is as follows:

attribute { = | <> | < | > | <= | >= } { value | "value" } [{ or | and } attribute { = | <> | < | > | <= | >= } { value | "value" }] *

where value, without quotes, specifies a value if the attribute type is numeric whereas "value" implies a character value. We note that the clausel syntax permits comparisons with constants only and does not allow for the comparison of the values of two different attributes. The syntax for selection in RQL is

let R be select from S where clausel.

The user's manual next introduces "projection and selection operation" (Chen and Driscoll,1982) and states that "a new table can be created by selecting columns and rows from an existing table according to qualifications placed on the row values". The syntax is

let R be select A,B,..... from S where clausel .

This combined projection/selection operator corresponds to the T-selector in ALDAT (cf ALDAT, page 7). For example with reference to table 2.1.1 and table 2.2.4 we can create the relation EGGDEL from DELIVERY by the command

```
let EGGDEL be select SUPPLIER DEST QTY from DELIVERY
          where PRODUCT = "Eggs".
```

7.3 RELATIONAL EDITOR

RQL does not support a relational editor permitting interactive editing of tuples one at a time. The only interactive procedure is insert R which, when invoked, causes RQL to prompt the user for the tuple to be input, one attribute value at a time, into relation R. The prompt explicitly displays the name, field width and data type of the attribute to be entered.

7.4 GLOBAL UPDATE

The command

update R with A = { value / "value" } [B = { value / "value" }]* where clausel

will cause all tuples satisfying clausel to be modified with the specified attribute values. For example, referring to the EGGDEL relation in table 2.2.4 the command

update EGGDEL with SUPPLIER = "Evans" QTY = 24
where DEST = "Brown"

results in

EGGDEL(SUPPLIER | DEST | QTY)
Evans | Brown | 24
Fingal | O'Neill | 26

Table 7.4.1

The command

delete R where clausel

allows the user to delete, with a single statement, all the tuples of R which satisfy clausel. For example

delete EGGDEL where QTY > 25

when applied to the relation in table 7.4.1 results in the new relation

EGGDEL(SUPPLIER | DEST | QTY)
Evans | Brown | 24

7.5 BINARY OPERATIONS

The join command has syntax

let R be join S T where clause 2 .

Clause2 is used only for the join operation and has the form

attrib from rel 1 { = | <> | < | > | <= | >= } attrib from rel 2 [{ or | and } attrib from rel 1 { = | <> | < | > | <= | >= } attrib from rel 2]

We note that the natural join is a special case in which clause2 consists of a single equality comparison of attributes from the two input relations. The ability to include Boolean combinations of comparisons between attributes from the operand relations generalizes the concept of natural join as defined in ALDAT. The family of Θ-joins, and generalizations involving multiple comparisons, can be derived from the join command in RQL albeit with an expensive implementation consisting of Cartesian product followed by tuple selection.

The set operations in RQL are union, intersection, relative complement and symmetric difference. Attributes must match in number, length and type in the two input relations. The new relation will have the same attribute names as the first input relation in the command. Syntax for the four set operations is as follows:

SET OPERATION	:	SYNTAX
Union	:	let R be in S or in T
Intersection	:	let R be in S and in T
Relative complement	:	let R be in S minus T
Symmetric difference	:	let R be in S or T but not both

7.6 DOMAIN ALGEBRA

The "aggregate function" in RQL corresponds to vertical domain algebra operations. Both simple and equivalence reduction are represented. The general form of the syntax for four of the five aggregate functions is

{ avg | min | max | sum } R.A [where clausel]

while the fifth operation has the syntax

count R [where clausel] .

The names of the functions are self explanatory. If the optional where clause is omitted simple reduction, involving all tuples of the relation, is assumed. Inclusion of the where clause limits the arithmetic function to tuples satisfying clausel and is therefore an implementation of equivalence reduction operations such as subtotalling. For example, referring to DELIVERY (table 2.1.1), the command

sum DELIVERY.QTY where PRODUCT = "Eggs"

will result in the value 46 being displayed i.e. the total number of eggs being shipped. Likewise

min DELIVERY.QTY where DEST = "Daly"

will display the result 17 which is the minimum value of QTY in tuples for which DEST = "Daly". It is evident that the specified attribute in these commands must be numeric. The command

count DELIVERY where QTY > 25 and PRODUCT <> "Eggs"

will result in the numeric display 5 i.e. the number of tuples for which the QTY exceeds 25 and the PRODUCT is not eggs.

There is no attempt to implement the horizontal domain algebra in RQL.

ALDAT	RQL	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	=	let R be select A,B,... from S
SELECT ON GENERAL TUPLE CONDITION	<	let R be select from S where clause
QUANTIFIERS	NO	
UPDATES	YES	update R with A= { value / "value" } [B= { value / "value" }]* where clause1
μ - JOIN		
	>	let R be join S T where clause2
$\cap$		let R be in S and in T (set intersection)
$\cup$	<	let R be in S or in T (set union)
+	<	let R be in S or T but not both(symm diff)
-	<	let R be in S minus T(relative complement)
LEFT	NO	
σ - JOIN		
$\subseteq$ $\nsubseteq$	NO NO	
$\supseteq$ $\not\supseteq$	NO NO	
$\wr$ $\not\wr$	NO =	
DOMAIN ALGEBRA		
HORIZONTAL	NO	
REDUCTION	<	{ avg / min / max / sum } R.A [where clause1]
EQUIVALENCE REDUCTION	<	count R [where clause1]
EDIT	<	insert R (permits interactive input)

8. RMS

8.1 INTRODUCTION

Produced by the same manufacturer as RQL, RMS is a relational menu system with the same system requirements as RQL. There are significant differences between the two systems, both with respect to the user interface (menu vs syntax) and the available commands.

