Durham University

Durham E-Theses

Some aspects of the implementation of a relational data base sublanguage

Lim, Richard Thuan Chan

How to cite:

Lim, Richard Thuan Chan (1975). *Some aspects of the implementation of a relational data base sublanguage*, Durham e-Theses. http://etheses.dur.ac.uk/8947/

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.

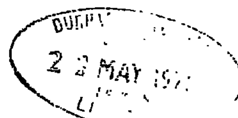
Academic Support Office, The Palatine Centre, Durham University, Stockton Road, Durham, DH1 3LE
e-mail: e-theses.admin@durham.ac.uk Tel: +44 0191 334 6107
http://etheses.dur.ac.uk

SOME ASPECTS OF THE IMPLEMENTATION OF A RELATIONAL DATA BASE SUBLANGUAGE

A thesis submitted for the degree
of Master of Science in the
Department of Computing of
the University of Durham

by

Richard Thuan Chan Lim, BSc



Faculty of Science
University of Durham

January 1975

To my parents

C O N T E N T S

	ACKNOWLEDGEMENTS	6
	ABSTRACT	7
1.	INTRODUCTION	8
	1.1 Scope of Work	8
	1.2 Historical Review	9
	1.3 Presentation of Work	13
2.	RELATIONAL MODEL, NORMAL FORM AND RELATIONAL LANGUAGES	14
	2.1 The Relational Model of Data	14
	2.2 Normalization of Relations	17
	2.2.1 Introduction	17
	2.2.2 The First Normal Form	18
	2.2.3 The Second Normal Form	20
	2.2.4 The Third Normal Form	23
	2.2.5 Objectives of Normalization	27
	2.2.6 Summary of Normalization	28
	2.3 Relational Languages	28
	2.3.1 Introduction	28
	2.3.2 A Relational Algebra	29
	2.3.3 Relational Calculus	29
	2.3.4 Relational Completeness	30
	2.3.5 Calculus versus Algebra	31

3. THE PROPOSED DATA BASE SYSTEM WITH REFERENCE TO ALPHA SUBLANGUAGE 32

3.1 Introduction 32

3.2 The Proposed Data Base System 34

3.3 Description of Implemented Data Sublanguage ALPHA 38

3.3.1 Basic Characteristics 38

3.3.2 Structure of Data Sublanguage ALPHA 41

3.3.3 Basic Symbols 43

3.3.4 Identifiers 45

3.3.5 Standard Function Identifiers 46

3.3.6 Constants 47

3.3.7 Statements 48

3.4 Data Sublanguage ALPHA Examples 59

3.4.1 Queries 60

3.4.2 Updates 68

3.4.3 Deletions 70

3.4.4 Insertions 71

3.4.5 Dropping and Establishing Data Base Relations 72

3.4.6 Input and Output Facilities 73

3.5 Summary of Data Sublanguage ALPHA 74

4. DESCRIPTION OF THE TRANSLATOR 75

4.1 Introduction 75

4.2 Translation Process 75

4.2.1 Information Tables 75

4.2.2 The Lexical Analyzer 78

4.2.3 The Syntax Analyzer 82

4.2.4 The Semantic Analyzer and Table 'Generator' 85

4.3 Implementation of the Translator 88

4.4 Warnings and Errors 90

5. EXPLANATION OF THE CODING TABLES 92

5.1 Introduction 92

5.2 Outline of the Ccding Tables 93

5.3 Description of the Coding tables 97

6. CONCLUSIONS 103

6.1 General 103

6.2 Suggested Improvements to the Translator 103

6.3 Suggestion for Further Work 106

REFERENCES 107

BIBLIOGRAPHY 109

APPENDIX A	Some Terminology Associated with the Relational Model of Data	110
APPENDIX B	Relational Algebra and Relational Calculus	113
APPENDIX C	Lexeme Values for Source Language Symbols	124
APPENDIX D	Floyd Production Language Statements	125
APPENDIX E	Implementation of the Translator	131
APPENDIX F	Examples of Tables Produced by the Translator	132

A C K N O W L E D G E M E N T S

The investigation described here was undertaken in the Department of Computing of the University of Durham with the kind permission of Dr J Hawgood, Head of the Department.

It is a pleasure for the writer to be able to record his gratitude to Mr J S Roper, Senior Lecturer in Computing, who provided the necessary supervision and constant guidance throughout the course of the investigation.

The writer is also grateful to Mr M Munro, of the Computer Unit, for his valuable advice and unfailing assistance in the development of the computer program employed in the investigation.

Special thanks are also due to Mr J Blackburn for his helpful comments on the presentation of the thesis.

Finally, the financial support received by the writer from the British Council is gratefully acknowledged.

A B S T R A C T

A relational model of data has been proposed by Codd [1] for protecting users of formatted data systems from the potentially disruptive changes in data representation caused by growth in the data base and changes in traffic. The adoption of this relational model of data has permitted the development of a universal data sublanguage, based on an applied predicate calculus, called ALPHA. Although ALPHA has been intended to be a sublanguage of the languages used by all terminal users of a shared, formatted data base, only the semantics of it has been fully described by Codd. [2] However, before ALPHA can be implemented, it is necessary to specify the syntax of this data sublanguage.

Accordingly, the work described here has involved the specification of the syntax of this data sublanguage and the development of a related PL/1 computer program, called the translator. The translator carries out the following functions:

(i) accepts as input a modified form of source statements; these statements include both data sublanguage ALPHA statements and computational facility statements

(ii) checks that conditions are met for the relational calculus; interactive debugging facilities are provided

(iii) produces as output a set of coding tables suitable as input to an interpreter; these tables, a modified version of those based on the work of Palmero, [3] are presented in a precise manner which clearly reflects the subsequent operations to be carried out.

It is intended that this work will form a basis for the development of a data base system.

1. INTRODUCTION

1.1 SCOPE OF WORK

With the increasing quantity of data in commericial and industrial enterprises, a need has arisen for the implementation of a simplified computer language to manipulate such data. The adoption of Codd's relational model of data [1] and the subsequent description of the semantics of a data sublanguage (a) (DSL) called ALPHA [2] has justified the need to specify fully the syntax of this sublanguage. This work has involved the specification of the syntax of DSL ALPHA and the development of a related PL/1 computer program, subsequently referred to as the translator. (b) This interactive translator translates DSL ALPHA statements and computational facility statements (source) into a set of coding tables (object code). The listing of the translator is given in a supplementary volume.

(a) When the computation-oriented components of a language for data base manipulation are removed, the remaining language components which support storage and retrieval of formatted data from shared data bases are referred to as a *data sublanguage*. Hence the term *data sublanguage* rather than *language*.

(b) A *translator* is a program which translates a source program into an equivalent object program. The source program is written in a source language, the object program is a member of the object language. If the source language is a high-level language like FORTRAN or ALGOL, and if the object language is the assembly language or machine language of some computer, the the translator is called a compiler.

1.2 HISTORICAL REVIEW

The implications of the corporate data base (c) have been well aired, [4] in particular the dangers of amassing and relating large volumes of data, from the point of security and the invasion of privacy.

In the conventional data processing system the content and structure of data is organized around specific applications (see Figure 1.1). Each application has its files organized to meet its own requirements although this inevitably leads to duplicated and usually inconsistent data. For example, the departments concerned with payroll, personnel selection and employee expenses would each have their own files although much of the data would be common to all three.

The disadvantages of the conventional approach became apparent when the computer was used to calculate and extract for management, information for control rather than straightforward data processing purposes. The required information had to be extracted from a number of files, then sorted and collated. The result was a large cumbersome and fragile approach known as the Integrated Management Information System - IMIS (see Figure 1.2). With this approach, the increasing number of interdependent files and programs presented a major problem such that if any one link in the chain broke, the whole system suffered.

(c) A *data base* is a collection of data structured in such a way that the currently understood relationships one to another are known.

Approaches to Data Management

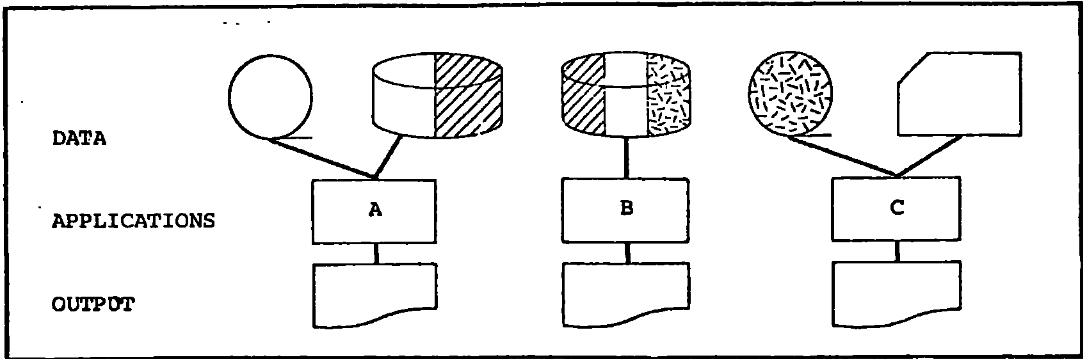


Fig. 1.1 Conventional Systems

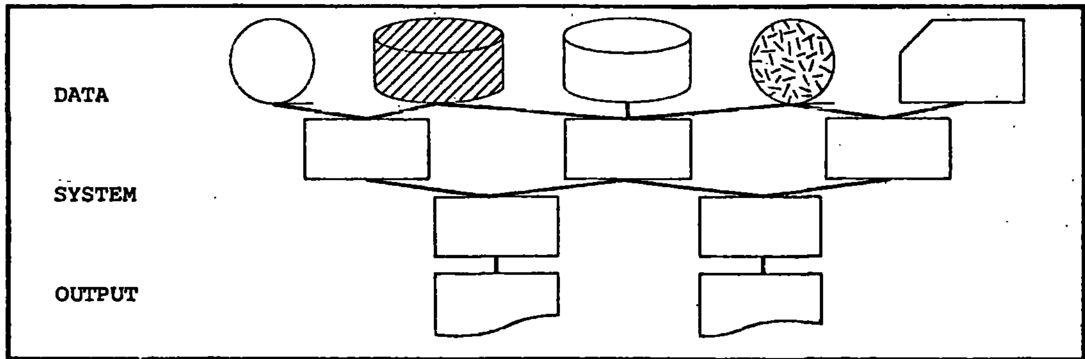


Fig. 1.2 Integrated Management Information System

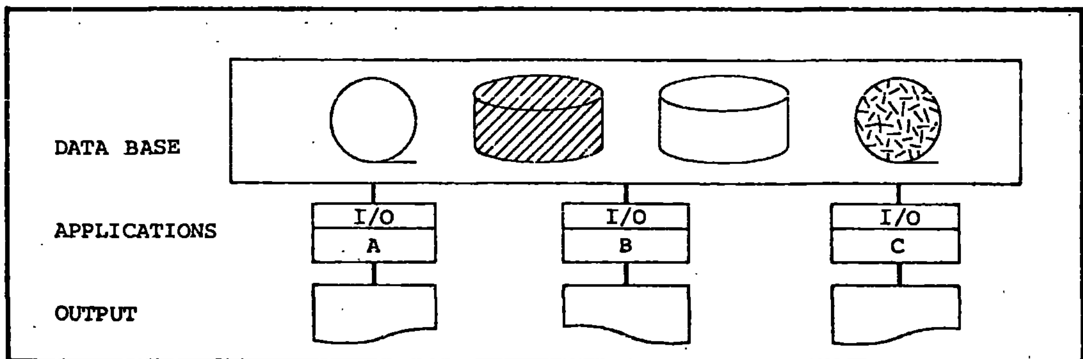


Fig. 1.3 Data Base Approach

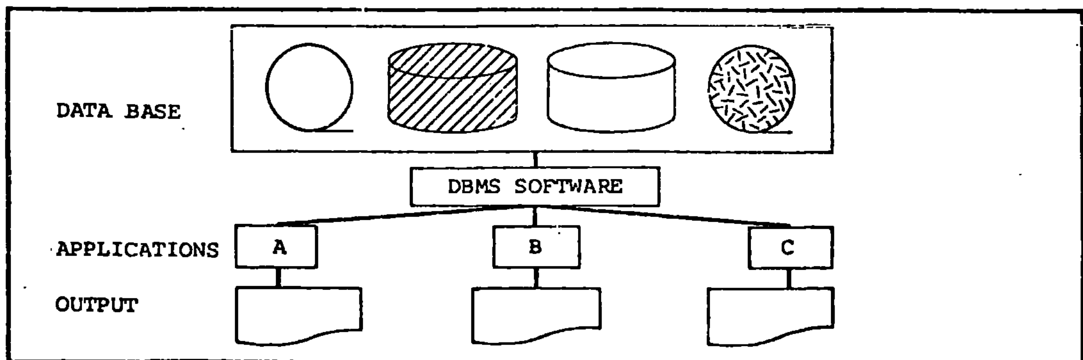


Fig. 1.4 Data Base Management System

The answer to these limitations and problems was seen to be in the corporate data base - the integration of data, rather than of applications (see Figure 1.3). This approach views the data of an enterprise in its entirety, rather than as belonging to individual applications. Each application is aware of and extracts only the data it requires from the data base. However, a major difficulty is that of structuring the data, and arranging its content, so as to meet optimally the requirements of all systems. Each time a new application is developed or an existing one changed the data base, and many applications already operational, require related changes.

The solution which represents the current stage of this technological progress is known as the Data Base Management System [d] - DBMS (see Figure 1.4). The DBMS can be regarded as the software supplied to make the data base approach feasible for all computer users. The system provides a means of defining the data independently of the application programs [e] so that changes to one application need not affect the data base and will certainly not affect other applications. It also provides standard means of accessing the data base, so that the programmer and analyst need no longer be concerned with details of file design, security and recovery procedures, and privacy controls.

(d) A *data base management system* is a method by which data is stored within, and retrieved from, the data base.

(e) *Application programs* are programs which access the data base through the data base management system.

It is already apparent that the DBMS permits a flexible system, reducing interdependence between applications and data to a manageable level, while at the same time providing the necessary integration of data. Future developments are likely to be based on improvements to the facilities provided by DBMS software. Within a few years all computer users will regard the DBMS as an essential item of software, just as now computer users take the operating system for granted.

Users of large data bases therefore need not know how the data is organized in the machine (the internal representation). Activities of users at terminals and most application programs will remain unaffected when the internal representation of data is changed and even when some aspects of the external representation (e.g., tapes, disks, etc.) are changed. Changes in data representation will often be needed as a result of changes in query, update and natural growth in the types of stored information.

To protect users of formatted data systems from the potentially disruptive changes in data representation caused by growth in the data base and changes in traffic, Codd [1] has proposed a simple tabular view of the data. The adoption of such a relational model of data has permitted the development of DSL ALPHA.

1.3 PRESENTATION OF WORK

This first section describes the development made in the field of data base management, and scope and presentation of work.

Section 2 describes the relational model of data proposed by Codd, where the application programmer and the interactive user view this data base as a time-varying collection of normalized relations of assorted degrees. Codd's concept of further normalization is described. The two principal types of language for manipulating relations are also identified. These are an applied predicate calculus and a collection of operations on relations which are termed a relational algebra.

Section 3 describes the overall strategy for the development of a data base handling system with reference to DSL ALPHA. A description of this ALPHA sublanguage together with some examples to illustrate its use are given.

Section 4 gives a detailed description of the translator.

Section 5 explains the coding tables 'generated' by the translator.

Section 6 summarizes this work and draws conclusions. Suggested improvements to the translator and suggestion for further work are included.

2. RELATIONAL MODEL, NORMAL FORM AND RELATIONAL LANGUAGES

2.1 THE RELATIONAL MODEL OF DATA

Codd and Date [5,6] have shown that casual and other users of large, formatted data bases need a simple tabular (relational) view of the data rather than a network view as typified by the proposals of the Codasyl Data Base Task Group (DBTG), or a tree-structured view. Sections 2.1 to 2.3 summarize Codd's relational model of data and the two principal types of language for accessing such data.

The relational model of data is one in which data can be logically seen in the form of data base tables called relations. The term *relation* is used here in its mathematical sense. Given sets D_1, D_2, ..., D_n (not necessarily distinct), R is a relation on these n sets if it is a set of elements of the form (d_1, d_2, ..., d_n) where $d_j \in D_j$ for each j = 1, 2, ..., n. More concisely, R is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$. D_j is referred to as the jth *domain* of R. R is said to be of *degree* n. The elements of a relation of degree n are called *n-tuples* or *tuples*. Since the null set is a subset of every set, it is possible to have relations of degree m where $m \leq n$.

The mathematical relationship *greater than* can be illustrated by the set of real numbers, X. The Cartesian product X × X is the set of all ordered pairs of real numbers. Any pair (x_1, x_2) is either true or false for

the condition $x_1 > x_2$. The subset of all pairs for which it is true defines the relation *greater than*. Definitions of the more commonly used terms together with some examples are also given in Appendix A.

A normalized [a] relation can be represented as a rectangular array with the following properties:

(i) it is column-homogenous, i.e., in any selected column the objects are all of the same kind whereas objects in different columns need not be of the same kind

(ii) each object is a number or a character string

(iii) all rows of a table must be distinct

(iv) the ordering of rows within a table is immaterial

(v) the columns of a table are assigned distinct names and the ordering of columns within a table is immaterial.

In order that users need not know the column ordering in a relation the underlying columns (domains), on which data base relations are defined, are assigned distinct attribute names. Suppose one of the underlying domains P# is a set of serial number of parts. A relation with attributes SUB_P# and SUP_P# then indicates that the role names (SUB, SUP) attached to the common stem (P#) serve to distinguish two distinct applications of the domain P# within the relation. An attribute of some other relation having the stem P# with or without the role name would also have values from this same underlying domain, i.e., serial number of parts. Additionally the data base relations must be named uniquely relative to themselves and to the attribute names.

(a) For explanation, see Section 2.2 .

One attribute (or collection of attributes) of a given relation which uniquely identifies each tuple of that relation is called the *primary key*. [b]

Basic operations (e.g., projection, join, division) [c] on relations disprove the contention that tables are inadequate data structures to support facile manipulation of relationships between and within the tables.

The relational model of data provides the following advantages:

(i) an entirely general method of manipulation of data and referring to it

(ii) a high degree of data independence [d] in that application programs and terminal activities can be made independent of changes in physical (or storage) representation of data

---

(b) Throughout the thesis the primary key of each relation is underlined.

(c) For definitions, see Appendix B.

(d) Data independence is the concept of separating the definitions of logical and physical data such that application programs and terminal activities need not be dependent on where and how physical units of data are stored. Logical data is that data as seen by the user and manipulated by the application programs; physical data is that data which the system stores on, or retrieves from some storage media. The user thus sees the elimination of several structural concepts (e.g., repeating groups, hierarchic and plex structures, and cross-referencing structures) supported by current data base systems. [5]

(iii) a simple structure [e] consistent with the semantics of the stored information; this makes it possible to use a logically simple language to interact with the data base.

To interact with this data base the user then needs to know the following:

(i) the names of the data base relations

(ii) the attribute names of each relation

(iii) the primary key of each relation (for updates).

2.2 NORMALIZATION OF RELATIONS

2.2.1 Introduction

Codd [5,7] has introduced the concept of normalization to make the collection of relations in the data base easier for the users to understand and control, and simpler to operate upon.

Normalization is a step-to-step reversible process of replacing a given collection of relations by successive collections such that the relations have a progressively simpler structure. The reversibility guarantees that the original collection of relations can be recovered and therefore no information is lost.

(e) The relational model is a representation of the data in terms of its natural structure only - it contains absolutely no consideration of storage/access details (pointers, physical ordering, indexing or similar access techniques); in a word, no 'representation clutter'.

2.2.2 The First Normal Form

Conversion to First Normal Form

Consider the relation P, representating PARTS, shown in Fig. 2.1 .