RMS must be accessed via a password which may be SUPER or REGULAR. Only users with SUPER passwords have the ability to perform operations which alter the content of a database such as destroy or rename a database, update a relation (table) with protection and so forth. In order to create a relation in RMS the DEFINE TABLE menu option is used. The procedure for actually defining the relation i.e. specifying number of attributes, attribute names, lengths and data types is performed interactively with the system prompting for each input. Full details are provided in the clearly written user's manual (Driscoll,1983).

Selecting the SORT TABLE menu option will cause the computer to request the name of the relation to be sorted and the attribute on which to sort the relation. RMS will then rearrange the relation on the requested attribute. Sorts may be fast or slow - the fast sort being possible only if the relation fits entirely in core memory. The user's manual suggests that one should try the fast sort first and only resort to the slow sort if an "out of memory" error occurs.

8.2 UNARY OPERATIONS

Selection is possible with the SELECT ROWS menu option. A new relation with attributes identical to the input relation must first be defined using DEFINE TABLE, and then RMS will fill this relation with only those tuples satisfying a given condition. The condition is entered by first typing in the attribute number in the form Cn, where 'C' stands for column and 'n' is the actual column number; then a general condition can be constructed using the usual combination of comparison, arithmetic operations (including exponentiation) and Boolean operators. Examples of selection conditions are:
C2 <= 5 , C3 > "M" , NOT(C3 = "B") , C4 = C2 AND C4 < C1

Note that attributes may be compared to each other and not merely to constants.

Projection may be implemented by selecting PROJECT COLUMNS from the command menu. It is necessary to use DEFINE TABLE to create the (empty) relation which is to receive the projected attributes. PROJECT COLUMNS is more general than the usual projection operator; for example it may be used to insert attributes of one relation (the operand) into another (the result) - this is accomplished by creating a new relation with the desired attributes of the original relation plus any additional attributes, which may be placed anywhere within the sequence of attribute declarations when DEFINE TABLE is used to set up the recipient (result) relation. PROJECT COLUMNS is then used to insert the old attributes into their respective locations in the new relation. Another application of this command would be to change the declaration of the data type for an attribute, for example increasing the length of a character attribute, by defining a new relation with the desired changes and projecting the old relation into the new one. See page 19 in the RMS user's manual for details.

8.3 RELATIONAL EDITOR

Inserting tuples into a table (relation) is effected via the INSERT ROW menu option. Once a relation is selected RMS allows the user to insert a new tuple attribute by attribute. The user's manual states the following: "Preceding each request for entry is a reminder of the column name and its data type. To represent the proper length, a block is displayed of the declared size in which you must place your data". Tuple updating and deletion require the UPDATE ROWS option which causes the relation to be displayed tuple by tuple. Five responses are available to the user as shown in the following table:

ENTER	: RESULT
Return	: No change - display next tuple
E	: Discontinue ('end') updating
D	: Delete current tuple
R	: Replace current tuple - system prompts for input as in INSERT ROW
+/- n	: moves forward (+) or backward (-) n tuples, where n is an integer

Table 8.3.1

8.4 GLOBAL UPDATE

The command ROW CALCULATIONS may be chosen from the menu to make mass changes to a relation i.e. to effect global updates. This is accomplished through a statement of the form

if condition then calculation

which causes the specified calculation to be performed for all tuples which satisfy the condition. The 'calculation' permits the user to compute a new attribute value from values of existing attributes. Very general computations may be performed involving numeric and literal comparisons, arithmetic operations and Boolean combination. It appears that ROW CALCULATIONS is actually used to change values of an existing attribute; however, combining this capability with the extended projection operation in RMS it is clear that the user can define a new relation which will contain all of the attributes of the original relation plus any new attribute(s) created via ROW CALCULATIONS. We shall return to this command when we discuss the domain algebra in section 8.6.

8.5 BINARY OPERATIONS

The JOIN menu option supports the same family of joins as in RQL i.e. the θ -joins of ALDAT, plus the ability to use multiple comparisons in the join condition. The RMS user's manual provides full details on how to create the result relation and how to specify the condition. The implementation is explicitly stated to be based on the Cartesian product plus selection logic, and a warning is issued that joining large relations may be slow. Some hints are provided for speeding up the join in special cases.

Four set operations are provided using the menu options INTERSECTION, UNION, DIFFERENCE, and TABLE MINUS TABLE. For example consider the relations FIRST and LAST (table 4.2.1, page 26). To find all customers whose first and last orders were placed in the same year, use the INTERSECTION command as follows - underlined words are printed by the computer:

RESULTANT TABLE NEW
TABLE 1 FIRST
TABLE 2 LAST

with the result

NEW(NAME YEAR)

C	1982
D	1970

The other set commands are used in a similar fashion.

8.6 DOMAIN ALGEBRA

The vertical domain algebra is implemented via the STATISTICS menu option. Once the relation and attribute (numeric or dollar data) are specified RMS computes and displays the following quantities:

NUMBER OF OBS
TOTAL
MIN OBS
MAX OBS
RANGE
MEAN
POPULATION VAR
SAMPLE STANDARD DEV.

No options are permitted here and all of the above quantities are displayed whenever STATISTICS is used. There is no provision for equivalence reduction operations such as subtotalling. In this respect the domain algebra of RMS is weaker than that of RQL. On the other hand RQL makes no attempt to implement the horizontal domain algebra whereas RMS, as mentioned previously, permits the creation of new attribute values via ROW CALCULATIONS and new attributes in which to store these values via the DEFINE TABLES and PROJECT COLUMNS menu options. This provides for a flexible implementation of the horizontal domain algebra.

ALDAT	:	RMS	:	SYNTAX AND COMMENTS
QT-SELECTORS				
PROJECT	:	>	:	PROJECT COLUMNS (may add extra attributes to existing relation)
SELECT ON GENERAL TUPLE CONDITION	:	=	:	SELECT ROWS
QUANTIFIERS	:	NO	:	
UPDATES	:	YES	:	ROW CALCULATIONS(permits changes to tuple via an if...then statement)
μ - JOIN				
$\cap$	:	>	:	JOIN (Boolean combination of comparisons)
$\cup$	:	<	:	UNION } set theoretic
+	:	<	:	DIFFERENCE } operations only
-	:	<	:	TABLE MINUS TABLE }
LEFT	:	NO	:	
σ - JOIN				
$\subseteq$ $\not\subseteq$	:	NO NO	:	
$\supseteq$ $\not\supseteq$	:	NO NO	:	
$\cap$ $\not\cap$	:	NO =	:	
DOMAIN ALGEBRA				
HORIZONTAL	:	<	:	ROW CALCULATIONS + PROJECT COLUMNS
REDUCTION	:	<	:	STATISTICS(NUMBER OF OBS, TOTAL, MIN OBS, MAX OBS, RANGE, MEAN, POPULATION VAR, SAMPLE STD DEV
EQUIVALENCE REDUCTION	:	NO	:	
EDIT	:	=	:	INSERT ROWS UPDATE ROWS

9. SEQUITUR

9.1 INTRODUCTION

Sequitur is described as "an easy-to-use relational database system with word-processing facilities". it requires the UNIX operating system. Rather than use the traditional database language approach to invoke commands, Sequitur employs a visual approach which involves full-screen editing techniques and a command menu. The 'command screen' displays a list of operations; at the bottom of the screen there is a prompt, "command:", and the user responds by typing the name of the desired command. The essential idea here is that all information is displayed on the screen in the form of a table. For example each relation has an associated "columns table" which displays information on the names, types etc. of the attributes in the relation. The following example is taken from Sequitur's reference manual (Sequitur,1982):

Database: Inventory Table: Columns

Column Name	Type	Height	Width	Lines
+Part No.	Number	1	10	11
+Part Name	Text	1	15	1
+List Price	Money	1	8	1
+Description	Text	2	20	5

Table 9.1.1

Similarly there is a 'Tables table' which displays information on all relations (tables) in a database. These tables may be edited and entries may be inserted, changed or deleted using the full-screen editor. When these two tables are filled the user has created a database which contains "empty" relations. The next step is to enter data into the relations. To Sequitur a database is a collection of named tables. Each database can contain as many as 750 relations and the number of databases is limited only by the storage capacity of the computer system.

Sequitur provides two ways of displaying data viz. Table Style and Page Style. In Table Style the user sees a number of rows with column headings at the top - this is the usual relational view of the data. By pressing two keys Sequitur allows the user to change the display to Page Style which presents the data one row (tuple) at a time and in a different format i.e. with attribute names on the left side of the screen followed by the actual values. For example the first tuple of the relation

SALESMAN(	NAME	AGE	SALARY	REMARKS)
	+Brown Er	40	50000	An aggress
	+Adams Ph	32	29000	Somewhat u

Table 9.1.2

when displayed in Page Style is as follows:

NAME: Brown Eric Michael
AGE: 40
SALARY: 50000
REMARKS: An aggressive salesman with an impressive record. He won the top salesman award in 1972 and again in 1981.

Table 9.1.3

The advantage of Page Style is that it allows the user to enter and edit pages of text and makes word-processing an integral feature of the system. The number of pages of text that can be entered in a row is limited only by the hardware. This approach is unique among DBMS's which we have examined and it is a resourceful method of overcoming the often irritating limitations imposed by fixed field widths. One can return to Table Style at any time by pressing the same two keys.

All operations in Sequitur, whether related to choosing commands, creating relations or formatting reports for output, are accomplished by adding to or editing information in tables which appear on the screen.

Before we proceed to the relational algebra it is necessary to note that in Sequitur "information is not duplicated as it is manipulated; it is stored in one place, the original data table. When you perform a selection or some other database command, you are actually creating a table consisting of 'pointers' to the selected rows in the data table. A table of pointers is called a virtual table". And further "in Sequitur, any editing operation is performed directly on the original information and all the pointers which you or other users have created in virtual tables then point to the changed information .... all Sequitur's operations work on virtual tables so you can perform any sequence of operations and output the results at any point".

An explicit SORT command may be selected from the command menu to implement ascending or descending sorting on the attributes specified in the SORT template which is presented to the user. Nested sorting is supported in Sequitur.

9.2 UNARY OPERATIONS

In Sequitur, selection may be either MANUAL, which displays tuples one at a time and lets the user specify whether or not to include them in the result relation, or SELECT FROM TABLE, which presents the user with a template i.e an empty relation. The user then employs the full-screen editor to fill in the template putting in the values which are required and leaving blank the attributes on which no conditions are imposed. For example if we want to select all the salesmen whose salary is less than $40,000 from the SALESMAN relation of table 9.1.2 we would fill in the template as follows:

Database: PERSONNEL Editing Template: SALESMAN

CONTROL	NAME	AGE	SALARY	REMARKS
+			<40000	
+				

Table 9.2.1

We see that a template row is used to specify a general tuple selection condition such as might be realized with a 'where' or 'if' clause in more traditional DBMS environment. A template row can describe one or more conditions for each attribute of the row. Conditions may involve arithmetic and literal comparisons and the Boolean operation ' ' for 'NOT'. For example the condition >H<N informs Sequitur to include all tuples that have text beginning with the letters I,J,K,L,M in the specified attribute.