P (P#,	PD,	QOH,	J(J#,	JD,	JM#,	QC))
102	CAD	24	11	SORTER	005	4
			32	COLLATOR	079	6
105	TOG	144	11	SORTER	005	25
			24	PUNCH	079	14
			57	READER	023	9

Fig. 2.1 An Unnormalized Relation

For each distinct kind of part the part number (P#), part description (PD) and quantity on hand (QOH) are recorded together with the project data for each project J using that kind of part. The project data, which is a repeating group, consists of project number (J#), project description (JD), project manager number (JM#) and quantity (QC) of this part committed to this project. The attributes P#, PD and QOH all have simple values while the attribute J has relations as its values. The relation P is said to be unnormalized as J is a non-simple attribute of that relation.

The first step of normalization is to split P into two separate relations:

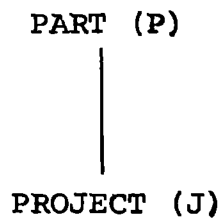
P_1 (P#, PD, QOH)

PJ_1 (P#, J#, JD, JM#, QC) .

These relations are shown in Figure 2.2 . This conversion

to first normal form is only possible if the unnormalized relation satisfies the following conditions:

(i) The graph of interrelationships of the non-simple attributes is a collection of trees, as shown below.



(ii) No primary key has a component attribute which is non-simple.

Definition of First Normal Form

A relation is in *first normal form* if it has the property that none of its attributes has elements which are themselves sets.

P_1	(P#,	PD,	QOH)
	102	CAD	24
	105	TOG	144

PJ_1	(P#,	J#,	JD,	JM#,	QC)
	102	11	SORTER	005	4
	102	32	COLLATOR	079	6
	105	11	SORTER	005	25
	105	24	PUNCH	079	14
	105	57	READER	023	9

Fig. 2.2 Relations in First Normal Form

2.2.3 The Second Normal Form

Conversion to Second Normal Form

In relation PJ_1 there are some irregularities in the dependence of attributes upon the primary key (P#, J#). Observation shows that JD and JM# are attributes of the J# component of the primary key while QC is an attribute of the entire key. These irregularities give rise to the following anomalies:

(i) Unless fictitious part numbers are introduced, data concerning a new project cannot be recorded until the project uses some parts (an insertion anomaly).

(ii) If only one kind of part remains in use by a project, deletion of data concerning that part causes deletion of the last remaining information on that project while previous deletions did not have this consequence (a deletion anomaly).

(iii) If a change is made to the value of an attribute of a project (e.g., the manager's serial number JM#), the number of copies of this information to be updated in the data model depends on the number of parts in use by that project at the instant the update is performed (an update anomaly).

This dependence of attributes upon the primary key (P#, J#) in PART-PROJECT relation PJ_1 can be removed by replacing relation PJ_1 by two of its projections: (f)

$$PJ_2 = \Pi_{\underline{P\#,\ J\#},\ QC}\ (PJ_1)$$
$$J_2 = \Pi_{\underline{J\#},\ JD,\ JM\#}\ (PJ_1)\ .$$

(f) For definition, see Appendix B.

The quantity QC of a part committed to a project is an attribute of the combination of part number P# and project number J#. Hence, it belongs in the relation PJ_2, not in J_2. The resulting relations are tabulated in Figure 2.3 .

Functional Dependence

Consider a relation R with attributes A and B. Attribute B of relation R is *functionally dependent* on attribute A of R if, at every instant of time, each value in A has only one value in B associated with it under R. If B is functionally dependent on A in R, then $R.A \rightarrow R.B$ else $R.A \not\rightarrow R.B$. If both $R.A \rightarrow R.B$ and $R.B \rightarrow R.A$ hold, then at all times R.A and R.B are in one-to-one correspondence and $R.A \leftrightarrow R.B$.

This definition can be extended for A and B to be two distinct collections of attributes. If B is not functionally dependent on any proper subset of A, then B is said to be *fully dependent* on A in R.

A functional dependence of the form $R.A \rightarrow R.B$ where B is a subset of A is called a *trivial dependence*.

Candidate Keys

Each *candidate key* K of relation R is a combination of attributes (or a single attribute) of R with the following properties:

P1 : (Unique Identification) In each tuple of R the value of K uniquely identifies that tuple, i.e., $R.K \rightarrow R.\Omega$ where Ω denotes the collection of all attributes of the specified relation.

P2 : (Non-redundancy) Property P1 is lost when any attribute of K is discarded, i.e., the number of attributes in K must be minimal.

Unless all possible values of the attributes of a relation R are known, property P2 cannot be imposed. Suppose K is defined to contain attributes D_1, D_2 and D_3. Imposing property P2, given the current values that D_1, D_2 and D_3 may assume, the key is minimal only if K contains D_1 and D_2. However, at some later time, values may be entered for D_1 and D_2 which will destroy property P1 and therefore attribute D_3 is needed to uniquely identify each tuple in R.

For each relation R in a data base, one of its several candidate keys is arbitrarily designated as the primary key of R. The usual operational distinction between the primary key and other candidate keys (if any) is that no tuple is allowed to have an undefined value for any of the primary key components, whereas any other components may have an undefined value. This restriction is imposed because of the crucial role played by primary keys in retrieval algorithms.

Any attribute of relation R which participates in at least one candidate key of R is a *prime attribute* of R. All other attributes of R are called *non-prime*.

Definition of Second Normal Form

A relation is in *second normal form* if it is in first normal form and every non-prime attribute of R is fully dependent on each candidate key of R.

Suppose R is in first normal form and one or both of the following conditions hold:

C1 : R has no non-prime attributes.

C2 : Every candidate key of R consists of just a single attribute.

Relation R is then said to be in second normal form.

P_2 (P#, PD, QOH)

P#	PD	QOH
102	CAD	24
105	TOG	144

PJ_2 (P#, J#, QC)

P#	J#	QC
102	11	4
102	32	6
105	11	25
105	24	14
105	57	9

J_2 (J#, JD, JM#)

J#	JD	JM#
11	SORTER	005
32	COLLATOR	079
24	PUNCH	079
57	READER	023

Fig. 2.3 Relations in Second Normal Form

2.2.4 The Third Normal Form

Conversion to Third Normal Form

The final step of normalization is best illustrated by considering the EMPLOYEE relation E tabulated in Figure 2.4 .

Each employee, identified by a distinct employee

E (E#,	JC,	D#,	M#,	CT)
1	b	y	12	n
2	c	z	13	n
3	c	z	13	n
4	d	x	11	g
5	a	y	12	n
6	b	w	14	g
7	a	x	11	g
8	b	z	13	n
9	d	x	11	g

Fig. 2.4 A Relation Not in Third Normal Form

serial number (E#) with a jobcode (JC), is assigned a department number (D#). Each department has its own manager, whose serial number (M#) is recorded, and each department is involved in contract type (CT) work, either governmental (g) or non-governmental (n). The non-trivial functional dependencies in E are exhibited in Figure 2.5 .

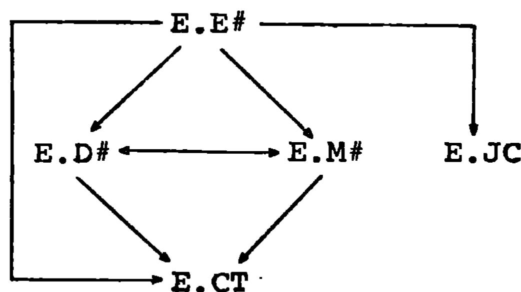


Fig. 2.5 Functional Dependencies in Relation E

Although relation E does not possess the kind of dependency described in Section 2.2.3, some of its attributes are transitively dependent on others and this gives rise to similar anomalies:

(i) Unless fictitious employee numbers are used, the D# and CT values for a new department cannot be established

before people are assigned to that department.

(ii) If only one employee remains attached to a department, deletion of that tuple for that employee causes deletion of the last remaining departmental information while previous deletions did not have this consequence.

(iii) If the manager of a department changes, more than one tuple has to be updated and the number of updates varies with time.

Conversion of relation E to third normal form consists of replacing E by two of its projections:

$$E_3 = \Pi_{\underline{E\#},\ JC,\ D\#}\ (E)$$
$$D_3 = \Pi_{\underline{D\#},\ M\#,\ CT}\ (E)\ .$$

These two relations are tabulated in Figure 2.7 .

Transitive Dependence

Suppose that A, B and C are three distinct collections of attributes of a relation R (R is of degree 3 or more) and that the following time-independent conditions hold:

$$R.A \rightarrow R.B, \qquad R.B \not\rightarrow R.A,$$
$$R.B \rightarrow R.C\ .$$

This implies that

$$R.A \rightarrow R.C \qquad R.C \not\rightarrow R.A\ .$$

Figure 2.6 summarizes the entire set of conditions on A, B and C. The condition $R.C \rightarrow R.B$ is neither prohibited nor required. Thus C is said to be *transitively dependent* on A under R.

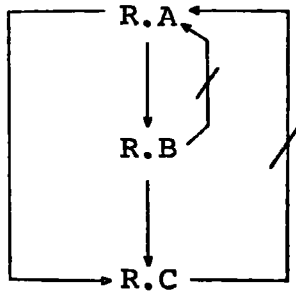


Fig. 2.6 Transitive Dependence of C on A under R

In the special case where R.C → R.B in addition, both B and C are transitively dependent on A under R.

Looking at Figure 2.5 it can be deduced that CT, D# and M# are transitively dependent on E# under E.

Definition of Third Normal Form

A relation R is in *third normal form* if it is in second normal form and every non-prime attribute of R is non-transitively dependent on each candidate key of R. Relations P_2, PJ_2 and J_2 tabulated in Figure 2.3 are also in third normal form.

The undesirable insertion, deletion and update dependencies have disappeared with the removal of the transitive dependencies. No information has been lost since the original relation E may be recovered by taking the natural join [g] of E_3 and D_3 on D#.

(g) For definition, see Appendix B.

E_3 (E#, JC, D#)

E#	JC	D#
1	b	y
2	c	z
3	c	z
4	d	x
5	a	y
6	b	w
7	a	x
8	b	z
9	d	x

D_3 (D#, M#, CT)

D#	M#	CT
y	12	n
z	13	n
x	11	g
w	14	g

Fig. 2.7 Relations in Third Normal Form

2.2.5 Objectives of Normalization

The objectives of normalization are:

(i) to make it feasible to tabulate any relation in the data base so that all objects are simple

(ii) to obtain a powerful retrieval capability by means of a simpler collection of relational operations than would otherwise be necessary

(iii) to avoid undesirable insertion, update and deletion dependencies in the relations

(iv) to make the relational model more informative to users

(v) to reduce the need for restructuring the collection of relations as new types of data are introduced and thus increase the life span of application programs

(vi) to make the collection of relations neutral to usage pattern especially where this pattern varies with time.

The first two objectives apply only to the first step of normalization while the last four objectives apply to all normalization steps.

2.2.6 Summary of Normalization

Figure 2.8 summarizes the relationship between the three normal forms.

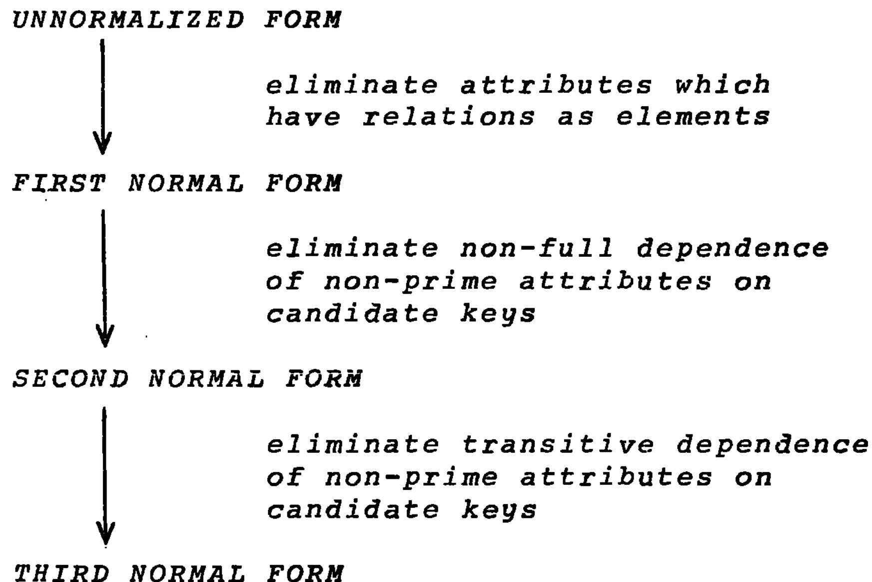


Fig. 2.8 The Three Normal Forms

2.3 RELATIONAL LANGUAGES

2.3.1 Introduction

A relational model of data, based on normalized relations of assorted degrees, can be manipulated by two principal types of relational language - the intermediate level, algebraic (typified by the Peterlee

IS/1 System)[8] and the high level, calculus-oriented data sublanguages, an example of which is described in Section 3.3 .

2.3.2 A Relational Algebra

In this algebraic approach data retrieval or selection is viewed as the formation of a new normalized relation from the existing collection of relations by some operation of the algebra. The basic operations of the relational algebra include the traditional set operations (Cartesian product, union, intersection, difference) and the new operations on these relations (projection, join, division, restriction). Definitions of these relational operations are given in Appendix B. These operations act upon entire normalized relations as their operands.

The collection of relations in a data base with these operations forms the relational algebra. These operations have been implemented on a computer system using APL.[9]

2.3.3 Relational Calculus

The relational calculus, based on an applied predicate calculus, may also be used in the formulation of queries of arbitrary complexity on any data base consis ting of a finite collection of relations in simple normal form. A query, represented as a relation-defining expression (an α-expression) in the relational calculus, is translated into a retrieval algorithm,[10] a sequence of operations

on the relations of the data base yielding the response relation. To facilitate the specification of such a query, Codd has also introduced a data sublanguage, DSL ALPHA, based on the relational calculus. An outline of this relational calculus which includes the definition of an α-expression is given in Appendix B.

The basis of this retrieval algorithm is essentially the reduction of an arbitrary α-expression into a semantically equivalent expression of the relational algebra. Palmero [3] has incorporated a number of improvements to this retrieval algorithm to obtain a more efficient algorithm. The introduction of the concept of semi-join as an intermediate object when the tuples of a relation are retrieved permits the algorithm to determine dynamically the order in which the relations are explored. Palmero used instead the γ-expression (a normalized β-expression) to represent the query.

2.3.4 Relational Completeness

An algebra or calculus is *relationally complete* if, given any finite collection of relations $R_1, R_2, \ldots, R_N$ in simple normal form, the expressions of the algebra or calculus permit definition of any relation definable from $R_1, R_2, \ldots, R_N$ by α-expressions (using a set of N range predicates in one-to-one correspondence with $R_1, R_2, \ldots, R_N$). Both the algebra and calculus described here provide a foundation for designing relationally complete query

languages.[h]

2.3.5 Calculus versus Algebra

Codd has shown that these two types of language (the algebraic and the calculus-oriented) for data base manipulation are equivalent in the sense that a relation-defining expression in the relational calculus can be mapped into a semantically equivalent expression in the algebra. Although the relational algebra possesses as much selective power as the relational calculus, the descriptive calculus approach as opposed to the constructive algebraic approach permits the user to request the desired data by its properties.

(h) For definition, see Section 3.1 .

3. THE PROPOSED DATA BASE SYSTEM WITH REFERENCE TO ALPHA SUBLANGUAGE

3.1 INTRODUCTION

Among languages for data base manipulation, the intermediate algebraic and the high level calculus-oriented sublanguages score heavily in protecting users from representation clutter by using a simple data model and by treating relations as data objects manipulable in their entirety.

A data sublanguage may be embedded (with appropriate syntactic modification) in a variety of host programming languages such as PL/1, COBOL and FORTRAN. All computation of functions is defined in the host language statements; all retrieval and storage operations in data sublangauge statements. Arithmetic functions defined in the host language can be invoked in the data sublanguage statements. Alternatively, the data sublanguage may stand alone (self-contained system) and is commonly referred to as a query language, even though it may contain provision for simple updating, inserting, etc.

Many users in the past were forced to use specialized query languages to meet their specific needs. The high cost of supporting a great variety of these languages with translators has suggested that the common functions in these translators be identified and programmed once and for

all. This, in turn, has suggested adopting as a source language for these translators a very high storage and retrieval language. The calculus-oriented type of language, an example of which is DSL ALPHA, appears best suited for this purpose. The adoption of this calculus-oriented approach has permitted successive improvements in general retrieval algorithms to be incorporated into data base systems without affecting users' programs.

A query may be represented in a number of forms as shown in Figure 3.1 .

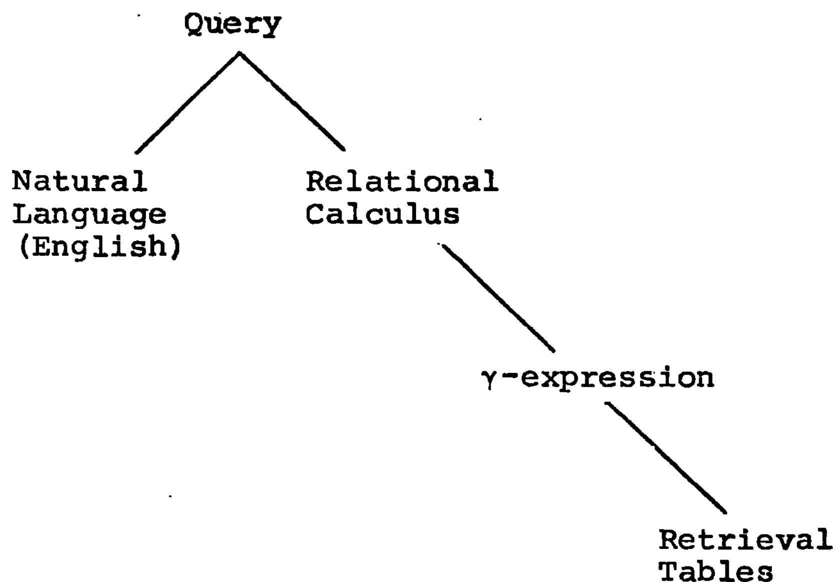


Fig. 3.1 Query Representation

A tabular representation of the query, in the form of retrieval tables, [a] is also derived from the

(a) Retrieval tables are those coding tables 'generated' for a query (see Section 5).

γ-expression [b] which is needed for the understanding of Palmero's retrieval algorithm. A detailed description of this retrieval algorithm will not be reiterated here as a paper on the algorithm is available. [3] Briefly, from the 'generated' retrieval tables, the required relations are retrieved from the data base and the semi-joins and reference relations are constructed and stored in temporary variables. These semi-joins are then combined to construct the relation T_{p+q} where p and q are the number of free variables and quantified variables respectively in the γ-expression. The quantifier operations (projection and division) are then applied to obtain the indirect response relation T which is then combined with the reference relations to construct the response relation to the query.

Palmero's retrieval algorithm will be used as a basis for the development of algorithms (e.g., for deleting, inserting, etc.) to be incorporated into data base systems.

3.2 THE PROPOSED DATA BASE SYSTEM

A schematic representation of the proposed data base system is presented in Figure 3.2 . The relational completeness of DSL ALPHA (i.e., any query expressible in the predicate calculus is expressible in ALPHA) makes

---

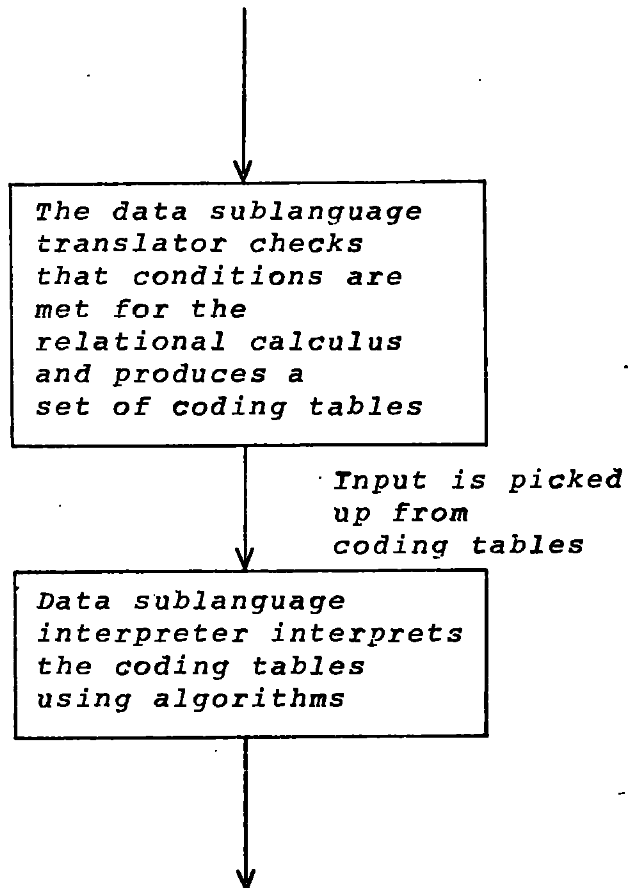
(b) It is conjectured that a γ-expression is semantically equivalent to an α-expression although there are a number of syntactic differences between them.

it an extremely suitable candidate for implementation. DSL ALPHA has been implemented as a query and command language, and an array representation of normalized relations in the data model has been assumed.