If we look again at the template, table 9.2.1, we see a CONTROL column on the left. The 'control' conditions are SLICE, GROUP, OR, NOT. SLICE allows the user to specify the attributes from the input which will be included in the resulting virtual table. Attributes may be omitted and those that remain may be arranged in any order by placing sequence numbers in the appropriate columns of the SLICE row of the template. The SLICE operation is Sequitur's implementation of projection. The control word GROUP is used to specify one or more attributes; Sequitur will then group together all tuples which have a common value in the specified attribute(s). When the select is performed the entire group will be included in the output if any tuple in the group matches the template. This generalizes selection as envisaged in ALDAT and provides a useful mechanism for classifying tuples by membership in a (user defined) group. The control word OR is placed between two rows in a template and is used to tell Sequitur to select all tuples that satisfy either the previous conditions or the following ones. NOT excludes any tuple of the input relation which matches the template row. If groups are specified, any group which contains

such a row is excluded. The manual contains the following example which exhibits several of the features discussed above:

Database: PEOPLE			Editing Template: SALARY	
CONTROL	NAME	SALARY	TITLE	YEARS
+SLICE	1	2		
+		>25000		
+NOT	Jones			
+OR				
+				>=7
+				

table 9.2.2

This template for the relation SALARY is Sequitur's formulation of the query "find the name and salary (in that order) of each employee whose salary exceeds $25000, excluding Jones, as well as those employees who have been with the company for at least seven years". The combination of selection and projection is equivalent to the T-selector discussed in ALDAT.

The UNIQUE command operates on specified column(s) of a single input relation. It makes a result relation consisting of tuples with unique entries in the specified column(s). Thus if two or more tuples have identical values in the specified attribute(s), the result includes only the first such tuple. The DUPLICATE rows command is similar to UNIQUE except that it finds all the tuples that have the same values for the specified attribute(s). The result relation consists of groups of two or more tuples with identical values in the specified attribute(s). The user's manual states that "DUPLICATE is useful for locating and deleting rows inadvertently entered twice".

9.3 RELATIONAL EDITOR

The EDIT command allows the user to insert, change or delete tuples. It also allows one to recover deleted tuples or restore changed tuples to previous states. When attribute values are changed Sequitur saves the new version of the tuple but also retains the old version; previous versions are kept until the user explicitly asks Sequitur to discard them. Previous versions may be examined and used to replace the current version at any time. The MANUAL select menu option is also an interactive editing command which provides the user with the ability to include or exclude a particular tuple in a result relation.

9.4 GLOBAL UPDATE

Sequitur has limited global update capabilities. The 'mass changes' which are available are REMOVE rows and RENAME columns. The manual describes these operations as follows: "REMOVE rows will take a table which is the result of a database command and remove the corresponding rows from the base table. If you have used the same column name in more than one table, RENAME columns allows you to change the name in one of the tables."

The APPEND command may be used to add a copy of one relation to the end of another. The result is two relations: "one, which is unchanged, and another, which has become a larger table containing both its original rows and the rows from another table".

9.5 BINARY OPERATIONS

The JOIN command combines relations by matching tuples from the two operand relations if they have the same entries in the attribute(s) specified in the command. Once again the command is entered by filling a template which shows both relations side by side. The join appears to be a simple equi-join: the possible control conditions in the template are MATCH, which specifies the attributes on which the join is to be performed, and SLICE which allows one to project only the required attributes in the result relation. Thus one can use the JOIN operation to implement a natural composition. Sequitur asks the user to enter the names of the two operand relations and also the name of the result relation: it also asks if you desire an 'outer join'. The manual states "an outer join means that every tuple in the first relation is included in the result relation whether or not it has a match in the second relation". In this case the attributes in the result relation which are derived from the second input relation are left blank. This corresponds to the left join or ljoin in ALDAT.

Set theoretic commands are UNION, INTERSECTION and DIFFERENCE. THe UNION command produces a result relation which consists of all the tuples from the first relation followed by all the tuples from the second relation. Sequitur's UNION is more general than simple set union in that the operand relations need not have identical attributes. The manual states that "the column names from the first table will be followed by those from the second table, left to right". Inevitably this implies that some of the entries in each row will be blank. The INTERSECTION

command finds tuples in two relations which have identical entries in specified attribute(s) and puts the tuples from the first relation that have a match in the second relation into a result relation which consists only of the attributes in the first relation, and only of those tuples in the first relation for which there is a match in the specified attribute(s) of the second relation. This is the 'left semi-join' and is equivalent to an intersection join followed by projection on the attributes of the first relation. Sequitur sorts the result relation on the matched attribute. The DIFFERENCE command operates in a similar fashion but the result relation contains only those tuples from the first relation which do not match entries in the specified attributes of the second relation: this corresponds to a left difference join, djoin, in ALDAT.

9.6 DOMAIN ALGEBRA

The domain algebra operations of Sequitur are described under the heading "output commands". The "Functions Table" has the format:

DO	TO	GROUPED BY	NEW NAME	SPACES
+				

The arithmetic function to be performed is entered in the DO column and the attribute on which it is to be performed is entered in the TO column. The GROUPED BY column is used to specify which attribute(s) in the relation are to be 'grouping attributes' - Sequitur groups together all tuples with identical entries in these attributes. In the NEW NAME column the user enters a label to be printed under the grouping attribute in the output report. Finally the SPACES column is used to specify the number of blank lines to insert between results i.e. it is used for carriage control. The available DO options are:
T(TOTAL), MI(MINIMUM), MA(MAXIMUM), A(AVERAGE), C(COUNT).
If one types 'T' or 'TOTAL' in the DO column, Sequitur will compute and print the total of all attributes specified in the TO column for each group of tuples i.e. it prints subtotals by groups. One can also request grand totals, averages etc. by leaving the GROUPED BY column blank. We note that these features are available as output operations and not in the algebraic sense of creating new relations which become a part of the database.

Sequitur's pamphlet (Mini/Micro 82) summarizes this discussion, under the heading 'Arithmetic Capability' as follows: "Sequitur has full arithmetic capability, including user defined computations for queries, global updates and

reports. Tables may have columns with calculated values. Arithmetic expressions may be used in select and join templates. Selections can also be based on relationships between columns in a row. Arithmetic expressions can be used anywhere Sequitur will accept a number. Reports can have columns which are calculated from other columns in a row, and calculated values can be printed anywhere on a form".