In the proposed data base system the translator accepts as input a modified version of source statements, checks that conditions are met for the relational calculus (interactive debugging facilities are provided) and produces as output a set of coding tables suitable for interpretation. A detailed description of the working of this translator is given in Section 4. The data sublanguage interpreter, which incorporates Palmero's retrieval algorithm and other algorithms, interprets the coding tables at interpretation time.

A query handling scheme of the proposed data base system is presented in Figure 3.3 . Palmero's retrieval algorithm analyzes queries in the form of retrieval tables produced by the translator. The actual retrieval of tuples from a relation (or relations) in the data base will be done through system subroutines to preserve physical data independence, i.e., the interpreter will not be aware of the physical representation of the data except for the fact that it will be able to know whether an attribute is a component of or is the primary key. These system subroutines are essentially the operations of the relational algebra defined in Appendix B. Since all the basic retrieval power is embodied in DSL ALPHA, this retrieval capability can be implemented once and for all.

DSL ALPHA statements
and
computational facility statements



Retrieval / Update / Deletion / Insertion /
Dropping or Establishing data base relation /
Input or Output facility

Fig. 3.2 The Proposed Data Base System

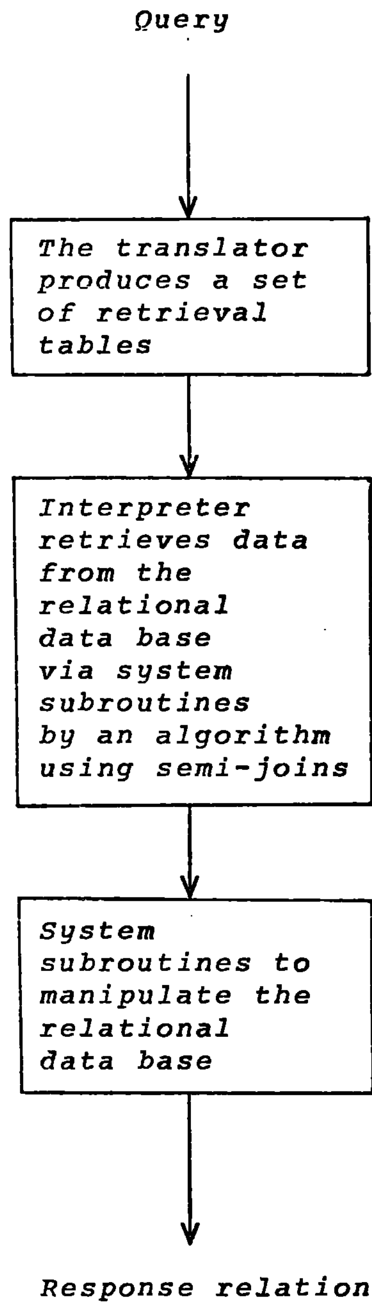


Fig. 3.3 A Query Handling Scheme of the Proposed System

Hence the proposed system provides a means of defining the physical representation of data independently of the translator and interpreter; and that changes to any of them will not affect the system subroutines and vice-versa.

3.3 DESCRIPTION OF THE IMPLEMENTED DATA SUBLANGUAGE ALPHA

3.3.1 Basic Characteristics

The ALPHA sublanguage is a language for manipulating relations. It permits the following functions:

(i) the addition of declared relations and their attributes to the system catalog; each declaration of a relation identifies the primary key for that relation

(ii) the specification for retrieval of any subset of data

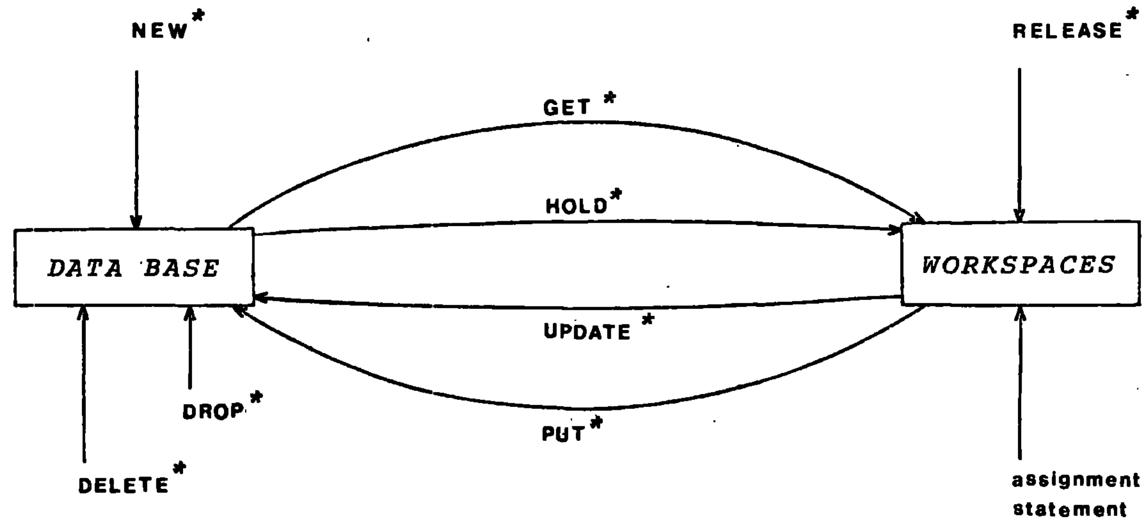
(iii) the addition of new elements or sets to declared relations

(iv) the deletion of elements or sets from declared relations

(v) the declaration (implicit) of variables; each variable being assigned a value

(vi) the listing of information contained in relations and values of variables.

Codd's concept of the user workspace in DSL ALPHA has to be explained. Data flows in and out of the data base via user workspace as directed by data sublanguage statements (see Figure 3.4). This data transmission appears to the user to take place all at once. A workspace may consist of space in core or disk. A single user has



* DSL ALPHA STATEMENTS

Fig. 3.4 <u>Information Flow</u>

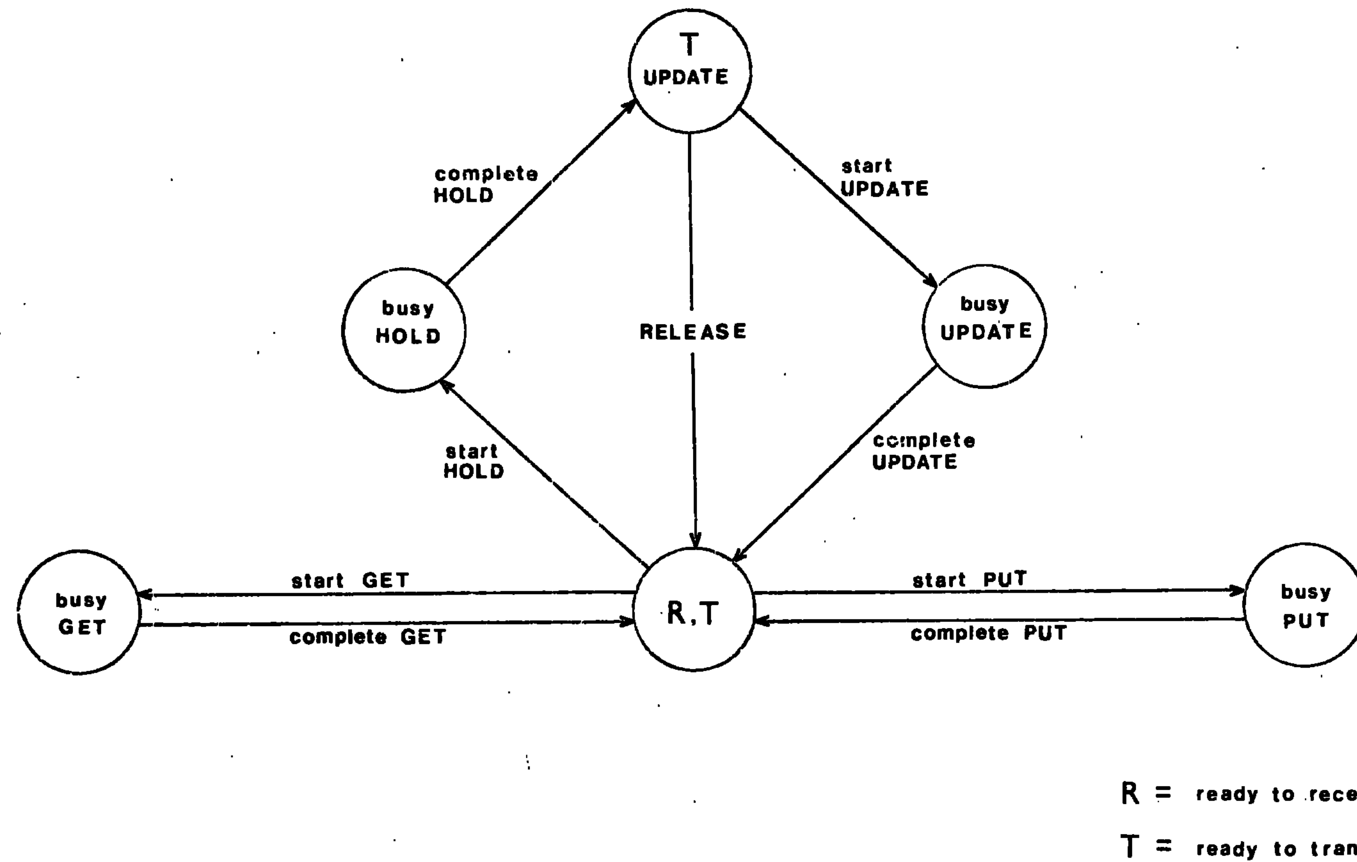


Fig. 3.5 <u>Transmission Control: State Transition Diagram</u>

several workspaces in use concurrently.

The workspace provides a data interface between DSL ALPHA statements and assignment statements.[c] The user never applies assignment statements directly to data residing in the data base, but instead fetches the desired data into one or more workspaces using data sublanguage statements, and then applies assignment statements to the data in these workspaces. Thus, at each instant of time, each workspace holds the array representation of one relation or a tuple.

A state diagram is given in Figure 3.5 showing permissible changes of state for any workspace. Whenever a workspace is in one of the four busy states it is inaccessible to both DSL ALPHA statements and assignment statements. In any of the non-busy states assignment statements may be applied to the workspace.

3.3.2 Structure of Data Sublanguage ALPHA

The structure of ALPHA sublanguage is determined by:

(i) N, the set of syntactic entities (or non-terminal symbols)

(ii) T, the set of basic (or terminal) symbols

(iii) S, the distinguished symbol

(iv) P, the set of syntactic rules (or productions).

(c) Assignment statements are incorporated in DSL ALPHA for updating purposes.

Notation

A syntactic entity is denoted by its name (a sequence consisting of letters and hyphens) enclosed in the brackets < and >. A syntactic rule has the form

```
<A> ::= X
```

where <a> is a member of N, and x is any possible sequence of basic symbols and syntactic entities. The set P contains the syntactic rule

```
<BAR> ::= |
```

implying that | is a basic symbol of the sublanguage. Adopting the convention that all references to this basic symbol in other syntactic rules shall be replaced by <bar> permits the unambiguous use subsequently of the notation

```
<A> ::= X
      | Y
        .
        .
        .
      | Z
```

as an abbreviation for the set of syntactic rules

```
<A> ::= X
<A> ::= Y
 .      .
 .      .
 .      .
<A> ::= Z
```

This particular notation is called Backus-Normal Form or Backus-Naur Form (BNF).

3.3.3 Basic Symbols

The ALPHA sublanguage is built up from the following basic symbols:

```
<BASIC-SYMBOL>  ::=  <LETTER>
                  |  <DIGIT>
                  |  <DELIMITER>
```

Letters

```
<LETTER>  ::=  A
            |  B
            |  C
            |  D
            |  E
            |  F
            |  G
            |  H
            |  I
            |  J
            |  K
            |  L
            |  M
            |  N
            |  O
            |  P
            |  Q
            |  R
            |  S
            |  T
            |  U
            |  V
            |  W
            |  X
            |  Y
            |  Z
```

Letters do not have individual meaning. They are used for forming identifiers and strings.

Digits

```
<DIGIT>  ::=  0
           |  1
           |  2
           |  3
           |  4
           |  5
           |  6
           |  7
           |  8
           |  9
```

Digits are used for forming identifiers, numbers and strings.

Delimiters

```
<DELIMITER>  ::=  <OPERATOR>
               |  <SEPARATOR>
               |  <BRACKET>
               |  <DECLARATOR>
               |  <QUANTIFIER>

<OPERATOR>  ::=  <ARITHMETIC-OPERATOR>
              |  <RELATIONAL-OPERATOR>
              |  <LOGICAL-OPERATOR>
              |  <VERBAL-OPERATOR>

<ARITHMETIC-OPERATOR>  ::=  +
                         |  -
                         |  /
                         |  *

<RELATIONAL-OPERATOR>  ::=  <
                         |  >
                         |  =
                         |  <=
                         |  >=
                         |  ¬=

<LOGICAL-OPERATOR>  ::=  &
                      |  |
                      |  AND
                      |  OR

<VERBAL-OPERATOR>  ::=  READ
                     |  LIST
                     |  RANGE
                     |  GET
                     |  HOLD
                     |  RELEASE
                     |  UPDATE
                     |  DELETE
                     |  DROP
                     |  PUT
                     |  NEW

<SEPARATOR>  ::=  :
               |  ,
               |  .
               |  =
               |  UP
               |  DOWN

<BRACKET>  ::=  (
             |  )
             |  $(
             |  $)

<DECLARATOR>  ::=  KEY
                |  FIXED31
                |  FIXED
                |  FLOAT16
                |  FLOAT
                |  CHAR
                |  CHARVAR

<QUANTIFIER>  ::=  ALL
                |  SOME
```

Separators serve the purpose of marking divisions between certain DSL ALPHA entities, while declarators and quantifiers are used to describe the properties of identifiers.

3.3.4 Identifiers

Syntax

```
<IDENTIFIER>  ::=  <LETTER>
                |  <IDENTIFIER> <LETTER>
                |  <IDENTIFIER> <DIGIT>
                |  <IDENTIFIER> _
                |  <IDENTIFIER> #

<WORKSPACE-NAME>  ::=  <IDENTIFIER>

<DATA-BASE-RELATION-NAME>  ::=  <IDENTIFIER>

<LOCAL-RELATION-NAME>  ::=  <IDENTIFIER>

<ATTR-NAME>  ::=  <IDENTIFIER>

<VARIABLE>  ::=  <IDENTIFIER>

<IDENTIFIER-LIST>  ::=  <IDENTIFIER>
                     |  <IDENTIFIER-LIST> , <IDENTIFIER>

<VARIABLE-LIST>  ::=  <VARIABLE>
                   |  <VARIABLE-LIST> , <VARIABLE>
```

Examples

SUPPLY

SUB_PART

W1

PART#

Semantics

Identifiers serve for the identification of workspace names, data base relation names, local relation names, attribute names and variables. The length of an identifier should contain twenty or fewer alphanumeric, hash and underscore characters. Certain identifiers are reserved words (see Section 3.3.5).

Names for data base relations and attributes are chosen by the community of users, while workspace names and local relation names are chosen by individual users. A workspace name identifies a DSL ALPHA workspace while

a data base relation name identifies a data base relation. A local relation name designates a typical tuple of the relation to which it is specified. It is also used to provide an alias for a data base relation name/workspace name used in more than one context in a single statement. An attribute name identifies an attribute. Users must not use any data base relation name or attribute name as a workspace name or local relation name.

Identifiers which represent variables are of type real or integer. Variables are implicitly declared. An integer variable has as its first character I, J, K, L, M or N while a real variable starts with one of the characters in the range A to H or O to Z.

3.3.5 Standard Function Identifiers

Syntax

```
<FUNCTION-IDENTIFIER>  ::=  AVERAGE
                          | COUNT
                          | MAX
                          | MIN
                          | TOTAL

<I-FUNCTION-IDENTIFIER>  ::=  IAVERAGE
                            | ICOUNT
                            | IMAX
                            | IMIN
                            | ITOTAL

<BOOL-FUNCTION-IDENTIFIER>  ::=  TOP
                               | BOTTOM
```

Semantics

The following identifiers are predeclared for the standard functions of analysis:

(i) COUNT, ICOUNT - counts the number of elements

(ii) TOTAL, ITOTAL - forms the sum of elements

(iii) MAX, IMAX - selects the maximum value

(iv) MIN, IMIN - selects the minimum value

(v) AVERAGE, IAVERAGE - forms the arithmetic mean

(vi) TOP, BOTTOM - returns the value <u>true</u> or <u>false</u>.

The standard functions COUNT and ICOUNT are applied to any finite set while the others are applicable to finite sets of numbers only. Examples to illustrate the use of some of these functions are given in Section 3.4 .

3.3.6 Constants

Syntax

```
<CONSTANT>  ::=  <NUMBER>
              |  <STRING-CONSTANT>

<NUMBER>  ::=  <SIGN> <INTEGER-CONSTANT>
            |  <SIGN> <REAL-CONSTANT>
            |  <INTEGER-CONSTANT>
            |  <REAL-CONSTANT>

<SIGN>  ::=  +
          |  -

<INTEGER-CONSTANT>  ::=  <DIGIT>
                      |  <INTEGER-CONSTANT> <DIGIT>

<REAL-CONSTANT>  ::=  . <INTEGER-CONSTANT>
                   |  <INTEGER-CONSTANT> .
                   |  <INTEGER-CONSTANT> . <INTEGER-CONSTANT>

<STRING-CONSTANT>  ::=  ' <OPEN-STRING> '

<OPEN-STRING>  ::=  <CHARACTER>
                 |  <OPEN-STRING> <CHARACTER>

<CHARACTER>  ::=  <LETTER>
               |  <DIGIT>
               |  <SPECIAL-CHARACTER>

<SPECIAL-CHARACTER>  ::=  BLANK
                       |  .
                       |  <
                       |  (
                       |  +
                       |  |
                       |  &
                       |  !
                       |  $
                       |  *
                       |  )
                       |  ;
                       |  ¬
                       |  -
                       |  /
                       |  ,
                       |  %
                       |  _
                       |  >
                       |  :
                       |  #
                       |  @
                       |  '
                       |  =
                       |  "
```

Examples

```
 25
+20.7
-400.579
'26.12.48'
'JONES'
```

Semantics

Arithmetic constants are numbers interpreted according to the conventional decimal notation. Each number has a uniquely defined type.

Strings consist of any sequence of at most 20 characters enclosed by ', the string quote.

3.3.7 Statements

The distinguished symbol S of the ALPHA sublanguage is the symbol <session>.

Syntax

```
<SESSION>  ::=  <STATEMENT-LIST> STOP

<STATEMENT-LIST>  ::=  <STATEMENT>
                    |  <STATEMENT-LIST> <STATEMENT>

<STATEMENT>  ::=  <COMP-FACILITY-STATEMENT>
               |  <QUERY-SEQUENCE-STATEMENT>
               |  <UPDATE-SEQUENCE-STATEMENT>
               |  <DELETE-SEQUENCE-STATEMENT>
               |  <DROP-STATEMENT>
               |  <PUT-STATEMENT>
               |  <NEW-STATEMENT>
```

The principal statement forms of the sublanguage are the computational facility statement, query sequence statement, update sequence statement, delete sequence statement, drop statement, put statement and new statement. A principal statement denotes a unit of action. By the

execution of a principal statement is meant the performance of this unit of action, which may consist of smaller units of action such as the evaluation of expressions (e.g., qualification expression) or the execution of other statements.

(i) Computational Facility Statements

Syntax

```
<COMP-FACILITY-STATEMENT>  ::=  <ASSIGNMENT-STATEMENT>
                             |  <INPUT-OUTPUT-STATEMENT>

<ASSIGNMENT-STATEMENT>  ::=  <WORKSPACE-NAME> . <ATTR-NAME> = <RIGHT-PART>

<RIGHT-PART>  ::=  ' <OPEN-STRING> '
                |  <ARITHMETIC-EXPRESSION>

<ARITHMETIC-EXPRESSION>  ::=  <SECONDARY>
                           |  <SIGN> <SECONDARY>
                           |  <ARITHMETIC-EXPRESSION> + <SECONDARY>
                           |  <ARITHMETIC-EXPRESSION> - <SECONDARY>

<SECONDARY>  ::=  <PRIMARY>
               |  <SECONDARY> * <PRIMARY>
               |  <SECONDARY> / <PRIMARY>

<PRIMARY>  ::=  <NUMBER>
             |  <VARIABLE>
             |  <WORKSPACE-NAME> . <ATTR-NAME>
             |  ( <ARITHMETIC-EXPRESSION> )

<INPUT-OUTPUT-STATEMENT>  ::=  <READ-STATEMENT>
                            |  <LIST-STATEMENT>

<READ-STATEMENT>  ::=  READ ( <VARIABLE-LIST> )

<LIST-STATEMENT>  ::=  LIST ( <IDENTIFIER-LIST> )
```

Examples

```
W.Q = W.Q * (30.867 - 3.469)
READ (ALPHA, BETA)
LIST (W, ALPHA, X, M)
```

Semantics

Assignment statements are applied only to data in the workspaces to alter the current values.

A read statement designates a free field input procedure. Identifiers specified in the read

statement are declared to be variables. Values are read, matched with the variables of the actual variable list in order of appearance, and assigned to the corresponding variables. The type of each value must be assignment compatible with the type of the corresponding variable. A list statement designates an output procedure with automatic format conversion. Information contained in data base relations and workspaces, and values of variables can be requested.

Precedence of Operators

The sequence of operations within an arithmetic expression is generally from left to right, with the following additional rules:

(i) According to the syntax given above the following rules of precedence hold:

first :	/	*
second :	+	-

(ii) The expression between a left parenthesis and the matching right parenthesis is evaluated by itself and this value is used in subsequent calculations. Consequently the desired order of operations within an arithmetic expression can be specified by appropriate positioning of parentheses.

(ii) Query Sequence Statements

Syntax

```
<QUERY-SEQUENCE-STATEMENT>  ::=  <RANGE-LIST> <GET-STATEMENT>

<RANGE-LIST>  ::=  <RANGE-STATEMENT>
               |   <RANGE-LIST> <RANGE-STATEMENT>

<RANGE-STATEMENT>  ::=  <RANGE-NAME-PART>
                    |   <RANGE-NAME-PART> <QUANTIFIER>

<RANGE-NAME-PART>  ::=  RANGE <RELATION-NAME> <LOCAL-RELATION-NAME>

<RELATION-NAME>  ::=  <WORKSPACE-NAME>
                  |   <DATA-BASE-RELATION-NAME>

<GET-STATEMENT>  ::=  <GET-NAME-PART>
                  |   <GET-NAME-PART> : <QUALIFICATION-EXPRESSION>
                  |   <GET-NAME-PART> : <QUALIFICATION-EXPRESSION> <GET-ELEMENT-ORDERING-EXPRESSION>
                  |   <GET-NAME-PART> <GET-ELEMENT-ORDERING-EXPRESSION>

<GET-NAME-PART>  ::=  GET <WORKSPACE-NAME> <QUOTA> <GET-TARGET-LIST>
                  |   GET <WORKSPACE-NAME> <GET-TARGET-LIST>

<QUOTA>  ::=  ( <IDENTIFIER> )
          |   ( <INTEGER-CONSTANT> )

<GET-TARGET-LIST>  ::=  <GET-TARGET>
                    |   <FUNCTION-DESIGNATOR>
                    |   <QUAL-ATTR-LIST> , <I-FUNCTION-DESIGNATOR>
                    |   ( <GET-TARGET-LIST> )

<GET-TARGET>  ::=  <GET-TARGET-ELEMENT>
               |   <GET-TARGET> , <GET-TARGET-ELEMENT>

<GET-TARGET-ELEMENT>  ::=  <LOCAL-RELATION-NAME>
                       |   <QUAL-ATTR-NAME>

<QUAL-ATTR-NAME>  ::=  <LOCAL-RELATION-NAME> . <ATTR-NAME>

<FUNCTION-DESIGNATOR>  ::=  <FUNCTION-IDENTIFIER> ( <FUNCTION-ARGUMENT> )

<FUNCTION-ARGUMENT>  ::=  <QUAL-ATTR-NAME>

<QUAL-ATTR-LIST>  ::=  <QUAL-ATTR-NAME>
                   |   <QUAL-ATTR-LIST> , <QUAL-ATTR-NAME>

<I-FUNCTION-DESIGNATOR>  ::=  <I-FUNCTION-IDENTIFIER> ( <I-FUNCTION-ARGUMENT> )

<I-FUNCTION-ARGUMENT>  ::=  <LOCAL-RELATION-NAME> , <ATTR-NAME-SEQUENCE> , <ATTR-NAME>

<ATTR-NAME-SEQUENCE>  ::=  <ATTR-NAME>
                       |   ( <ATTR-LIST> )

<ATTR-LIST>  ::=  <ATTR-NAME>
              |   <ATTR-LIST> , <ATTR-NAME>

<QUALIFICATION-EXPRESSION>  ::=  <COMPLEX-QUALIFICATION>
                             |   <BOOL-FUNCTION-DESIGNATOR>
                             |   <I-FUNCTION-DESIGNATOR> <RELATIONAL-OPERATOR> <NUMBER>

<COMPLEX-QUALIFICATION>  ::=  <QUALIFICATION>
                          |   <COMPLEX-QUALIFICATION> OR <QUALIFICATION>

<QUALIFICATION>  ::=  <QUAL>
                  |   <QUALIFICATION> AND <QUAL>

<QUAL>  ::=  <W-COMPONENT>
         |   $( <COMPLEX-QUALIFICATION> $)

<W-COMPONENT>  ::=  <MATRIX-THETA>
                |   <W-COMPONENT> | <MATRIX-THETA>

<MATRIX-THETA>  ::=  <THETA-COMPONENT>
                 |   ( <MATRIX-THETA> )

<THETA-COMPONENT>  ::=  <TERM>
                    |   <THETA-COMPONENT> & <TERM>

<TERM>  ::=  ( <MONADIC-TERM> )
         |   ( <DYADIC-JOIN-TERM> )
```

```
<MONADIC-TERM>  ::=  <QUAL-ATTR-NAME> <RELATIONAL-OPERATOR> <CONSTANT>

<DYADIC-JOIN-TERM>  ::=  <QUAL-ATTR-NAME> <RELATIONAL-OPERATOR> <QUAL-ATTR-NAME>

<BOOL-FUNCTION-DESIGNATOR>  ::=  <BOOL-FUNCTION-IDENTIFIER> ( <BOOL-FUNCTION-ARGUMENT> )

<BOOL-FUNCTION-ARGUMENT>  ::=  <INTEGER-CONSTANT> , <QUAL-ATTR-NAME>

<GET-ELEMENT-ORDERING-EXPRESSION>  ::=  <GET-ORDER>
                                     |  <GET-ELEMENT-ORDERING-EXPRESSION> <GET-ORDER>

<GET-ORDER>  ::=  UP <QUAL-ATTR-NAME>
              |   DOWN <QUAL-ATTR-NAME>
```

Examples

```
RANGE  PART  P
GET  W  (P.P#, P.PNAME, P.QOH) : (P.QOH < 25)  UP P.WEIGHT

RANGE  SUPPLY  Z
GET  W  COUNT(Z.S#) : (Z.J# = 5)
```

Semantics

A query sequence statement constructs the response relation in a specified workspace.

The range list informs the system that certain local relation names are used to designate typical tuples of the corresponding relations. This range list stays in effect until the response relation to the query is constructed.

The target list in a get statement defines the relation to be constructed. This list is separated from the qualification expression (if present) by a colon which can be taken to mean 'such that'. Any number of distinct relations may be referenced in the target list. When a function designator or an image function designator appears in the target list then only one relation is referenced. The quota, if specified, indicates to the system the number of tuples to be retrieved. The

qualification expression (if present) provides the selection criteria for tuples of the response relation. The element ordering expression can be specified to inform the system that tuples are to be delivered to the workspace in a particular ordering.

All tuple variables (i.e., local relation names) appearing in the qualification expression and not in the target list of a get statement must be quantified explicitly using the attributes SOME or ALL on range statements. The quantifier SOME indicates existential quantification of the associated local relation name while ALL indicates universal quantification. Tuple variables appearing in the target list should not be quantified.

Precedence of Operators

According to the syntax given above the following rules of precedence hold:

first :	< > = <= >= ¬=
second :	&
third :		
fourth :	AND
fifth :	OR

Precedence can be imposed upon the logical operators, AND and OR, separating the w components in the qualification expression. The w component between a left bracket $(and the matching right bracket $) is evaluated and this result is used in subsequent calculations. Hence the desired order of execution of

a logical combination of two or more w components in the qualification expression can be specified by appropriate positioning of these brackets.

(iii) Update Sequence Statements

Syntax

```
<UPDATE-SEQUENCE-STATEMENT>  ::=  <UPDATE-HEAD> <RELEASE-STATEMENT>
                               |  <UPDATE-HEAD> <UPDATE-STATEMENT>
                               |  <UPDATE-BODY> <RELEASE-STATEMENT>
                               |  <UPDATE-BODY> <UPDATE-STATEMENT>

<UPDATE-HEAD>  ::=  <RANGE-LIST> <HOLD-STATEMENT>

<HOLD-STATEMENT>  ::=  <HOLD-NAME-PART>
                    |  <HOLD-NAME-PART> : <QUALIFICATION-EXPRESSION>
                    |  <HOLD-NAME-PART> : <QUALIFICATION-EXPRESSION> <GET-ELEMENT-ORDERING-EXPRESSION>
                    |  <HOLD-NAME-PART> <GET-ELEMENT-ORDERING-EXPRESSION>

<HOLD-NAME-PART>  ::=  HOLD <WORKSPACE-NAME> <HOLD-TARGET-LIST>

<HOLD-TARGET-LIST>  ::=  <HOLD-TARGET>
                      |  <LOCAL-RELATION-NAME>
                      |  ( <HOLD-TARGET-LIST> )

<HOLD-TARGET>  ::=  <HOLD-TARGET-ELEMENT>
                 |  <HOLD-TARGET> , <HOLD-TARGET-ELEMENT>

<HOLD-TARGET-ELEMENT>  ::=  <QUAL-ATTR-NAME>

<RELEASE-STATEMENT>  ::=  RELEASE

<UPDATE-STATEMENT>  ::=  UPDATE

<UPDATE-BODY>  ::=  <UPDATE-HEAD> <COMP-FACILITY-LIST>

<COMP-FACILITY-LIST>  ::=  <COMP-FACILITY-STATEMENT>
                        |  <COMP-FACILITY-LIST> <COMP-FACILITY-STATEMENT>
```

Examples

```
RANGE  PART  P
HOLD  W  (P.QOH) : (P.P# = 3)
W.QOH = W.QOH + 5
UPDATE

RANGE  SUPPLY  Z
HOLD  W  (Z.S#) : (Z.P# ¬= 3)
RELEASE
```

Semantics

A hold statement has the same effect as a corresponding get statement in regard to the information made available to a specified workspace. An additional effect of a hold statement is that it warns the system to be prepared to return modified data to the tuples supplying the retrieved data. This returning of modified data is requested by an update statement. The system suspends other non-computational facility accesses until this update is completed or cancelled. Cancellation is requested by a release statement.

The hold statement (unlike the get statement) is restricted to a single relation occurring in its target list. Thus multi-relation updates entail the use of more than one hold statement and more than one workspace.

(iv) Delete Sequence Statements

Syntax

```
<DELETE-SEQUENCE-STATEMENT>  ::=  <RANGE-LIST> <DELETE-STATEMENT>

<DELETE-STATEMENT>  ::=  <DELETE-NAME-PART>
                      |  <DELETE-NAME-PART> : <QUALIFICATION-EXPRESSION>

<DELETE-NAME-PART>  ::=  DELETE <LOCAL-RELATION-NAME>
```

Examples

```
RANGE   PART   P
DELETE   P : (P.P# = 5)

RANGE   SUPPLY   Z
DELETE   Z
```

Semantics

A delete sequence statement removes tuples from only one relation. The qualification expression in a delete statement is similar to that in the hold statement or get statement, i.e., it can involve any number of relations. When a boolean function designator or an image function designator appears in the qualification expression then only one relation is involved.

(v) Drop Statements

Syntax

```
<DROP-STATEMENT>   ::=  <DROP-NAME-PART>
                     |  <DROP-NAME-PART> . <ATTR-NAME-SEQUENCE>

<DROP-NAME-PART>   ::=  DROP <DATA-BASE-RELATION-NAME>
```

Examples

```
DROP   PROJECT
DROP   PART.(COLOR, WEIGHT)
```

Semantics

A drop statement removes all information about a data base relation or its attribute (or attributes) from the data base catalog.

The statement DROP R (where R is a relation name) should be contrasted with

```
RANGE   R   L
DELETE   L
```

(where L is a local relation name) which merely deletes all

tuples of R but leaves R as an established, although empty, relation of the data base.

(vi) Put Statements

Syntax

```
<PUT-STATEMENT>  ::=  <PUT-NAME-PART>
                   |  <PUT-NAME-PART> . <PUT-REST>
                   |  <PUT-NAME-PART> . <PUT-REST> <PUT-ELEMENT-ORDERING-EXPRESSION>
                   |  <PUT-NAME-PART> <PUT-ELEMENT-ORDERING-EXPRESSION>

<PUT-NAME-PART>  ::=  PUT <WORKSPACE-NAME> <RELATION-NAME>

<PUT-REST>  ::=  <ATTR-ORDERING-EXPRESSION>

<ATTR-ORDERING-EXPRESSION>  ::=  <ATTR-NAME-SEQUENCE>

<PUT-ELEMENT-ORDERING-EXPRESSION>  ::=  <PUT-ORDER>
                                     |  <PUT-ELEMENT-ORDERING-EXPRESSION> <PUT-ORDER>

<PUT-ORDER>  ::=  UP <ATTR-NAME>
               |  DOWN <ATTR-NAME>
```

Examples

```
PUT  W1  W2
PUT  W  PART.(P#, COLOR, QOH)  UP P#
```

Semantics

A put statement inserts tuples located in a workspace into a relation in the data base, or into another workspace.

In all put statements only one data base relation is specified. If the tuples to be inserted are ordered, the element ordering in that workspace can be specified to advise the system.

(vii) New Statements

Syntax

```
<NEW-STATEMENT>  ::=  NEW <DATA-BASE-RELATION-NAME> <ATTR-RECORD>

<ATTR-RECORD>  ::=  ( <ATTR-FIELD> )
                 |  <ATTR-RECORD> ( <ATTR-FIELD> )

<ATTR-FIELD>  ::=  <ATTR-NAME> , <ATTR-TYPE-LENGTH>
               |   <ATTR-NAME> , <KEY-TYPE> , <ATTR-TYPE-LENGTH>

<ATTR-TYPE-LENGTH>  ::=  <ATTR-TYPE>
                     |   <ATTR-TYPE> , <ATTR-LENGTH>

<ATTR-TYPE>  ::=  FIXED31
              |   FIXED
              |   FLOAT16
              |   FLOAT
              |   CHAR
              |   CHARVAR

<ATTR-LENGTH>  ::=  <INTEGER-CONSTANT>

<KEY-TYPE>  ::=  KEY
```

Examples

```
NEW  PROJECT (J#, KEY, CHAR, 4)(JNAME, CHAR, 10)
NEW  SUPPLIER (S#, KEY, FIXED)(SNAME, CHAR, 15)
```

Semantics

A new statement declares a new data base relation. Each declaration of a data base relation identifies its attributes, primary key as well as the data types of these attributes.

The keywords FIXED and FLOAT denote attributes to be of type integer and real respectively, whereas FIXED31 and FLOAT16 are the double precision representation. CHAR and CHARVAR denote fixed and varying length character strings respectively.

3.4 DATA SUBLANGUAGE ALPHA EXAMPLES

Examples to illustrate the use of the implemented DSL ALPHA are given in the following order:

(i) queries

(ii) updates

(iii) deletions

(iv) insertions

(v) dropping and establishing data base relations

(vi) input and output facilities.

The sample data base which the examples exploit, [d] consists of four relations as tabulated in Table 3.6 .

Relation	Attributes
FACTORY	S#, SNAME, ADDRESS
MATERIAL	P#, PNAME, WEIGHT, QOH
TASK	J#, JNAME, PRIORITY
ORDER	S#, P#, J#, DATEDUE, QUANTITY

Table 3.6 Sample Data Base

The relation ORDER satisfies the following conditions:

(i) The set of supplier numbers (S#) appearing in the ORDER relation is a subset of supplier numbers appearing in the FACTORY relation.

(ii) A similar constraint applies to part numbers (P#) relative to the MATERIAL relation.

(iii) A similar constraint applies to job numbers (J#) relative to the TASK relation.

(d) Exploiting a data base includes queries, updates, etc.

3.4.1 Queries

Simple Query

** Find all the part numbers of materials being supplied by the factories.

```
RANGE  MATERIAL  M
GET  W  (M.P#)
```

The set of distinct part numbers appearing in the part number attribute of data base relation MATERIAL is copied into workspace W. Duplicate values of P# are not delivered to W. After execution of this query sequence statement, W may be treated as a unary relation whose sole attribute is P#.

Query with Qualification

** Find the part numbers, names and quantities on hand (QOH) where the quantity on hand is less than 100.

```
RANGE  MATERIAL  M
GET  W  (M.P#, M.PNAME, M.QOH) : (M.QOH < 100)
```

The set of distinct MATERIAL tuples (P#, PNAME, QOH) satisfying the requirement that the quantity on hand is less than 100 is copied into workspace W. W may then be regarded as a ternary relation with attributes P#, PNAME and QOH.

A convention called the *name inheritance rule* permits the attributes to be referred to by the terms

W.P#, W.PNAME, W.QOH

Query with Quota and Element Ordering Expression

** Same query as above, except that no more than six elements are to be retrieved and elements are to be delivered to the workspace in order by increasing P# (major order) and decreasing WEIGHT (minor order).

```
RANGE  MATERIAL  M
GET  W (6) (M.P#, M.PNAME, M.QOH) : (M.QOH < 100)
                                      UP M.P# DOWN M.WEIGHT
```

If no element ordering expression is specified and the data base contains more elements satisfying the specified qualification than the quota, the system determines which of the qualified elements are to be delivered. By including an element ordering expression along with a quota, the user can exert partial (and in some cases, complete) control over which elements are delivered.

Single Existential Quantifier

** Find the names and addresses of those factories whose orders have been placed for part number 5.

```
RANGE  FACTORY  F
RANGE  ORDER  O
GET  W  (F.SNAME, F.ADDRESS) : ∃O((F.S# = O.S#)&(O.P# = 5))
```

This get statement may be paraphrased as follows: Find the names and addresses of every factory such that there exists an order tuple having the same supplier number together with an associated part number of value 5.

Where a quantifier appears at the extreme left of a qualification, it is moved up to the range statement for

the tuple variable which is quantified. The query is then expressed as

```
RANGE  FACTORY  F
RANGE  ORDER  O  SOME
GET  W  (F.SNAME, F.ADDRESS) : (F.S# = O.S#) & (O.P# = 5)
```

It is thus possible to express a query in the implemented ALPHA sublanguage using the attributes SOME or ALL on range statements, so that no quantifiers appear explicitly in the get statement.