In summary we note that Sequitur has an original and creative approach with some excellent features, notably integrated word processing and a consistent visual technique for command specification via table displays and full screen editing. While we have seen claims that Sequitur is "relationally complete" we find no reference to a command for relational division in the reference manual. Also it is not clear from the manual if the features of the horizontal domain algebra are available.

ALDAT	:Sequitur:	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	=	SLICE
SELECT ON GENERAL TUPLE CONDITION	=	SELECT
QUANTIFIERS	NO	
UPDATES	NO	the 'mass' changes named in the manual are REMOVE ROWS and RENAME COLUMNS
μ - JOIN		
$\cap$	=	JOIN INTERSECTION (ijoin + project rel1)
$\cup$	<	UNION (generalizes set union - appends tuples from second relation to end of first relation - attributes need not match)
+	NO	
-	<	DIFFERENCE (djoin + project rel1)
LEFT	NO	
σ - JOIN		
$\subseteq$ $\not\subseteq$	NO NO	
$\supseteq$ $\not\supseteq$	NO NO	
$\circ$ $\not\circ$	NO =	
DOMAIN ALGEBRA		
HORIZONTAL	?	:Sequitur brochures claim that horizontal domain algebra operations are available. The user's manual does not indicate how to realize them.
REDUCTION	<	:using the FUNCTIONS TABLE one can specify a DO option which may be T(TOTAL), MI(MINIMUM),MA(MAXIMUM),A(AVERAGE),C(COUNT).
EQUIVALENCE REDUCTION	<	:The DO command will subtotal etc. if a GROUPED BY clause is included.
EDIT	=	EDIT invokes the full screen relational editor. One can insert, delete or update as well as restore tuples to previous states.

10 MICRORIM

10.1 INTRODUCTION

MicroRIM operates on any of the following microprocessor systems: 8080, 8085, Z-80, APPLE II or III with Z-80 card. It requires a CP/M (release 2 or higher) or MSDOS operating system. MicroRIM requires 52000 bytes of main memory. The minimum secondary storage is one floppy diskette drive with 152 K bytes.

Data security is provided via password types which may be specified as OWNER or USER with subclassification as READ or MODIFY passwords. This prevents unauthorized users from changing data in the database. Once a database is defined using the define DBNAME command the attributes command lets the user describe each attribute in the database. The user's manual states "all attributes typically are described once and then combined into relations or tables later using the relations command". The attributes command has the syntax

attname type [length] [key]

where attname is the name of the attribute, type defines the data type, length specifies the field width and key declares the attribute to be a key field. Square brackets, as usual, indicate optional choices, braces specify compulsary choices.

Keys are optional. In MicroRIM a key field (or attribute) is a search key rather than a key in the relational sense. When an attribute is specified as a key field MicroRIM builds an index (inverted) file using the B-Tree technique for that field. This permits faster searches for locating tuples with particular values of the key attribute. Attributes can be changed from key to non-key and vice-versa using the delete key and build key commands, respectively. The syntax for these commands is

delete key for A in R
build key for A in R

The command relations is used to combine attributes to form relations. After entering the relations command, each relation is defined in a single statement using the format

RELNAME with attname1 [attname2 .....] .

Up to 20 relations may be defined in a single database.

The command

load R

is used to enter tuples to a new relation or to add tuples to a relation which already exists.

10.2 UNARY OPERATIONS

Both projection and selection are combined in a single command in MicroRIM. The command, project, has the following syntax

project R from S using { A,B,.. / all } [sorted by C,D,..]
[where clause]

Both where and sort clauses are optional. These clauses specify the tuples and their order, respectively, in the new relation. The where clause has the form

where condition1 [{and / or} condition2 .....] .

Up to 10 conditions for tuple selection can be included. These conditions are quite general and include comparison of attributes. The mandatory using clause specifies the attributes and their order in the new relation. All specifies all attributes and their existing order from the second input relation. In this case the command becomes simple tuple selection on the specified where condition. Project does not automatically eliminate duplicate tuples so that the delete duplicates command may be needed to tidy up the result relation.

MicroRIM also has an explicit select command, but it is non-algebraic in that it is strictly an output command. Its general syntax is

select A,B,... from R [sorted by C,D,..][where clause] .

and it provides essentially the same capabilities as project though strictly for output since no relation is created as a result of this command.

10.3 RELATIONAL EDITOR

Interactive editing is performed by typing RIMEE which invokes the relational editor. The editor has two modes - L for load, E for edit. If L is used a screen showing each data field and its type is presented; underlines indicate the positions on which to enter data. In edit mode the user is requested to enter a where clause to select the tuples to be modified. Tuples satisfying the where clause will then be presented one at a time and the user can type over, delete or insert characters using cursor control.

10.4 GLOBAL UPDATE

Global editing is possible with the change command:

change A to value in R where clause.

For example the relation ROUTE (table 2.2.1,page 6) may be modified with the command

change DEL# to 7 in ROUTE where SUPPLIER eq Green

to yield the result

ROUTE (DEL# SUPPLIER DEST)

DEL#	SUPPLIER	DEST
4	Adams	Brown
7	Green	Daly
1	Fingal	O'Neill
7	Green	Daly
5	Cowan	Stanley
2	Adams	White

Table 10.4.1

The command assign is similar to change. The manual states that "assign allows the user to change the value of an attribute by assigning an additional value to that same attribute". The syntax is

assign A to value of attrib2 op value of attrib1 in R
[where clause]

where 'op' can be +, -, * or /. The value of attrib1 must be an integer, real or dollar data type. The manual gives the following example:

```
SALARY (NAME    WAGE )
        Tom     50.00
        Dick    75.00
        Harry   65.00

assign WAGE to WAGE + $5.00 in SALARY

SALARY (NAME    WAGE )
        Tom     55.00
        Dick    80.00
        Harry   70.00
```

10.5 BINARY OPERATIONS

The join command has the syntax

join R using A with S using B forming T[where operator]

The operator in the where condition may be EQ, NE, GT, GE, LT, LE. The attributes on which the join is to be performed must be named. If the where expression is omitted EQ is assumed, resulting in the intersection (equi) join. Clearly the join in MicroRIM is limited to the θ -joins in ALDAT. The join will execute much faster if the EQ comparison is used and the attribute named in the second relation is key.

The command

intersect R with S forming T [using A B C ...]

forms a new relation where the tuples of R and S have common matching attributes. If the two input relations have identical attributes this is the set theoretic intersection. If the using clause is omitted all attributes from both operand relations will be in the result relation. Common attributes will be listed only once. "MicroRIM compares each tuple of S with all tuples of R using all the common (same name) attributes in each relation. When a matching condition is found, attributes from both relations 1 and 2 are transferred to the new relation". This describes the ijoin command in ALDAT.

The subtract command

subtract R from S forming T [using A,B,.....]

creates a new relation where the tuples of S have no matching common attributes with tuples of R. The manual states that

"subtract is the opposite of the intersect command". It is equivalent to the right difference join (drjoin) in ALDAT.

The union command

union R with S forming T using A,B,....

"forms a new relation T similar to the way the intersect command combines relations except in how the union command handles rows which do not match (on common attributes): whereas the intersect command discards rows which do not match on common attributes, the union command includes them". Missing values are filled with the null value -0- in the result relation. The attributes used in the matching process are common to all three relations. The manual claims that "this command will operate much faster if there is an attribute common to both relations which is key". The following example comes from the manual:

EMP-DATA (

NAME	SHIFT	POSITION	BOSS
Smith	1	Clerk	Ho
Kay	1	Sales	Ho
Hays	2	Maint	Wagner
Buck	3	Maint	Wiley

)

D-NAME (

BOSS	TITLE	DPT-NM
Ho	Mgr	Mktg
Jones	Spv	Mfg
Wagner	VP	Plant

)

union EMP-DATA with D-NAME forming E-D-DATA
using NAME BOSS DPT-NM

E-D-DATA (

NAME	BOSS	DPT-NM
Smith	Ho	Mktg
Kay	Ho	Mktg
Hays	Wagner	Plant
Buck	Wiley	-0-
-0-	Jones	Mfg

)

We see that MicroRIM's union command is identical to union join in ALDAT and becomes set union if both relations have identical attributes. Note that union can be used to add a new attribute to an existing relation by creating a temporary relation with an extra 'empty' attribute of the type required and then using the union command to transfer the data from the old relation into the new one. Full details, including an example, are given in the manual.

10.6 DOMAIN ALGEBRA

The command

tally A from R where clause

is used to print out the distribution of values for an attribute: each distinct value and the number of occurrences is reported. For example, referring to table 10.4.1,

tally SUPPLIER from ROUTE

will cause the following output:

SUPPLIER	NUMBER OF OCCURRENCES
Adams	2
Cowan	1
Fingal	1
Green	2.

Operations of the (vertical) domain algebra are provided by the command

compute function A from R [where clause] .

possible functions are count, min, max, sum, and all. Count gives the number of non-missing values for the attribute in tuples which satisfy the where clause. The other names are self-explanatory. The ability to include the where clause permits some simple reduction (totals) and equivalence reduction (subtotals) commands to be performed.

The assign command mentioned earlier may be combined with the union command to change values in a specified attribute and then add that attribute to another relation. This capability appears to be MicroRIM's only attempt to implement the horizontal domain algebra.

ALDAT	:MicroRIM:	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	: =	: project R from S using {A,B,...} / {all} [sorted by D,E,...] [where clause]
SELECT ON GENERAL TUPLE CONDITION	: =	: project R from S using all where clause
QUANTIFIERS	: NO	:
UPDATES	: YES	: change A to value in R where clause
		: assign A to val2 op val1 in R where clause
		:(op may be +,-,/,*. vali refers to a value of attribute i)
μ - JOIN		
∩	: =	:join R using A with S using B forming T where operator
		: (operator may be EQ, NE, GT,GE, LT, LE - with the default EQ yielding the equi-join)
		:intersect R with S forming T [using A,B,..]
		: (generalized set intersection)
∪	: =	: union R with S forming T [using A,B,..]
		: (generalized set union)
+	: NO	:
-	: =	: subtract R from S forming T [using A,B,..]
LEFT	: NO	:
σ - JOIN		
⊆ ⊈	: NO NO	:
⊇ ⊉	: NO NO	:
∩ ∅	: NO =	:

ALDAT	:MicroRIM:		SYNTAX AND COMMENTS
DOMAIN ALGEBRA			
HORIZONTAL	:	<	:assign A to val2 op val1 in R where clause
	:		:(op may be +,-,/,*) provides limited horiz-
	:		:ontal domain algebra
REDUCTION	:	<	: tally A from R where clause
	:		: compute function A from R where clause
	:		: (functions: count,min,max,sum,all)
EQUIVALENCE	:	<	:
REDUCTION	:		:
EDIT	:	=	: RIMEE (invokes relational editor)
	:		: tuples are selected via a where clause and
	:		: are edited interactively.

11 RL-1

11.1 INTRODUCTION

RL-1 operates under IBM DOS on the IBM PC. The user's manual is very well written and is an excellent introduction to the relational approach to data management. It includes many examples and the first time user of RL-1 (or any other relational system) would find it useful. The commands in RL-1 fall in one of three categories :

1. The Data Manipulation Language (DML) incorporates commands for creating and manipulating relations, including the relational algebra.