Multiple Tuple Variables with Common Range

** Find the supplier numbers of those factories which have the same address as factory IRONWORKS.

```
RANGE  FACTORY  F
RANGE  FACTORY  E  SOME
GET  W  (F.S#) : (E.SNAME = 'IRONWORKS') &
                 (F.ADDRESS = E.ADDRESS)
```

In this query F and E have the common range FACTORY but must be distinguished.

Single Universal Quantifier

** Find the supplier numbers of those factories which do not supply part number 3.

```
RANGE  FACTORY  F
RANGE  ORDER  O
GET  W  (F.S#) : ¬∃O((F.S# = O.S#)&(O.P# = 3))
```

This get statement is logically equivalent to

```
GET  W  (F.S#) : ∀O¬((F.S# = O.S#)&(O.P# = 3))
```

where the quantifier is at the extreme left of the qualification expression (i.e., in prenex normal form). When this universal quantifier is moved to its corresponding range statement, the query appears as

```
RANGE  FACTORY  F
RANGE  ORDER  O  ALL
GET  W  (F.S#) : ¬((F.S# = O.S#)&(O.P# = 3))
```

As ¬(p & q) is equivalent to ¬(p) | ¬(q) where p and q are monadic or dyadic terms, the ¬ symbol in the above get statement can be eliminated and the query written as

```
RANGE  FACTORY  F
RANGE  ORDER  O  ALL
GET  W  (F.S#) : (F.S# ¬= O.S#) | (O.P# ¬= 3)
```

Multiple Quantifiers

** Find the names of all factories, each of which supplies all tasks.

```
GET  W  (F.SNAME) : ∀T∃O((F.S# = O.S#)&(O.J# = T.J#))
```

In this statement the meaning is changed when the ordering of quantifiers is altered. Thus, when the quantifiers are moved to their corresponding range statements, these statements must appear in the order shown below:

```
RANGE  FACTORY  F
RANGE  TASK  T  ALL
RANGE  ORDER  O  SOME
GET  W  (F.SNAME) : (F.S# = O.S#) & (O.J# = T.J#)
```

Multiple Relation Target List

** For each task obtain as a triple the job number, job name and supplier address for all factories which supply that task.

```
RANGE   TASK   T
RANGE   FACTORY   F
RANGE   ORDER   O   SOME
GET   W   (T.J#, T.JNAME, F.ADDRESS) : (T.J# = O.J#) &
                                       (O.S# = F.S#)
```

A multiple relation target list sometimes causes attributes of a workspace to inherit identical names. An established data base relation requires that its attributes be distinctly named. It is, however, possible for an attribute of one relation to have the same name as an attribute of some other relation. Consider the possibility that the attributes JNAME and SNAME in the sample data base had been named as NAME. A query of the form

```
RANGE   MATERIAL   M
RANGE   TASK   T
GET   W   (M.NAME, T.NAME)   ...
```

will cause these two attributes of W to inherit identical names and therefore cannot be distinguished.

In such cases when information is transmitted from the data base into a workspace W in accordance with a target list of the form

$$(L_1.A_1, L_2.A_2, \ldots, L_n.A_n)$$

the jth attribute of W inherits the prefixed name R_j where

L_j is a local relation name whose range is the relation R_j and A_j is an attribute of relation R_j. This prefix is dropped if the unprefixed name does not occur as part of any other attribute name for that workspace.

Using a Workspace as Just Another Relation

** Find the names and addresses of all factories which supply at least those tasks supplied by factory PAINTWORKS.

One approach to formulate this query is to decompose it into two queries. A unary relation containing the set of job numbers supplied by PAINTWORKS is first constructed in workspace W1. This set is then used to obtain the required suppliers' names and locations as a binary relation in workspace W2.

```
RANGE  ORDER  O
RANGE  FACTORY  F  SOME
GET  W1  (O.J#) : (O.S# = F.S#) & (F.SNAME = 'PAINTWORKS')
RANGE  FACTORY  F
RANGE  W1  X  ALL
RANGE  ORDER  O  SOME
GET  W2 (F.SNAME, F.ADDRESS) : (F.S# = O.S#) &
                                (O.J# = X.J#)
```

Multiple w components in the Qualification

** Find the supplier numbers of those factories which supply the materials for job number 25 or supply part number 7 to job number 47.

```
RANGE  ORDER  O
GET  W  (O.S#) : (O.J# = 25) OR (O.J# = 47) & (O.P# = 7)
```

Simple Functions in the Target List

** Find the number of factories which do not supply part number 5.

```
RANGE  FACTORY  F
RANGE  ORDER  O
GET  W  COUNT(F.S#) : ¬∃O((F.S# = O.S#)&(O.P# = 5))
```

Expressing the qualification expression in prenex normal form and eliminating the ¬ symbol, the query is expressed as

```
RANGE  FACTORY  F
RANGE  ORDER  O  ALL
GET  W  COUNT(F.S#) : (F.S# ¬= O.S#) | (O.P# ¬= 5)
```

The result to this query is a single value. This value in the workspace W is referred to as W.S# .

Boolean Functions in the Qualification

Let X be a tuple belonging to relation R, and let A be a numeric attribute of R and N be an integer. Then

TOP(N, X.A) has the value true if the A-component of tuple X has a value which is Nth largest in R.A (the projection of R on A); otherwise its value is false.

BOTTOM(N, X.A) is similarly defined, except 'largest' is replaced by 'smallest'.

** Find the job names of all tasks having the fourth highest priority.

```
RANGE  TASK  T
GET  W  (T.JNAME) : TOP(4, T.PRIORITY)
```

When a boolean function designator appears in the qualification expression of a get statement then the target list and qualification of that statement must refer to the same relation.

Image Functions in the Target List

In the ORDER relation the set of supplier numbers associated with a given part number is an example of an *image set*. The function ICOUNT applied to this set yields the number of factories which supply this particular material.

This kind of construction to generate image sets is so common that composite functions called I-functions have been introduced. For example, ICOUNT applied to an ORDER tuple, the P# attribute and the S# attribute yields the number of factories which supply the material identified by the P# component of that ORDER tuple. ITOTAL applied to an ORDER tuple, and the attributes P# and QUANTITY yields the total quantity being supplied with the material identified by the P# component of that ORDER tuple. IMAX, IMIN and IAVERAGE correspond in an analogous way to MAX, MIN and AVERAGE.

** For each part number supplied to a task, find as a triple the part number, the job number and the total quantity on order for that material supplied to that task.

```
RANGE  ORDER  O
GET  W  (O.P#, O.J#, ITOTAL(O, (P#,J#), QUANTITY) )
```

In this query image sets of QUANTITY values are conceptually

formed for each distinct pair of values (O.P#, O.J#). The name inheritance rule results in the following attribute names for the workspace W:

W.P#, W.J#, W.QUANTITY

** For each part number supplied to a task, find the part number, the job number and the number of factories which supply that material to that task.

```
RANGE  ORDER  O
GET  W  (O.P#, O.J#, ICOUNT(O, (P#,J#), S#) )
```

Image Functions in the Qualification

Image functions can appear in both the target list and the qualification expression.

** Find the part numbers of materials supplied to more than three tasks.

```
RANGE  ORDER  O
GET  W  (O.P#) : ICOUNT(O, P#, J#) > 3
```

3.4.2 Updates

Simple Update

** Alter the job name for job number 20 to CAM.

```
RANGE  TASK  T
HOLD  W  (T.JNAME) : (T.J# = 20)
W.JNAME = 'CAM'
UPDATE
```

Upon receipt of a HOLD operation the system retains enough primary key information (associating it with the

workspace W) so that it can perform the update properly. If the value of the primary key in W has changed when the UPDATE is received, the system flags an error.

Primary Key Update

A distinction must be made between updates of primary keys (including components) and updates of other attributes because of the crucial role of primary keys in identification and search, and the great impact on the user community of changes in these keys.

** Change the primary key of the task with job number 6 to 7.

```
RANGE  TASK  T
GET  W  (T) : (T.J# = 6)
RANGE  TASK  T
DELETE  T : (T.J# = 6)
W.J# = 7
PUT  W  TASK
```

The query sequence statement retrieves the entire TASK tuple which has 6 in its J# component. The delete sequence statement then deletes that tuple from the data base relation TASK (not from W). The assignment statement alters the primary key to 7. Finally, the put statement inserts this new tuple into the data base relation TASK.

Multi-relation Update

** A partial shipment of part number 4 has arrived for job number 5 due on 1 April 1974. Reduce the corresponding quantity by 40 and increase the quantity

on hand for that material by 40.

```
RANGE  ORDER  O
HOLD  W  (O.QUANTITY) : (O.P# = 4) & (O.J# = 5) &
                        (O.DATEDUE = '1.4.74')
W.QUANTITY = W.QUANTITY - 40
UPDATE
```

The system retains at least the full compound value of the primary key from the HOLD to the UPDATE, in order to be able to return the new values to the proper tuples.

```
RANGE  MATERIAL  M
HOLD  W  (M.QOH) : (M.P# = 4)
W.QOH = W.QOH + 40
UPDATE
```

3.4.3 Deletions

Simple Deletion

** Delete all tuples from the data base relation TASK.

```
RANGE  TASK  T
DELETE  T
```

No workspace is required as an information sink and this applies to all deletions.

Qualified Deletion

** Delete all tuples of the data base relation ORDER involving factory STEELWORKS and task CAG in combination with one another.

```
RANGE  ORDER  O
RANGE  FACTORY  F  SOME
RANGE  TASK  T  SOME
DELETE O : (F.SNAME = 'STEELWORKS') & (O.S# = F.S#) &
           (O.J# = T.J#) & (T.JNAME = 'CAG')
```

3.4.4 Insertions

Simple Insertion

** Insert into the ternary data base relation TASK the 3-tuples now located in workspace W.

```
PUT  W  TASK
```

The system checks that no duplicate values of the primary key are introduced by the PUT operation.

Partial Insertion

** Insert into the quaternary relation MATERIAL the (P#, PNAME, QOH) triples now located in workspace W.

```
PUT  W  MATERIAL.(P#, PNAME, QOH)
```

The attribute ordering of W is as follows: first P#, then PNAME followed by QOH . The element ordering in W may be any ordering.

The system converts each triple from W into a 4-tuple by appending the absent value as the value of MATERIAL.WEIGHT .

Ordered Insertion of Bulk Data

** Same as above, except that the element ordering in W is by P# increasing, and the system is to be advised

for efficiency reasons.

```
PUT  W  MATERIAL.(P#, PNAME, QOH)  UP P#
```

If it happens that the data base relation MATERIAL is represented in storage by a set of data ordered on P#, the system can make this insertion more rapidly if advised about the ordering in W. This could be important in cases where the number of tuples to be inserted is large.

3.4.5 Dropping and Establishing Data Base Relations

Dropping a Data Base Relation

** Remove all information about the data base relation TASK.

```
DROP  TASK
```

If any of the tuples of TASK still exist, they are deleted.

Dropping one or more Attributes of a Data Base Relation

** Remove all information about the attributes WEIGHT and QOH from the data base relation MATERIAL.

```
DROP  MATERIAL.(WEIGHT, QOH)
```

Establishing a New Data Base Relation

Suppose a new relation has been constructed in workspace W and it is desired that this relation be established in the data base for shared use. Before the data can be copied from W into the data base, the relation to be established must first be declared. This involves giving it a name which is not in conflict with those of

other data base relations and naming its attributes in accordance with community-established rules, identifying its primary key and specifying the data types of these attributes.

** Establish the relation now located in workspace W as a data base relation SUPPLIER. Assume that the relation in W has identical attributes as data base relation FACTORY.

```
NEW  SUPPLIER (S#, KEY, CHAR, 5)(SNAME, CHAR, 15)
              (ADDRESS, CHAR, 20)
PUT  W  SUPPLIER
```

3.4.6 Input and Output Facilities

Input Facility

** Declare two integer variables and assign to them the values 5 and 10.

```
READ (I, J)
```

This read statement implicitly declares two integer variables I and J. The system suspends all other accesses until the two integer values are read and assigned to the corresponding variables.

Output Facility

** List the information contained in the relation located in workspace W , and the values of the variables I and J.

```
LIST (W, I, J)
```

3.5 SUMMARY OF DATA SUBLANGUAGE ALPHA

The implemented ALPHA sublanguage possesses the following characteristics:

(i) the full power of the relational calculus - an applied predicate calculus with tuple variables

(ii) the capability of specifying any of the operations:

- fetch value or set of values
- change value or set of values
- insert element (i.e., tuple) or set into a relation
- delete element (i.e., tuple) or set from a relation
- declare a relation and its attributes for inclusion in the established set of data base relations
- drop a relation or any of its attributes from the data base
- assign value to declared variable or set of values to declared variables
- list value of variable or set of values of variables, and/or information contained in a relation or set of relations

(iii) the capability (through range statements) of delimiting the scope of interaction between a user on the one hand and the data base on the other.

4. DESCRIPTION OF THE TRANSLATOR

4.1 INTRODUCTION

As mentioned earlier, the interactive translator carries out the following functions:

(i) accepts as input source statements of the implemented DSL ALPHA

(ii) checks that conditions are met for the relational calculus

(iii) produces as output a set of coding tables.

4.2 TRANSLATION PROCESS

Figure 4.1 shows the translation process in detail; dotted arrows represent flow of information while solid arrows indicate program flow.

Analysis of the source statement and construction of the coding tables are performed in a parallel, interlocked manner. To do this the translator builds up during the analysis phase a symbol table which is used during both analysis and construction.

4.2.1 Information Tables

During translation information about identifiers and constants is stored in the following tables:

(i) symbol table

(ii) constant table

(iii) workspace table

(iv) variable table.

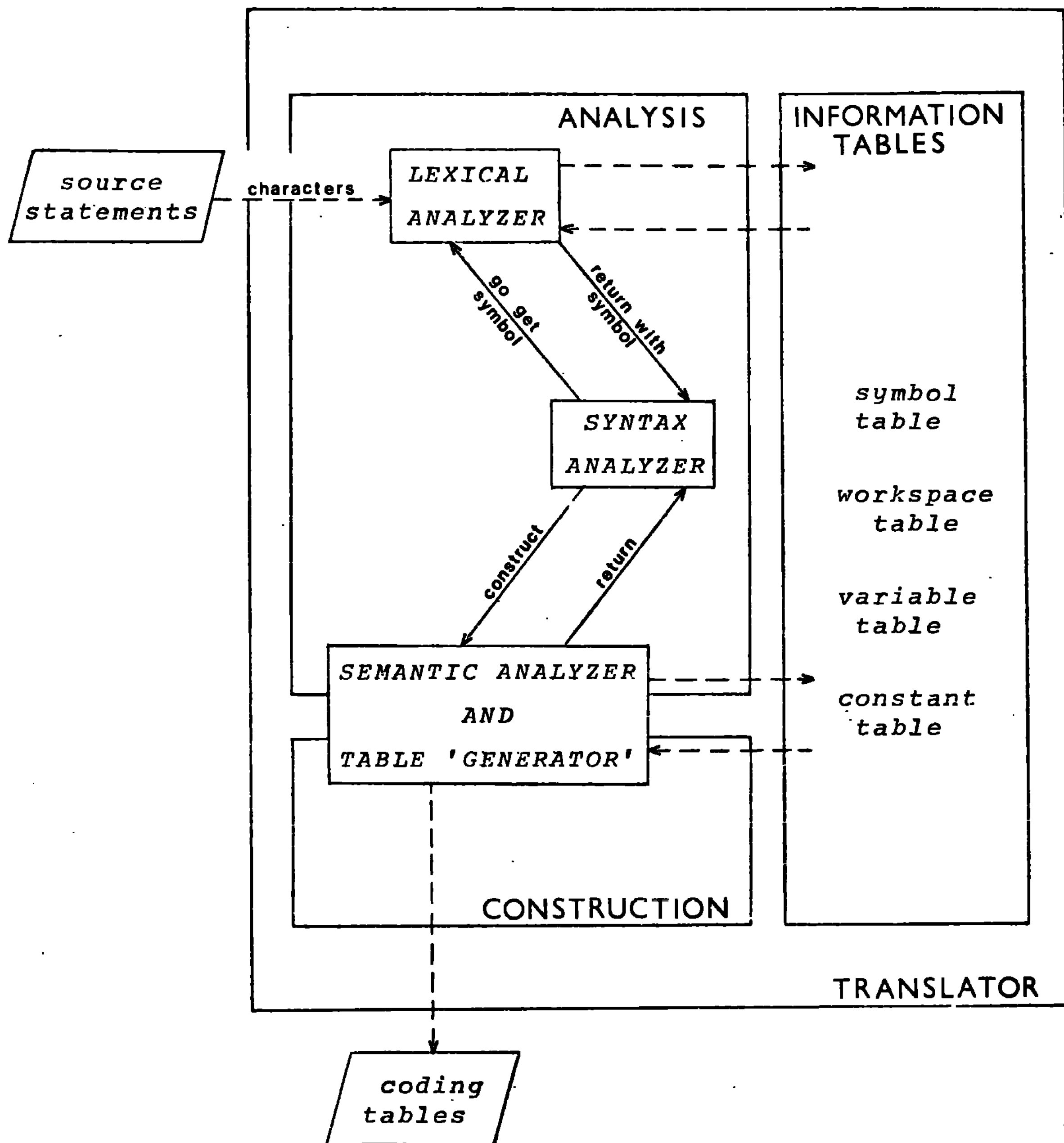


Fig. 4.1 Logical Connection of the Translator Parts

As each source statement is analyzed, information about identifiers is stored in the symbol table. This is a table of identifiers together with their attributes. The attributes are the type of identifier and any other information about it which is needed to 'generate' the coding tables. The constant table stores constants used in the source statement. The symbol table and constant table are temporary tables, i.e., they are cleared at the beginning of a user's session and after a given set of coding tables has been interpreted.

Information about workspace names used and variables declared is saved in the workspace table and variable table respectively for later use. Both of these tables are permanent tables, i.e., the information contained in them is valid throughout the user's session.

The symbol table has the form

	Identifier	Lexeme Value	Information Field
Entry 1			
Entry 2			
Entry N			

where the identifiers are the actual symbols and the lexeme values and information field their attributes. Only entries for local relation names in the symbol table utilize the information field; this field will indicate the attribute of the associated local relation name (i.e., whether it is free or quantified). The workspace table has only the identifier argument, and the variable

table the identifier argument plus the information field. This information field in the variable table will indicate the values of the declared variables and such values are entered by the sublanguage interpreter.

4.2.2 The Lexical Analyzer

The lexical analyzer or scanner is that part of the translator which scans the characters of each source statement from left to right and builds the source program words or symbols - integers, composite symbols (e.g., ¬=), identifiers, keywords, etc. The internal representation of these symbols (i.e., lexeme values) are then passed on to the syntax analyzer. Thus the syntax analyzer never actually sees the symbols but their lexeme values. These symbols are arranged into classes and each symbol is given a representation within a class. For example, the class of quantifiers are internally represented as

ALL 0110
SOME 0210 .

Each lexeme value is stored as a four digit number. The last two digits constitute the class number and the first two (if present) the number within that class. Thus the symbols ALL and SOME are represented by 1 and 2 respectively, within class 10. The different classes of symbols and their lexeme values are given in Appendix C.

The scanner is a routine called by the syntax analyzer whenever the syntax analyzer requires a new lexeme value. When called, this routine recognizes the next source symbol and passes the lexeme value of this symbol to the syntax analyzer.

The lexical scan thus involves the following sequence of steps:

(i) Is source symbol a string constant or delimiter? If it is proceed to step (iv).

(ii) Is source symbol a keyword (reserved word) or identifier? (see below) If it is proceed to step (iv).

(iii) Assume symbol to be an integer or real number, and construct its value.

(iv) Assign to symbol its lexeme value and return this value to the syntax analyzer.