2. The Relational Editor

3. A program Interface which permits access to application programs in high level languages.

A key in RL-1 is a true relational key.

10.2 UNARY OPERATIONS

All of the commands of the relational algebra include the option of projecting on a desired set of attributes via the keyword select. However the select command is very versatile and encompasses projection, joins etc. The general syntax for the unary operations is:

$$\underline{\text{select}} \ \left\{ \begin{matrix} A,B,\ldots \\ * \end{matrix} \right\} \ \underline{\text{from}} \ R \ [\underline{\text{where}} \ \text{clause}] [\underline{\text{ordered by}} \ [-] F,[G,H\ldots]] \ \left[\begin{matrix} \underline{\text{create}} \ S \\ \underline{\text{insert}} \ S \end{matrix} \right]$$

If the where and ordered by clauses are omitted the command is simple projection on the attribute list or on all of the attributes (i.e. display the entire relation) if the asterisk is used instead of an attribute list. The where clause permits tuple selection on the usual logical condition involving Boolean combinations of numeric and string comparisons. Simple selection (i.e. without projection) is effected by using a * and including a where clause. The user's manual is careful to explain that "multiple key attributes do not allow the definition of one primary order inherent in the data" but goes on to acknowledge that it is often necessary to order the data for a clearer presentation - hence the ordered by clause which causes

lexicographic ordering on the attribute list specified in the clause; any number of attributes is allowed. Inclusion of the optional minus sign causes sorting in descending order. The create phrase specifies a name for the result relation. Note that if the create phrase is not included the result is for output only and no new relation is created. On the other hand when create is used the result relation is given the specified name and saved, but is not automatically displayed on the screen. To generate screen output use the statement "select * from S".

While create S directs the result of the select command to a new relation S, the insert S option causes RL-1 to append the result to an existing relation. Thus the insert phrase, although part of the general select command, posts new tuples to an existing relation and is therefore a type of update command.

11.3 RELATIONAL EDITOR

The full screen relational editor REDIT supports interactive editing and includes update and delete commands to change values at the current cursor location or delete the current tuple, respectively. The command 'cancel record' is used to cancel all updated values on the current tuple and reinstate the original values. The insert command will insert a new tuple with default values which are then updated to contain the desired values. The find and search commands are used to locate tuples which have specified values for attributes: find works only on key attributes while search can be applied to any attribute, resulting in considerably longer execution times. Find may be used with various modifiers such as next, previous, maximum and so forth. Full details are given in the user's manual.

10.4 GLOBAL UPDATE

The update command may be used to change the data in a single relation. Thus

update A = exp [,B = exp,....] from R [where clause]

will cause the specified attributes in relation R to be changed in accordance with the attribute expressions for tuples satisfying the where clause. For example, if all egg deliveries are to be decreased by twenty percent due to an egg shortage,

then the appropriate command for the DELIVERY relation (table 2.1.1) would be

```
update QTY = QTY - (QTY*0.2) from DELIVERY
             where PRODUCT eq "Eggs".
```

Another mass change which can be used is

delete from R where clause

which will modify R by deleting all tuples which satisfy the where clause.

11.5 BINARY OPERATIONS

The join command in RL-1 has the form

select { A,B,... / * } from S joined with T over D [=E]

The attribute on which the join is to be performed is specified in the over clause. If the attribute has the same name, D say, in both input relations then "over D" is sufficient whereas "over D = E" is required if the attribute is named D in one relation and E in the other. This is a limited form of the natural join since the join can only be performed over a single attribute. Other versions of the θ-join family are not implemented in RL-1.

11.6 DOMAIN ALGEBRA

Horizontal and vertical commands of the domain algebra are available in RL-1. In order to create a new attribute which is calculated from existing attributes simply include the appropriate expression in the form [A=] exp in the attribute list. Consider the relation

FOODS

PRODUCT	QTY	UNIT-PRICE
Milk	5	1.50
Bread	2	0.75
Jam	1	1.10

Table 11.6.1

select PRODUCT , (QTY*UNIT-PRICE) from FOODS

yields the result relation (output only)

PRODUCT	(QTY*UNIT-PRICE)
Milk	7.50
Bread	1.50
Jam	1.10

Table 11.6.2

Note that the expression (QTY*UNIT-PRICE) is itself used as the attribute name. In order to replace this with some meaningful name such as TOT-PRICE simply write

select PRODUCT , TOT-PRICE = (QTY*UNIT-PRICE) from FOODS

to get the same relation as in table 11.6.2 but with the expression (QTY*UNIT-PRICE) replaced by the column heading TOT-PRICE.

Vertical domain algebra is implemented with the command

select statistic [A=] exp from R

where the statistic may be any one of the following:
sum, avg, count, max, min, sd.
The names are self-explanatory (sd stands for standard deviation). For example

select sum (QTY*UNIT-PRICE) from FOODS

yields the result

SUM (QTY*UNIT-PRICE)
10.10

Note that this result, though only a single value, is output in the form of a relation in accordance with the relational approach. In order to produce a result for each subgroup of a relation use a grouped by phrase as in the example, based on DELIVERY (table 2.1.1):

select SUPPLIER , sum(QTY) from DELIVERY grouped by SUPPLIER

which yields

SUPPLIER	SUM(QTY)
Adams	105
Green	121
Fingal	49
Cowan	31

Table 11.6.3

The result shows subtotals of QTY for each supplier. When using the grouped by phrase it may be desirable to restrict the output to those aggregrate values of each subgroup which satisfy a certain condition. In this case the having clause is used. For example

```
select SUPPLIER , sum(QTY) from DELIVERY grouped by SUPPLIER
                          having sum(QTY) gt 100
```

has the result

SUPPLIER	SUM(QTY)
Adams	105
Green	121

We conclude that RL-1 implements a reasonable subset of the relational and domain algebras. It adopts a data management approach which is consistent with the relational philosophy and avoids ad hoc procedures. The join command is limited to the natural join (ijoin) and there is no explicit reference to set theoretic commands such as union and intersection.