In (ii) above, the list of keywords is first searched to see if the symbol is in that list. If symbol is not a reserved word, it must then be an identifier. A keyword is searched with the technique called hash coding which uses some computable function (hash function) [(a)] of the numeric representation of the keyword. The hash function, determines the starting point in the table (hash table) [(b)] to search for the keyword. The hash table (see Table 4.2) can be visualized as containing a key field (hash addresses) and an information field (keywords). A search for a

---

(a) The *hash function* used in the lexical analyzer is $N - (N \div 22) \times 22$ ($\div$ denotes integer divide) where $N = |M| \div L$; M being the numeric representation of the first four characters of the keyword and L the number of characters of the keyword. A series of tests were carried out for the function $N - (N \div k) \times k$ where $k = 5, 6, 7, \ldots, 29$. The value of k was chosen to be 22 because this is the smallest integer which gives the hash function a fairly uniform distribution of hash addresses over the list of keywords.

(b) The *hash table* has been constructed as a conventional hash table with external overflow.

KEY	INFORMATION
0	COUNT
1	DELETE
2	DROP
3	READ
4	FIXED31
5	RANGE
6	ALL
	NEW
	KEY
7	GET
8	PUT
	STOP
	ICOUNT
9	RELEASE
11	HOLD
	AND
12	MAX
	UPDATE
13	MIN
	TOP
	IAVERAGE
14	UP
	IMIN
15	FLOAT
	CHAR
	CHARVAR
16	DOWN
	IMAX
17	AVERAGE
	BOTTOM
18	SOME
	OR
	ITOTAL
19	TOTAL
	FIXED
20	FLOAT16
21	LIST

Table 4.2 Hash Table

keyword thus reduces to calculating the function and performing a sequential search on the entry (or entries) at the hash address.

As each identifier is encountered the following sequence of steps are carried out:

(i) Perform a search on the two permanent tables (workspace table and variable table) to see if that identifier is in either table. If that identifier is in either table proceed to step (iv).

(ii) Perform a search on the symbol table to see if that identifier is in that table. If that identifier is in the table proceed to step (iv).

(iii) Append to the end of the symbol table that identifier and its lexeme value.

(iv) Return a pointer to the location of that identifier.

The type of an identifier is not known until the analysis of the source statement is complete, and is determined by the syntactic position of the identifier in the statement. Similarly, an identifier is inserted in the workspace table or variable table only when syntactic context shows that it identifies a workspace name or variable name respectively. This identification process is unique.

Entries for the source statement

RANGE SUPPLY S SOME

are initially made by the lexical analyzer in the symbol table as

SUPPLY	37	0
S	37	2

where 37 is the lexeme value for identifier. A '0' in the information field denotes that the associated local relation name is free, '1' denotes universal quantification and '2' existential quantification. For all other entries it takes the null value.

When analysis is complete the symbol table shows

SUPPLY	39	0
S	40	2

where 39 is the lexeme value for the data base relation name and 40 the lexeme value for the local relation name.

4.2.3 The Syntax Analyzer

The syntax analyzer or parser performs a complete syntax check on each source statement of the implemented ALPHA sublanguage.

The parser consists of a sequence of syntax statements and a routine which directs the execution of these statements. This routine uses a syntax stack. The syntax statements are referred to as Floyd Production Language (FPL) statements which were generated using a modified version of an algorithm used by DeRemer, [11] which maps the set of BNF productions in the data sublanguage into FPL statements. The FPL statements consist of labelled, mutually exclusive groups of statements called sections. Each section has a specific task to perform and is activated by transfer of control to its first statement.

The syntax analyzer [c] thus consists primarily of a sequence of FPL statements or productions, each of which has the form

L : a | *b* *Rn* *k* → | (*l*) *...* | *S* *F*(*x*)

where *L* (if present) is a production label

a and *b* are source symbols [d]

Rn (if present) is the semantic routine

k (if any) is the number of symbols to be removed from top of stack

l (if any) is the number of source symbols to be scanned

... (if any) denote the number of source symbols to be stacked and scanned

S is a success label

F (if present) is a fail label

x is an error message number (if the fail label is present).

The first vertical bar (|) indicates the top of the stack; the symbol to its left represents the one being looked for in the stack. The appearance of *k* → and *...* on the left of the

---

(c) This particular syntactic analysis was used for the translator because the syntax of the DSL ALPHA was not fully specified at the start of the project and work proceeded with writing the bulk of the lexical analyzer and the routine which directs the execution of FPL statements. It was found that, as work progressed, it was quite easy to alter the statements of the parser whenever changes occurred in the specification of the syntax of this data sublanguage.

(d) The symbol *ANY* when substituted for *a* and/or *b* matches any other symbol in the data sublanguage.

next two |'s indicates that the stack is to be transformed.

A production thus indicates pattern matching and usually a stack transformation. During the execution of the parser there is a current production from which the symbols *a* and *b* are compared with the symbol at the top of the syntax stack and the current source symbol respectively. If they do not match and if the fail label is absent, then the next production in the sequence becomes current and the comparison is again made. This continues until a match occurs. When it does, the semantic routine is called and *k* symbols are removed from the stack. The execution of the actions (*l*) and *...* will cause the scanner to be called. The production labelled *s* then becomes the current production and matching continues.

If pattern matching fails at a current production and a fail label is present, a syntax error message will be printed and an error section of the productions becomes current.

Suppose the production

```
T(46):  LRNAME | ,   R17   1 → | (1)   *|  <ANAME-SEQ1>H
                                           FAIL_1(12)
```

is the current production. If the top symbol of the stack is a local relation name and the current source symbol is a comma, then the following occurs:

(i) the semantic routine R17 of the semantic analyzer is executed

(ii) the top symbol is removed from the stack

(iii) the next source symbol is scanned which replaces the current symbol

(iv) the current symbol is stacked and the next source symbol is scanned which then becomes the current symbol

(v) the production labelled <ANAME-SEQ1>H becomes the current production.

If pattern matching fails, the error message number 12 will be printed and the production labelled FAIL_1 becomes current.

The FPL statements of the parser are coded up in the form of tables. The parser thus consists of these tables and a routine which interprets the tables. The FPL statements of the parser are given in Appendix D.

4.2.4 The Semantic Analyzer and Table 'Generator'

In the translator the table 'generation' parts are fused with the semantic routines of the semantic analyzer. When a syntactic construct is recognized, the syntax analyzer calls a semantic routine which executes the following functions:

(i) checks the construct for semantic correctness, i.e., a construct may be syntactically correct but its meaning may be incorrect and it is this type of error that is detected by the semantic analyzer. These are either errors that cannot be detected by the syntax analyzer or errors that can be detected during syntactic analysis but have been left to the semantic analyzer so as to simplify the FPL statements,

e.g., (a) Correct specification of local relation names where appropriate

Suppose a query is expressed as

```
RANGE   FACTORY  F
RANGE   ORDER  O
GET   W   (F.SNAME, F.ADDRESS) : (F.S# = O.S#) & (O.P# = 3)
```

A semantic error will be detected since the local relation name O appearing in the qualification expression and not in the target list of the get statement must be quantified.

A semantic error will also be detected for the query

```
RANGE   FACTORY  F
RANGE   ORDER  O  ALL
GET   W   (F.S#) : (F.S# ¬= O.S#) | (O.P# ¬= 3)   UP   O.S#
```

as the local relation name O appearing in the element ordering expression is not specified in the target list of the get statement.

(b) Correct use of function designators

A query expressed as

```
RANGE   MATERIAL   M
GET   W   COUNT(M.P#) : TOP(1, M.QOH)
```

will cause a semantic error to be detected because the occurrence of a function designator in the target list does not permit the use of a boolean function designator in the qualification expression of the get statement.

A semantic error will also be flagged for the query

```
RANGE   ORDER   O
GET   W   (O.P#) : ICOUNT(O, P#, J#) > 3.5
```

as the number of tasks being supplied with the material identified by the P# component of the order tuple must be

an integer constant.

(c) Consistency of attributes specified/declared

```
RANGE  ORDER  O
GET  W  (O.P#, O.J#, ITOTAL(O, P#, QUANTITY) )
```

A semantic error will be flagged for this query as there is one less attribute specified in the attribute list of the image function designator in the target list of the get statement.

The newly declared data base relation

```
NEW  SUPPLIER (S#, KEY, CHAR, 5)(SNAME, CHAR)
```

will also cause a semantic error to be detected as the attribute length is not specified for the attribute SNAME of data type character string.

(ii) fills in the necessary information (if any) in the information tables,

e.g., Identification of certain identifiers after syntactic analysis

An insertion is made for an identifier in the workspace table or variable table only when syntactic context shows that it identifies a workspace name or variable name respectively.

(iii) 'generates' the coding tables; these tables, produced in a form suitable as input to the sublanguage interpreter, are explained in Section 5. While these tables are being 'generated', checks are made to ensure that the tables do not overflow,

e.g., Overflow of coding tables

An error message showing implementation restriction

will be printed if the number of entries made in a coding table exceeds the maximum number permitted for that table.

(iv) fills in values of constants (if any) in the constant table,

e.g., Constants encountered in source statements.

The value 5 encountered in the assignment statement

W.P = 5

is put in the constant table and a pointer to its location in that table is returned.

In some of the semantic routines of the semantic analyzer, a stack is utilized to facilitate the ease of checking for semantic correctness of constructs and filling in of tables. Where possible, the same semantic routine has been used for constructs with similar meaning.

The syntax analyzer continues to analyze the source statement once control is returned to it.

4.3 IMPLEMENTATION OF THE TRANSLATOR

The translator is implemented as the following routines:

(i) *MAINPRG* and *PROG* which initialize the FPL statements of the parser

(ii) *PARSE* which directs the execution of the FPL statements

(iii) *SYN_ERR* which prints out the appropriate error message for each syntax error detected

(iv) *SEM_ROU* which checks the semantic correctness of each source statement and fills in the appropriate tables

(v) *LEX_ANL* which assigns lexeme values to source language symbols and adds an entry to the symbol table for each new identifier encountered.

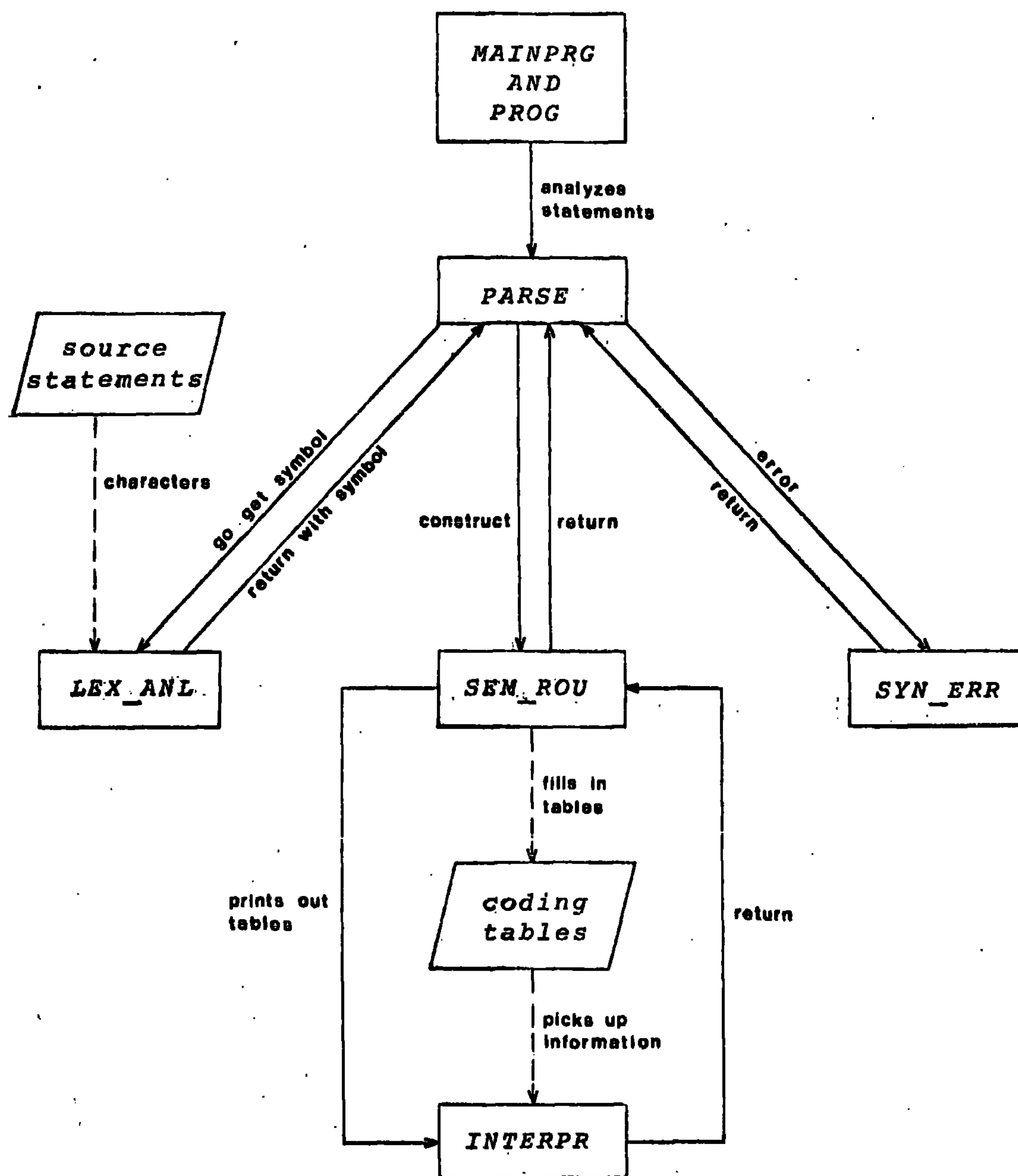


Fig. 4.3 Logical Connection of the Translator Routines

(vi) *INTERPR* [e] which prints out the coding tables.

Figure 4.3 represents a logical connection of these translator routines. Some further details on the implementation of the translator are given in Appendix E.

4.4 WARNINGS AND ERRORS

Warnings and errors flagged by the translator fall into three classes:

(i) warnings and errors detected by the lexical analyzer

(ii) syntax errors detected by the syntax analyzer

(iii) severe errors detected by the semantic analyzer.

Translation of the source statement continues after a warning (or warnings) is flagged, e.g., a warning message will be printed if the length of an identifier exceeds twenty characters.

An error (or errors) detected in the source statement will cause that statement to be deleted and control to return to the appropriate section. No provision is made for any sophisticated error recovery procedure as the translator, being interactive, enables the on-line user to correct any error quite easily. Error recovery could be undesirable at times.

Lexical errors detected in source statements include alphabetic characters appearing in numbers (e.g., 27.A35), illegal composite symbols (e.g., ¬<) and characters, missing end quotes, etc.

(e) As the sublanguage interpreter is not yet implemented, this routine presently prints out the coding tables 'generated'.

When a syntax error is detected in a construct one of the productions, labelled fail_1, fail_2 or fail_3, becomes current. The source symbols are then scanned until the end of the statement is reached and no further syntactic analysis is performed. A current fail_1 section indicates a syntax error in a DSL ALPHA statement, fail_2 section indicates a syntax error in a computational facility statement and fail_3 section an invalid source statement.

A semantic error will cause no further calls of semantic routines but will allow syntactic analysis to continue until the end of the statement is reached in which case a production labelled fail_4 or fail_5 becomes current. A current fail_4 section indicates a semantic error in a DSL ALPHA statement and fail_5 section a semantic error in a computational facility statement.

5. EXPLANATION OF THE CODING TABLES

5.1 INTRODUCTION

The following tables are 'generated' by the translator:

(i) assignment table (ASS_TAB)

(ii) attribute list table (AL_TAB)

(iii) attribute table (ATT_TAB)

(iv) dyadic term in one variable table (DTOV_TAB)

(v) dyadic term table (DT_TAB)

(vi) element ordering expression table (OR_TAB)

(vii) get target table (GT_TAB)

(viii) hold target table (HT_TAB)

(ix) identifier table (ID_TAB)

(x) local relation name table (LRN_TAB)

(xi) function and monadic term table (FN_MT_TAB)

(xii) quota table (Q_TAB)

(xiii) range table (RA_TAB)

(xiv) relation name table (RN_TAB)

(xv) statement table (ST_TAB)

(xvi) w component precedence table (WP_TAB)

(xvii) workspace name table (WN_TAB).

These coding tables, logically of the form described by Palmero, [3] are presented in a precise manner which clearly reflects the subsequent operations to be carried out by the data sublanguage interpreter.

The interpreter is invoked after translation of the following statements:

(i) get statement

(ii) hold statement

(iii) delete statement

(iv) drop statement

(v) put statement

(vi) new statement

(vii) release statement

(viii) update statement

(ix) read statement

(x) list statement

(xi) assignment statement.

Table 5.1 shows the tabular representation of these statements. Get, hold and delete statements must be preceded by range statements.

Section 5 is best understood by studying the DSL ALPHA examples in Section 3.4; the coding tables 'generated' by the translator for these examples together with the required information tables are presented in Appendix F.

5.2 OUTLINE OF THE CODING TABLES

The coding tables are themselves normalized relations; table 5.2 shows the attributes of these tables. Associated with each coding table (except those with one object) is a variable which indicates the number of rows filled in that table. A zero value for this variable of a table,

STATEMENTS	TABULAR REPRESENTATION
Get Statement	RA_TAB ST_TAB WN_TAB Q_TAB GT_TAB DT_TAB, DTOV_TAB, FN_MT_TAB WP_TAB OR_TAB
Hold Statement	RA_TAB ST_TAB WN_TAB HT_TAB DT_TAB, DTOV_TAB, FN_MT_TAB WP_TAB OR_TAB
Delete Statement	RA_TAB ST_TAB LRN_TAB DT_TAB, DTOV_TAB, FN_MT_TAB WP_TAB
Drop Statement	ST_TAB RN_TAB AL_TAB
Put Statement	ST_TAB WN_TAB RN_TAB AL_TAB OR_TAB
New Statement	ST_TAB RN_TAB ATT_TAB
Release Statement	ST_TAB
Update Statement	ST_TAB
Read Statement	ST_TAB ID_TAB
List Statement	ST_TAB ID_TAB
Assignment Statement	ST_TAB ASS_TAB

Table 5.1 Tabular Representation of Statements

CODING TABLES		ATTRIBUTES	
RA_TAB	*	RNAME	(relation name)
	*	LRNAME	(local relation name)
		QUANT	(quantifier)
		WCOMP	(w component)
GT_TAB	**	FUNCT	(function)
	*	ALIST	(attribute list)
		ALISTPTR	(attribute list pointer)
	*	LRNAME	(local relation name)
	*	ANAME	(attribute name)
DT_TAB	*	LRNAME1	(local relation name one)
	*	ANAME1	(attribute name one)
	*	LRNAME2	(local relation name two)
	*	ANAME2	(attribute name two)
	**	RELOP	(relational operator)
		WCOMP	(w component)
		TCOMP	(theta component)
DTOV_TAB	*	RNAME	(relation name)
	*	LRNAME	(local relation name)
	*	ANAME1	(attribute name one)
	*	ANAME2	(attribute name two)
	**	RELOP	(relational operator)
		WCOMP	(w component)
		TCOMP	(theta component)
FN_MT_TAB	**	FUNCT	(function)
	*	ALIST	(attribute list)
		ALISTPTR	(attribute list pointer)
	*	RNAME	(relation name)
	*	LRNAME	(local relation name)
	*	ANAME	(attribute name)
	**	RELOP	(relational operator)
		WCOMP	(w component)
		TCOMP	(theta component)
		CONTYPE	(constant type)
		CONLEN	(constant length)
		CONPTR	(constant pointer)
OR_TAB	**	ORDER	(ordering)
	*	LRNAME	(local relation name)
	*	ANAME	(attribute name)
WP_TAB		WCOMP1	(w component one)
		WCOMP2	(w component two)
	**	OPER	(operator)
HT_TAB	*	LRNAME	(local relation name)
	*	ANAME	(attribute name)

Table 5.2 (contd. on following page)

CODING TABLES		ATTRIBUTES	
ATT_TAB	*	ANAME	(attribute name)
		KTYPE	(key type)
		ATYPE	(attribute type)
		ALEN	(attribute length)
ASS_TAB		OP1	(operand one)
		OP2	(operand two)
	**	OPER	(operator)
		IOP1	(image operand one)
		IOP2	(image operand two)
* AL_TAB * ID_TAB		These tables have one attribute only	
** ST_TAB * WN_TAB Q_TAB * LRN_TAB * RN_TAB		These tables have one object only	

* Values are pointers to identifiers in the symbol, workspace or variable tables

** Values are the lexeme values for source language symbols

Null values in attribute/table names marked * and ** imply that no references are made to either the symbol, workspace, or variable tables, or to the lexeme values of source symbols.