ALDAT	RL-1	SYNTAX AND COMMENTS
QT-SELECTORS		
PROJECT	=	select {A,B,...} from R [where clause]
		[ordered by [-] F,G,...] [create S / insert S]
SELECT (N GENERAL TUPLE CONDITION	=	The select command is a T-selector and encompasses both projection and selection.
QUANTIFIERS	NO	
UPDATES	YES	update A=exp[,B=exp,...] from R [where clause]
		NOTE: using the insert clause in the select command causes tuples to be posted to a rel.
μ - JOIN		
∩	<	select A,B,... from S joined with T over D[=E] .
∪	NO	
+	NO	
-	NO	
LEFT	NO	
σ - JOIN		
⊆ ⊈	NO NO	
⊇ ⊉	NO NO	
∩ ∅	NO =	
DOMAIN ALGEBRA		
HORIZONTAL	=	create new attributes by including the required expression in the attribute list.
REDUCTION	<	select statistic[A=] exp from R[grouped by B] [having exp]
EQUIVALENCE REDUCTION	<	(statistic may be sum,avg,count,max,min,sd) (grouped by implements subtotals etc.) (having restricts the result to statistics which satisfy a given condition).

ALDAT	RL-1	SYNTAX AND COMMENTS
EDIT	<	: REDIT invokes the full screen editor. Tuples : may be inserted, changed, deleted and : restored to the pre-edit condition. : :Find and search are used to locate tuples :which have specified attribute values. Find :can be used only with key attributes.

12. CONCLUSION

The database management systems which we have considered in this paper are all relational systems. Each one implements some of the operations of the relational algebra. Only MRDSA can claim relational completeness. Among the nine systems we find MRDSA, RQL and RMS to be products of an academic environment and as such to be well suited for pedagogical and research applications. The other systems: CONDOR, dBASE II, LOGIX, MicroRIM, RL-1 and SEQUITUR are intended for the commercial marketplace and are oriented toward business applications. Consequently these systems all include user-friendly features such as report writers, help menus and so forth. We have chosen not to investigate these features in detail in the present paper since our objective was to compare relational features of DBMS packages.

We find great diversity in the terminology employed in the systems; for example the relational operation of projection is variously called project, copy, slice, and select. The implementations are equally diverse with some systems using a high-level query syntax, others employing a menu approach (RMS, SEQUITUR) and one providing an integrated word-processing capability (SEQUITUR).

The variety and richness of the implementations is quite impressive in view of the short history of relational DBMS packages for microcomputers. No doubt this reflects an attempt to get into the rapidly growing database management market at an early stage and the emergence of so many viable packages in such a short time is tribute to the ingenuity and energy of the system developers. However there is a negative aspect to this rapid growth which resides in the fact that the attempt to develop systems with commercial appeal has resulted in the addition of ad-hoc procedures which seem to add little to the overall effectiveness of the resulting systems. Quite the contrary, this results in a departure from the relational philosophy.

It is our conviction that, in the long run, systems which are implemented so as to adhere strictly to the rules of the relational algebra, and which are relationally complete, will be the most likely to become established in the marketplace and will be the eventual winners in this increasingly important and exciting field.

References

ABW Corporation, 1982. Sales leaflet for RL-1.

ABW Corporation, 1982. RL-1 Relational Database Management System Users manual, version 1.02. (July 1982).

Ashton-Tate, 1981. dBASE II Assembly-Language Relational Database Management System.

K.S. Barley and J.R.Driscoll, 1981. A survey of data-base management systems for microcomputers. Byte (Nov. 1981) 208-234.

S. Bonthron and M. Darnovsky, 1982. Sequitur Reference Manual, version 3.11. (Nov. 1982).

D. Chen and J.R.Driscoll, 1982. RQL User's Manual (release 1,2). Hello Software.

G. Chiu, 1982. MRDSA User's Manual, Technical Report 82.9, McGill University School of Computer Science.

E.F. Codd, 1970. A Relational Model of Data for Large Shared Data Banks. CACM 13, No. 6. (June 1970).

E.F.Codd, 1971. Relational completeness of data-base sublanguages in R. Rustin ed. Data Base Systems, Prentice-Hall 1972, 65-98.

Condor, 1981. Condor Series 20/rDBMS User's Operation Manual, Release 2-04. (June 1981).

R.H. Edelson, 1981. dBASE II relational DBMS for CP/M. Interface Age (Aug. 1981) 60-63.

Hello Software, 1982. Sales Leaflet for RMS/RQL.

Hello Software, 1983. RMS User's Manual.

Logical Software, 1981. LOGIX Relational Database Management for UNIX Tutorial (Nov. 1981)

T.H. Merrett, 1977. Relations as programming language elements. Information Proc. Lett., 6, 1, 20-33.

T.H. Merrett, 1984. Relational Information Systems. Reston Publishing Company 1984.

T.H. Merrett and G. Chiu, 1983. MRDSA: full support of the relational algebra on an Apple II. Proceedings: Conference on Design, Implementation and Use of Relational DBMS on microcomputers, Toulouse 14-15 Feb. 1983.

T.H. Merrett and B.E.Smith, 1983. A Comparison of Relational Database Systems for Microcomputers. Ibid. 31-51.

MicroRIM, 1982. Sales Leaflet.

MicroRIM, 1982. User's Manual (MSDOS version) (Jan. 1983).

Pacific Software, 1982. SEQUITUR, Mini/MIcro 82, 1-13.

T. VanRossum, 1983. Implementation of a Domain Algebra and a Functional Syntax. McGill University School of Computer Science M.Sc. Thesis. Technical Report SOCS-83.