Table 5.2 Attributes of Coding Tables

say DT_TAB, during interpretation of a get statement implied that this table was not 'generated' for that statement.

5.3 DESCRIPTION OF THE CODING TABLES

The coding tables are described in the following order:

(i) RA_TAB

(ii) ST_TAB

(iii) WN_TAB

(iv) Q_TAB

(v) GT_TAB

(vi) HT_TAB

(vii) DT_TAB, DTOV_TAB, FN_MT_TAB

(viii) OR_TAB

(ix) WP_TAB

(x) LRN_TAB

(xi) RN_TAB

(xii) AL_TAB

(xiii) ATT_TAB

(xiv) ID_TAB

(xv) ASS_TAB.

RA_TAB

The RA_TAB is constructed from a range list. For each local relation name in the w_i component of the qualification expression, the following entries are made:

(i) whether the local relation name is free (indicated by '0') or quantified ('1' for universal quantification

and '2' for existential quantification)
(ii) the associated relation name
(iii) the w component number.

ST_TAB

The entry in ST_TAB identifies the DSL ALPHA statement or computational facility statement for interpretation.

WN_TAB

The entry in WN_TAB identifies a workspace name in the workspace table.

Q_TAB

The entry in Q_TAB identifies the quota. A null entry implies that there is no quota.

GT_TAB

The GT_TAB is constructed from the target list in a get statement. An entry is made in the following cases:
(i) for each target element in the target list; such an entry identifies the local relation name or more commonly the local relation name and attribute name
(ii) for the occurrence of a function designator in the target list; such an entry identifies the function identifier and function argument
(iii) for the occurrence of an image function designator in the target list; such an entry identifies the qualified attribute list preceding this designator, as well as the image function identifier and image function argument. Only the image function argument utilizes the columns,

attribute list and attribute list pointer. This pointer points to the next attribute name in the sequence indicated by a row number in GT_TAB; the last attribute name in the sequence being indicated by the null link.

HT_TAB

The HT_TAB is constructed from the target list in a hold statement. An entry is made for the occurrence of a local relation name, or for each target element in the target list.

DT_TAB, DTOV_TAB, FN_MT_TAB

These tables are constructed from the qualification expression in a get, hold or delete statement.

DT_TAB, DTOV_TAB - These two tables contain dyadic join terms only. An entry is made in DT_TAB for each dyadic join term, and in DTOV_TAB for each dyadic join term in one variable (i.e., having the same local relation name), in the qualification expression. Such an entry identifies the variables being joined, the attributes involved in the join, and the relational operator as well as the component numbers.

FN_MT_TAB - An entry is made in FN_MT_TAB in the following cases:

(i) for each monadic term in the qualification expression; such an entry identifies the local relation name and attribute name, the relational operator, and the constant as well as the component numbers. A constant pointer determines the position of the constant in the

constant table, the constant type being indicated by the numerals 1, 3 or 5 ('1' indicates that the constant is an integer, '3' indicates a real number and '5' a character string). For character strings, the constant length is also entered.

(ii) for the occurrence of an image function designator in the qualification expression; such an entry identifies the image function identifier and image function argument, and the relational operator as well as the number. The pointer determines the position of the number in the constant table, the type being indicated by the numerals 1 and 3 (same notation as in (i) above).

(iii) for the occurrence of a boolean function designator in the qualification expression; such an entry identifies the boolean function identifier and boolean function argument. The integer constant in this boolean function argument is indicated by the constant pointer which determines its position in the constant table.

OR_TAB

The OR_TAB is constructed from the element ordering expression in a get, hold or put statement. For each occurrence of the keyword UP or DOWN in this ordering expression, the following entries are made:

(i) whether the specified ordering is UP or DOWN

(ii) the local relation name (for get/hold statement only)

(iii) the attribute name.

WP_TAB

The WP_TAB is constructed from two or more w components in the qualification expression to show precedence imposed upon the operators. An entry is made for a logical combination of two w components. Such an entry identifies the two w components as operands together with their logical operator. A negative value in any of the w component attributes implies that a result obtained from an earlier combination is used as the operand; the result being indicated by a row number in WP_TAB.

LRN_TAB

The entry in LRN_TAB identifies a local relation name in the symbol table.

RN_TAB

The entry in RN_TAB identifies a data base relation name in the symbol table or a workspace name in the workspace table.

AL_TAB

The AL_TAB is constructed from the attribute name sequence in a drop statement or put statement. An entry is made for each attribute name in this sequence.

ATT_TAB

The ATT_TAB is constructed from the attribute record in a new statement. For each attribute field in the record, the following entries are made:

(i) the attribute name

DURHAM UNIVERSITY SCIENCE 22 MAY 1975

(ii) whether the attribute name is a component of or is the primary key (indicated by '1'), or not (indicated by '0')

(iii) the attribute type ('1' indicates FIXED31, '2' indicates FIXED, '3' indicates FLOAT16, '4' indicates FLOAT, '5' indicates CHAR and '6' CHARVAR).

(iv) the attribute length (for character strings).

ID_TAB

The ID_TAB is constructed from the variable list or identifier list in a read statement or list statement respectively. An entry is made for each identifier in the variable list or identifier list.

ASS_TAB

The ASS_TAB is constructed from a Reverse Polish representation of the assignment statement. An entry is made for each pair of operands in the Reverse Polish expression of this statement. Such an entry identifies the two operands and the operator following them. Operands are indicated by various forms; values in the two image operand columns indicate what the values in the two operand columns represent correspondingly. A '0' indicates absence of an operand, '1' indicates a pointer to an integer in the constant table, '2' indicates a pointer to an identifier in an information table, '3' indicates a pointer to a real number in the constant table, '4' indicates a line number in ASS_TAB and '5' a pointer to a character string in the constant table.

6. CONCLUSIONS

6.1 GENERAL

The principal motivation for the relational approach was the need for a high degree of data independence. Such a relational model of data, because of its simplicity, provides a unifying feature for the development of the proposed data base system with reference to DSL ALPHA. The user is provided with only a logical structure and hence need not be concerned with the complexity of linkages, networks, repeating groups and indexes.

The majority of users should not have to learn either the relational calculus or algebra in order to interact with such data bases. However, requesting data by its properties (the calculus approach) is far closer to natural language rather than formulating a sequence of algebraic operations. Thus, a calculus-oriented language provides a good target language for a more user-oriented source language. The simplicity of the use of DSL ALPHA as a high level storage and retrieval language has therefore been considered the justification for its implementation.

6.2 SUGGESTED IMPROVEMENTS TO THE TRANSLATOR

Additional facilities to the implemented ALPHA sublanguage can be incorporated. Below are two of the suggested improvements to the translator:

(i) At translation time no provision is made to check

that the data base relations and their attribute names specified in queries, updates, etc. exist in the user's data base. Data type compatibility of these attributes is also not checked at translation time. Such errors will be detected only during interpretation when the coding tables are interpreted. This can be rectified by incorporating the following facility in the data sublanguage:

```
<USING-STATEMENT> ::=  USING <RELATION-LIST>
<RELATION-LIST> ::= <RELATION-LIST> , <RELATION-NAME>
                    <RELATION-NAME>
```

When each relation name is encountered in the using statement, the translator will then inspect the relation index [(a)] to find a match for that relation name. When a match is found the attribute names of that relation together with their data types are brought down to the symbol table and any subsequent occurrence of an attribute name is matched in the symbol table. An error is flagged when matching fails. A query will then be written as

```
USING   MATERIAL
RANGE   MATERIAL  M
GET   W   (M.P#, M.PNAME, M.QOH) : (M.QOH < 100)
```

The organization of the symbol table will have to be more complicated to deal with this facility.

(a) The relation index consists of a list of relation names in the data base.

(ii) The source statements of the data sublanguage can incorporate the concept of 'implicit' relations equivalent of the DEFINE command in the Peterlee IS/1 System. [8] The translator will then produce the set of information and retrieval tables suitable for storage and subsequent interpretation.

The following DSL ALPHA statements

```
DEFINE  PARTS
  USING  MATERIAL
  RANGE  MATERIAL  M
  GET  W  (M.P#, M.WEIGHT) : (M.QOH < 100)
END
```

will create PARTS and define it as the part numbers and weights of those materials where the quantity on hand is less than 100. This definition is then translated and the tables associated with PARTS are stored. If, at a later time, the response to this query is required, the retrieval tables will then be interpreted by the execution of the statement

```
EXECUTE  PARTS
```

to generate the response relation in workspace W.

The execution of the statement

```
DESTROY  PARTS
```

will destroy the tables stored under file PARTS.

With the above two facilities incorporated in the translator the system becomes more flexible and secure. Flexibility is achieved by DEFINE and security by USING in that all

attributes can be checked for data type compatibility when the stored tables are subsequently interpreted, i.e., if a user changes any of the data types in the data base between translation and interpretation this can be easily checked at interpretation time since the attribute names of each relation together with their types have been stored in the symbol table.

6.3 SUGGESTION FOR FURTHER WORK

Queries expressed in DSL ALPHA typically require the following:

(i) that the user defines extra tuple variables which have as values rows or portions of rows of a relation

(ii) that the user states the query using terms associated with universal and existential quantifiers.

To overcome these difficulties for the more infrequent data base user, Boyce and Chamberlin [12,13] have introduced a natural block Structured English Query Language (SEQUEL) for expressing queries on a relational data base without requiring the mathematical sophistication of the predicate calculus (bound variables, quantifiers, etc.).

It is suggested that further work could be carried out on the development of a translator to convert SEQUEL statements into similar tables as that produced for the implemented DSL ALPHA. This will involve the respecification of the syntax of SEQUEL.

R E F E R E N C E S

[1] Codd, E F, 'A Relational Model of Data for Large Shared Data Banks', Comm ACM, Vol 13, No 6, pp 337-387, 1970.

[2] Codd, E F, 'A Data Base Sublanguage Founded on the Relational Calculus', Proc ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Calif, 1971.

[3] Palmero, E F, 'A Data Base Search Problem', Proc of the Fourth International Symposium on Computers and Information Science, Miami Beach, 1972.

[4] Palmer, I, 'Scicon Data Base Management', Scientific Control Systems Limited, London, 1973.

[5] Codd, E F, 'Normalized Data Base Structure - A Brief Tutorial', Proc ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Calif, 1971.

[6] Codd, E F and Date, C J, 'The Relational and Network Approaches - Comparison of the Application Programming Interfaces', Proc ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Calif, 1974.

[7] Codd, E F, 'Further Normalization of the Data Base Relational Model', Data Base Systems, Courant Computer Science Symposia, Vol 6, Prentice-Hall, New York, 1971.

[8] Notley, M G, 'The Peterlee IS/1 System', IBM Scientific Centre Report 0018, UK, 1972.

[9] Palmero, F P, 'An APL Implementation of Relational Operators and a Search Algorithm', IBM Research Report RJ 1273, San Jose, Calif, 1973.

[10] Codd, E F, 'Relational Completeness of Data Base Sublanguages', Data Base Systems, Courant Computer Science Symposia, Vol 6, Prentice-Hall, New York, 1971.

[11] DeRemer, F L, 'Generating Parsers for BNF Grammars', Spring Joint Computer Conference, 1969.

[12] Boyce, R F and Chamberlin, D D, 'SEQUEL - A Structured English Query Language', Proc ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, Calif, 1974.

[13] Boyce, R F and Chamberlin, D D, 'Using a Structured English Query Language as a Data Description Facility', IBM Research Report RJ 1318, San Jose, Calif, 1973.

B I B L I O G R A P H Y

1. Codd, E F, 'Seven Steps to Rendezvous with the Casual User', Proc IFIP TC-2 Working Conference on Data Base Management Systems', Cargese, Corsica, April 1974, North-Holland, Amsterdam.
2. Fike, C T, 'PL/1 for Scientific Programmers', IBM Systems Research Institute, Prentice-Hall, Englewood Cliffs, NJ, 1970.
3. Gries, D, 'Compiler Construction for Digital Computers', Wiley, 1971.
4. Hopgood, F R A, 'Compiling Techniques', MacDonald/ Elsevier, 1969.
5. IBM, 'PL/1 (F) Language Reference Manual', IBM Systems Reference Library, 1972.
6. Naur, P (ed), 'Revised Report on the Algorithmic Language ALGOL 60', Comm ACM, Vol 6, pp 1-17, 1963.
7. Page, E S and Wilson, L B, 'Information Representation and Manipulation in a Computer', Cambridge University Press, 1973.

APPENDIX A

SOME TERMINOLOGY ASSOCIATED WITH THE RELATIONAL MODEL OF DATA

PAYROLL (EMPLOY#,	NAME,	AGE,	SALARY,	ANNL_INCRE)
1	Brown	30	1800	0.2
2	Smith	21	1500	0.15
3	Jackson	33	1600	0.15
4	Johnson	29	1700	0.15
5	Wright	38	2200	0.25

Fig. A.1 PAYROLL Relation

Referring to Figure A.1 the following can be defined:

(i) An *object* is a unit of information. Objects may be represented in a computer by character strings, integers or real numbers, e.g., Brown, 2200, 0.2 .

(ii) A *domain* or *set* consists of objects grouped into any meaningful fashion, e.g., the set of names, the set of ages. Each distinct use of a domain in defining a relation is called an *attribute* of that relation.

A *simple domain* is a set all of whose objects are integers, or a set all of whose objects are character strings. A *compound domain* is the expanded cartesian product * of a finite number (say k, k≥1) of simple domains; k being the degree of the compound domain.

Two simple domains are *union-compatible* if both are domains of integers, real numbers or character strings.

---

* For definition, see Appendix B.

Two compound domains A and B are union-compatible if they are of the same degree (say n) and for every j (j = 1, 2, ..., n) the jth simple domain of A is union-compatible with the jth simple domain of B.

A *relationship* is defined as an association between one or several, not necessarily distinct, domains, e.g., 'has a salary of' is a relationship.

(iii) A *tuple* is an ordered set with an object from each attribute such that a relationship exists between the objects, e.g., (1, Brown, 30, 1800, 0.2).

(iv) A *relation*, when declared, is the set of all tuples (elements) of a given relationship, e.g., the data base table PAYROLL.

An *unnormalized* relation is one whose attributes have relations as elements. A *normalized* relation is one whose attributes are simple, i.e., no attribute is itself a relation. A relation defined on simple attributes only is said to be *simple normal*.

Two relations R and S are union-compatible if the attributes of R and S are identical.

(v) The *degree* of a relation is the number of attributes in the relation, e.g., the PAYROLL relation has a degree of 5.

Relations of degree 1 are called *unary*, degree 2 *binary*, degree 3 *ternary* and degree n *n-ary*.

(vi) The *cardinality* of a relation is the number of tuples in the relation, e.g., the cardinality of PAYROLL relation is 5.

(vii) A *candidate key* of a relation is the minimum combination of attributes needed to uniquely identify a tuple of the relation.

(viii) The *primary key* of a relation is one chosen key of the candidate keys of the relation.

APPENDIX B

RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

B.1 Relational Algebra

Of the traditional set operations employed in the relational algebra, the cartesian product yields an expanded product while the operations (union, intersection, difference) are applicable only to pairs of union-compatible normal relations.

Expanded Cartesian Product

Consider two relations A and B. For every $a \epsilon A$, $b \epsilon B$ where $a = (a_1, a_2, \ldots, a_m)$ and $b = (b_1, b_2, \ldots, b_n)$, the concatenation of a and b defined by

$$a^\frown b = (a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n) \quad ,$$

is the set of (m+n)-tuples.

The expanded cartesian product of relation A (degree m) and relation B (degree n) defined by

$$A \circledast B = \{ (a^\frown b) : a \epsilon A \wedge b \epsilon B \} \quad ,$$

yields the relation of degree m+n.

Union

The union of two relations A and B defined by

$$A \cup B = \{x : x \in A \vee x \in B\} ,$$

is the set of tuples which belong to at least one of the relations.

Intersection

The intersection of two relations A and B defined by

$$A \cap B = \{x : x \in A \wedge x \in B\} ,$$

is the set of all tuples that are common to the two relations.

Difference

The difference of two relations A and B defined by

$$A - B = \{x : x \in A \wedge x \notin B\} ,$$

is the set of those tuples of A which do not belong to B.

Projection

A *projection* involves generating a new relation from a given relation by subsetting and reordering attributes.

Suppose r is a tuple of the n-ary relation R. For j = 1, 2, ..., n the notation r[j] designates the jth component of r. For other values of j, r[j] is undefined. The notation can be extended to a list $A = (j_1, j_2, \ldots, j_k)$ of integers (not necessarily distinct) from the set 1, 2, ..., n as follows:

$$r[A] = (r[j_1], r[j_2], \ldots, r[j_k]) \quad .$$

When the list A is empty, r[A] = r. Let R be a relation of degree n, and A a list of integers (not necessarily distinct) from the set 1, 2, ..., n. Then the projection of R on A* is defined by

$$R[A] = \{r[A] : r \in R\} \quad .$$

R (D_1,	D_2,	D_3)
a	6	s
d	4	t
c	2	s
f	3	u
h	4	t

R[3, 2] (D_1,	D_2)
s	6
t	4
s	2
u	3

Fig. B.1 Relation R and One of its Projections

Projection provides an algebraic counterpart to the existential quantifier.

---

* The notation $\Pi_A(R)$ is often used to denote the projection of R on A where Π is a selection operator.

Join

A *join* involves generating a new relation from two given relations by concatenating a subset of the tuples from the first relation to a subset of the tuples from the second relation.

Let θ denote any of the relations $=$, $\neq$, $<$, $\leq$, $>$ and $\geq$. The θ-join of relation S on attribute A with relation T on attribute B is defined by

$$S[A \ \theta \ B]T = \{(s^\frown t) : s \in S \wedge t \in T \wedge (s[A] \ \theta \ t[B])\} \quad ,$$

providing every element of S[A] is θ-comparable with every element of T[B] (s[A] is θ-comparable with t[B] if for every $s \in S$ and $t \in T$, s[A] θ t[B] is either true or false, i.e., not undefined).

S (X, Y, Z)

X	Y	Z
a	2	1
b	3	5
a	5	4
c	3	2

T (P, Q)

P	Q
5	s
4	t

S[Z = P]T (X, Y, Z, P, Q)

X	Y	Z	P	Q
b	3	5	5	s
a	5	4	4	t

Fig. B.2 Relations S, T and One of the Joins

The most commonly used join is the join on =, which is known as the *equi-join*. In the case of the equi-join, two of the attributes of the resulting relation are identical in content. If one of the redundant attributes is removed by projection, the result is the natural join of the given relations.

Division

If R is a binary relation then the *image set* of x under R is defined by

$$g_R(x) = \{y : (x,y) \in R\} \quad .$$

Consider a relation S of degree m divided by a relation T of degree n. Let A and B be two attribute-identifying lists (without repetitions) for S and T respectively, and let $\bar{A}$ denote the attribute-identifying list that is complementary to A and in ascending order. For example, if the degree m of R were 5 and A = (2,5), then $\bar{A}$ = (1,3,4). Providing S[A] and T[B] are union-compatible, the division of S on A by T on B defined by

$$S[A \div B]T = \{s[\bar{A}] : s \in S \wedge T[B] \subseteq g_S(s[\bar{A}])\} \quad ,$$

is the set of values $s[\bar{A}]$ in S such that all tuples in T[B] are contained in the image set $g_S(s[\bar{A}])$. When S is empty, S divided by T is empty, even if T is also empty.

S	(X,	Y,	Z)
	1	14	x
	2	14	y
	3	14	z
	4	12	x
	5	12	y

T	(P,	Q)
	x	2
	y	1
	y	2
	z	3

$$S[Z \div P]T = \phi$$

$$S[Y,Z][Z \div P]T = \{14\}$$

Fig. B.3 Division of S by T

Division provides an algebraic counterpart to the universal quantifier.

Restriction

A *restriction* involves selecting some subset of the tuples from a given relation to form a new relation of the same or lower cardinality.

Suppose R is a relation, and A and B are two attribute-identifying lists for R. Let θ denote any of the relations $=$, $\neq$, $<$, $\leq$, $>$ and $\geq$. The θ-restriction of R on attributes A and B is defined by

$$R[A \;\theta\; B] = \{r : r \in R \wedge (r[A] \;\theta\; r[B])\} \quad ,$$

providing every element of R[A] is θ-comparable with every element of R[B].

R (A, B, C)

A	B	C
p	2	3
r	6	5
q	1	4
q	3	3
r	7	4

R[B > C] (A, B, C)

A	B	C
r	6	5
r	7	4

Fig. B.4 Relation R and One of its Restrictions

B.2 Relational Calculus

Symbols, Terms and Formulae

The symbols of the relational calculus are given in Table B.5 .

Individual Constants	$\alpha, \beta, \ldots$
Attribute Identifiers	a, b, ...
Tuple Variables	$r_1, r_2, \ldots$
Range Predicates (monadic)	$P_1, P_2, \ldots$
Join Predicates (dyadic)	$=, \neq, <, \leq, >, \geq$
Logical Symbols	$\exists, \forall, \wedge, \vee, \neg$
Brackets	[ ] ( ) ,

Table B.5 Symbols of the Relational Calculus

An arbitrarily specified one-to-one correspondence $P_j \leftrightarrow R_j$ ($j = 1, 2, \ldots, N$) is established between the range predicates and the N simple normal relations in the data base, where P_j indicates membership of tuples in relation R_j.

An expression of the form $r_1[b]$ where r_1 is a tuple variable and b an attribute identifier is called an *indexed tuple variable*.

The terms of the relational calculus are of two types, namely range terms and join terms. A *range term* is a monadic predicate followed by a tuple variable, e.g., the range term $P_j r$ indicates that tuple variable r has relation R_j as its range.

A *join term* is either a monadic term (e.g., $r_1[a] = \alpha$) or a dyadic join term (e.g., $r_1[a] < r_2[b]$). The general form of a join term is

$$\lambda \; \Theta \; \mu \qquad \text{or} \qquad \lambda \; \Theta \; \alpha$$

where λ, μ are indexed tuple variables

α is a constant

Θ is one of the join predicate symbols.

The well-formed formulae (WFF) of the relational calculus are defined as follows:

(i) A range term or join term is a WFF.

(ii) If Γ is a WFF so is $\neg\Gamma$.

(iii) If Γ_1 and Γ_2 are WFFs so are $\Gamma_1 \vee \Gamma_2$ and $\Gamma_1 \wedge \Gamma_2$.

(iv) If Γ is a WFF in which r is a free variable* (e.g., $r[a] = 3$) then $\exists r(\Gamma)$ and $\forall r(\Gamma)$ are WFFs with r as a bound variable (e.g., $\exists r(r[a] = 3)$).

(v) No other formulae are WFFs.

A *range WFF* is a quantifier-free WFF, all of whose terms are range terms, e.g., $P_5 r_3 \wedge P_6 r_2$. A *range WFF over R* is a range WFF whose only free variable is r (e.g., $P_1 r \wedge \neg P_2 r$).

A *proper range WFF over r* is a range WFF (e.g., $P_4 r$, $P_4 r \wedge P_1 r$) subject to the following constraints:

(i) The symbol $\neg$ does not appear unless it immediately follows the symbol $\wedge$.

(ii) Whenever the same r_i occurs in two or more range terms, the range predicates (i.e., P's) in those terms must be associated with relations which are union-compatible.

---

* A free variable is one which is not quantified.

Alpha Expression

An *alpha expression* which defines the response relation in the relational calculus is of the form

$$t : w$$

where t is a target list and w a qualification expression.

The target list contains k distinct terms $t_1, t_2, \ldots, t_k$ each consisting of a complete tuple variable (e.g., r_1) or more commonly an indexed tuple variable (e.g., $r_1[a]$). The qualification expression is either a single w component or a logical combination of several w components, $w_1 \oplus w_2 \oplus w_3 \ldots$, over a common set of free variables where $\oplus$ denotes any of the connectives $\vee$, $\wedge$ and $\wedge\neg$.

Each w component has the general form

$$w = U_1 \wedge U_2 \wedge \ldots \wedge U_p \wedge V$$

where p is the number of free variables ($p \geq 1$)

U's are conjunctions of proper range WFFs (U_1 through U_p) over p distinct tuple variables

V is either null (i.e., does not appear) or it is a WFF subject to the following:

(i) every quantifier ($\exists$ or $\forall$) in V is range-coupled, e.g., $\forall P_2 r_2$

(ii) every free variable belongs to the set whose ranges are specified by $U_1, U_2, \ldots, U_p$

(iii) V is devoid of range terms, i.e., there are no terms of the form $P_i r_i$.

The w component is a range-separable WFF if it is a conjunction of the form, $U_1 \wedge U_2 \wedge \ldots \wedge U_p \wedge V$,

satisfying the above notation. One consequence of these requirements is that a range-separable WFF has at least one free variable, e.g., $P_4 r_1 \wedge \exists P_5 r_2 (r_1[b] = r_2[c])$.

The initial step of Codd's reduction algorithm [10] requires that V in the α-expression be in prenex normal form form, i.e., operators (usually $\neg$) do not occur to the left of quantifiers. * In this α-expression V has the form

$$V = Q_{p+1}\ Q_{p+2}\ \cdots\ Q_{p+q}\ (\theta_1 \vee \theta_2 \vee \cdots \vee \theta_K)$$

where q is the number of quantified variables ($q \geq 0$)

Q's are range-coupled quantifiers

each θ_i is a conjunction of join terms over the variables $r_1, r_2, \ldots, r_{p+q}$, e.g.,
$(r_1[a] = r_2[b]) \wedge (r_1[a] < 5)$

If $q=0$ then $V = \theta_1 \vee \theta_2 \vee \cdots \vee \theta_K$.

The part of V following the Q's is called the *matrix*. There should be no negation symbols preceding any join term in the matrix. Wherever a join term using relation θ is immediately preceded by $\neg$, the $\neg$ symbol is eliminated by replacing Θ by its complement (the complements of $=, \neq, <, \leq, >$ and $\geq$ are $\neq, =, \geq, >, \leq$ and $<$ respectively), e.g., $\neg(r_2[a] = 25)$ is equivalent to $(r_2[a] \neq 25)$.

An alphabetic change is systematically applied (if necessary) to the variables in the α-expression so that the free variables $r_1, r_2, \ldots, r_p$ and the bound variables $r_{p+1}, r_{p+2}, \ldots, r_{p+q}$ are numbered in the order of their

---

* $\neg\exists P_1 r_3 (r_2[b] = 6)$ is equivalent to $\forall P_1 r_3 \neg(r_2[b] = 6)$

first occurrence in the qualification expression. *

The following points should be noted:

(i) Tuple variables appearing in the target list t are never quantified.

(ii) Tuple variables appearing in w and not in t must be quantified.

An example of an α-expression is

$$(r_1[a], r_1[c]) : P_1r_1 \wedge \forall P_2r_2((r_1[a] = r_2[b]) \wedge (r_2[c] > 5))$$

where $t = (r_1[a], r_1[c])$ is the target list

$w = P_1r_1 \wedge \forall P_2r_2((r_1[a] = r_2[b]) \wedge (r_2[c] > 5))$ is the qualification expression.

In the qualification expression,

$$U_1 = P_1r_1$$

$$Q_2 = \forall P_2r_2$$

$$\theta_1 = (r_1[a] = r_2[b]) \wedge (r_2[c] > 5)$$

---

* The first occurrence of a bound variable is with its quantifier.

APPENDIX C

LEXEME VALUES FOR SOURCE LANGUAGE SYMBOLS

Symbol	Value
GET	1
DELETE	2
DROP	3
HOLD	4
RELEASE	5
UPDATE	105
DOWN	106
UP	206
PUT	7
NEW	8
READ	9
LIST	109
ALL	110
SOME	210
STOP	11
RANGE	116
AVERAGE	117
COUNT	217
MAX	317
MIN	417
TOTAL	517
IAVERAGE	118
ICOUNT	218
IMAX	318
IMIN	418
ITOTAL	518
AND	119
OR	120
TOP	121
BOTTOM	221
FIXED31	130
FIXED	230
FLOAT16	330
FLOAT	430
CHAR	530
CHARVAR	630
KEY	44

<table>
<tr><th>Symbol</th><th>Value</th></tr>
<tr><td>+ (infix)</td><td>122</td></tr>
<tr><td>- (infix)</td><td>222</td></tr>
<tr><td>/</td><td>123</td></tr>
<tr><td>*</td><td>223</td></tr>
<tr><td><</td><td>124</td></tr>
<tr><td>></td><td>224</td></tr>
<tr><td>=</td><td>324</td></tr>
<tr><td><=</td><td>424</td></tr>
<tr><td>>=</td><td>524</td></tr>
<tr><td>¬=</td><td>624</td></tr>
<tr><td>&</td><td>125</td></tr>
<tr><td>|</td><td>126</td></tr>
<tr><td>:</td><td>127</td></tr>
<tr><td>'</td><td>128</td></tr>
<tr><td>.</td><td>129</td></tr>
<tr><td>(</td><td>131</td></tr>
<tr><td>)</td><td>132</td></tr>
<tr><td>$(</td><td>133</td></tr>
<tr><td>$)</td><td>134</td></tr>
<tr><td><STRING-CONSTANT></td><td>35</td></tr>
<tr><td><INTEGER-CONSTANT></td><td>136</td></tr>
<tr><td><REAL-CONSTANT></td><td>236</td></tr>
<tr><td><IDENTIFIER></td><td>37</td></tr>
<tr><td><WORKSPACE-NAME></td><td>38</td></tr>
<tr><td><DATA-BASE-RELATION-NAME></td><td>39</td></tr>
<tr><td><LOCAL-RELATION-NAME></td><td>40</td></tr>
<tr><td><VARIABLE></td><td>41</td></tr>
<tr><td>± (prefix)</td><td>43</td></tr>
</table>

APPENDIX D

FLOYD PRODUCTION LANGUAGE STATEMENTS

The following abbreviations have been used:

EOS	END-OF-STATEMENT
WSNAME	<WORKSPACE-NAME>
DBRNAME	<DATA-BASE-RELATION-NAME>
IDENT	<IDENTIFIER>
1ELTSTK	FIRST-ELEMENT-ON-STACK
INT	<INTEGER-CONSTANT>
VAR	<VARIABLE>
F-IDENT	<FUNCTION-IDENTIFIER>
LRNAME	<LOCAL-RELATION-NAME>
I-F-IDENT	<I-FUNCTION-IDENTIFIER>
B-F-IDENT	<BOOL-FUNCTION-IDENTIFIER>
REL-OP	<RELATIONAL-OPERATOR>
STR	<STRING-CONSTANT>
REAL	<REAL-CONSTANT>
ATYPE	<ATTR-TYPE>
REL	RELEASE
UD	UPDATE

The names for the production labels used in the Floyd Production Language statements are not meant to convey any meaning as these statements have been optimized and therefore the label names have lost their significance.

State	Token 1	Token 2	Rule	Shift	Flag	Mark	Next	Fail
<START> :	ANY	ANY	R63			*	<SESSION>H	
<SESSION>H :	RANGE	ANY	R7			*	<RANGE>H	
	DROP	ANY	R7			*	T(137)	
	NEW	ANY	R7			*	T(146)	
	STOP	ANY					EXIT	
	EOS	ANY		1->			<START>	
	READ/LIST	ANY	R7				T(162)	FAIL_3(1)
<POST-RANGE>H :	ANY	ANY	R63			*	<POST-RANGE+1>H	
<POST-RANGE+1>H :	GET	ANY	R8			*	<GET>H	
	HOLD	ANY	R8			*	<HOLD>H	
	DELETE	ANY	R8			*	T(122)	
	RANGE	ANY	R60			*	<RANGE>H	
	STOP	ANY					EXIT	
	EOS	ANY		1->			<POST-RANGE>H	FAIL_1(2)
<RANGE>H :	WSNAME	ANY	R2	1->		*	T(18)	
	DBRNAME	ANY	R2	1->		*	T(18)	
	IDENT	ANY	R3	1->		*	T(18)	FAIL_1(3)
T(18) :	IDENT	ANY	R4	1->			<QUANTIFIER>H	FAIL_1(4)
<QUANTIFIER>H :	ANY	SOME/ALL	R5		(1)		T(21)	
	ANY	ANY					T(21)	
T(21) :	ANY	EOS	R62				<STATEMENT>T	FAIL_1(5)
<STATEMENT>T :	RANGE	ANY	R6				<POST-RANGE>H	
	GET	ANY	R18	2->			<POST-GET>H	
	HOLD	ANY	R18				<REL/UD/COMP>H	
	DELETE	ANY	R1	2->			<POST-GET>H	
	IELTSTK	ANY	R1				<POST-GET>H	
	DROP	ANY	R1	1->			<POST-GET>H	
	NEW	ANY	R1	1->			<POST-GET>H	
	PUT	ANY	R1	1->			<POST-GET>H	
<GET>H :	IDENT	ANY		1->		*	<QUOTA>H	
	WSNAME	ANY		1->		*	<QUOTA>H	FAIL_1(6)
<QUOTA>H :	(	INT	R9			**	T(40)	
	(	VAR	R10			**	T(40)	
<GT-LIST>H :	(	ANY				*	<GT-LIST>H	
	F-IDENT	ANY	R13			*	T(41)	
	LRNAME	.	R11			**	T(57)	
	LRNAME	ANY	R11	1->			<GTELEMENT-COM>T	FAIL_1(7)
<GTELEMENT-COM>T :	ANY	,			(1)	*	<GT-ELEMENT>H	
	ANY	ANY					<GTLIST-END>T	
T(40) :	)	ANY		3->		*	<GT-LIST>H	FAIL_1(8)
T(41) :	(	ANY				*	T(42)	FAIL_1(9)
T(42) :	LRNAME	.	R11			**	T(44)	FAIL_1(10)
	ANY	ANY					T(44)	
T(44) :	IDENT	)	R12	5->	(1)		<GTLIST-END>T	FAIL_1(10)
T(45) :	(	ANY				*	T(46)	FAIL_1(11)

```
T(46) :                 LRNAME | ,        R17  1->  | (1)  *  | <ANAME-SEQ1>H      FAIL_1(12)
<ANAME-SEQ1>H :          IDENT | ANY      R14       |         | T(51)
                             ( | ANY                |      *  | <ALIST1>H          FAIL_1(13)
<ALIST1>H :              IDENT | ,        R14  1->  | (1)  *  | <ALIST1>H
                         IDENT | )        R14  1->  | (1)     | T(51)              FAIL_1(13)
T(51) :                    ANY | ,        R15  1->  | (1)  *  | T(52)              FAIL_1(14)
T(52) :                  IDENT | )        R16  3->  | (1)     | <GTLIST-END>T      FAIL_1(12)
<GTLIST-SEQ>T :            ANY | ,                  | (1)  *  | <GTLIST-CONTD>H
<GTLIST-END>T :              ( | ANY                |         | T(56)
                           ANY | ANY                |         | <COLON1>H
T(56) :                    ANY | )             1->  | (1)     | <GTLIST-END>T      FAIL_1(15)
T(57) :                  IDENT | ANY      R12  3->  |         | <GTLIST-SEQ>T      FAIL_1(16)
<GTLIST-CONTD>H :  I-F-IDENT | ANY      R13       |      *  | T(45)
                        LRNAME | .        R11       |     **  | T(57)
                        LRNAME | ANY      R11  1->  |         | <GTELEMENT-COM>T   FAIL_1(7)
<GT-ELEMENT>H :         LRNAME | .        R11       |     **  | T(63)
                        LRNAME | ANY      R11  1->  |         | <GTELEMENT-COM>T   FAIL_1(7)
T(63) :                  IDENT | ANY      R12  3->  |         | <GTELEMENT-COM>T   FAIL_1(16)
<COLON1>H :                ANY | :                  | (1)  *  | <QUAL-EXP>H
<GET-ORDER>H :             ANY | UP/DOWN            |     **  | T(85)
                           ANY | ANY                |         | T(21)
<QUAL-EXP>H :        I-F-IDENT | ANY      R19       |      *  | T(87)
                     B-F-IDENT | ANY      R20       |      *  | T(81)
<QUAL>H :                   $( | ANY                |      *  | <QUAL>H
<QUAL+1>H :                  ( | (                  |      *  | <QUAL+1>H
T(71) :                      ( | ANY           1->  |      *  | T(72)              FAIL_1(17)
T(72) :                 LRNAME | ANY      R21       |      *  | T(73)              FAIL_1(18)
T(73) :                      . | IDENT    R22       |     **  | T(74)              FAIL_1(19)
T(74) :                 REL-OP | ANY                |      *  | <TERM-RHS>H        FAIL_1(20)
<TERM-RHS>H :           LRNAME | ANY      R21       |      *  | T(79)
                           STR | )        R23  5->  | (1)     | <&/|>T
                           +/- | ANY      R56       |      *  | T(78)
T(78) :               INT/REAL | ANY      R57  5->  | (1)     | <&/|>T             FAIL_1(21)
T(79) :                      . | IDENT              |      *  | T(80)              FAIL_1(19)
T(80) :                    ANY | )        R24  7->  | (1)     | <&/|>T             FAIL_1(22)
T(81) :                      ( | INT/REAL           |     **  | T(82)              FAIL_1(23)
T(82) :                      , | LRNAME   R21       |     **  | T(83)              FAIL_1(23)
T(83) :                      . | IDENT              |      *  | T(84)              FAIL_1(23)
T(84) :                    ANY | )        R33  7->  | (1)     | <GET-ORDER>H       FAIL_1(23)
T(85) :                 LRNAME | .                  |      *  | T(86)              FAIL_1(24)
T(86) :                    ANY | IDENT    R34  3->  | (1)     | <GET-ORDER>H       FAIL_1(24)
T(87) :                      ( | ANY      R35       |      *  | T(88)              FAIL_1(11)
T(88) :                 LRNAME | ,        R21  1->  | (1)  *  | <ANAME-SEQ2>H      FAIL_1(12)
<ANAME-SEQ2>H :          IDENT | ANY      R32       |         | T(93)
                             ( | ANY                |      *  | <ALIST2>H          FAIL_1(13)
```

```
<ALIST2>H :            IDENT | ,        R32  1->  | (1)  *  | <ALIST2>H
                       IDENT | )        R32  1->  | (1)     | T(93)              FAIL_1(13)
T(93) :                  ANY | ,        R25  1->  | (1)  *  | T(94)              FAIL_1(14)
T(94) :                IDENT | )        R26       | (1)  *  | T(95)              FAIL_1(12)
T(95) :                  ANY | ANY                |         | T(211)
<&/|>T :                 ANY | &                  | (1)  *  | T(71)
<QUALEXP-END>T :           ( | ANY                |         | T(106)
                         ANY | |        R28       | (1)  *  | <QUAL+1>H
<AND/OR>T :              AND | ANY      R30  1->  |         | <AND/OR+1>T
<AND/OR+1>T :            ANY | AND      R29       |      ** | <QUAL>H
                          OR | ANY      R30  1->  |         | <AND/OR+3>T
<AND/OR+3>T :            ANY | OR       R29       |      ** | <QUAL>H
                          $( | ANY                |         | T(107)
                      DELETE | ANY                |         | T(21)
                         ANY | ANY                |         | <GET-ORDER>H
T(106) :                 ANY | )             1->  | (1)     | <QUALEXP-END>T     FAIL_1(15)
T(107) :                 ANY | $)            1->  | (1)     | <AND/OR+9>T        FAIL_1(27)
<AND/OR+9>T :             $( | ANY                |         | T(107)
                         ANY | ANY                |         | <AND/OR>T
<HOLD>H :              IDENT | ANY           1->  |      *  | <HT-LIST>H
                      WSNAME | ANY           1->  |      *  | <HT-LIST>H         FAIL_1(28)
<HT-LIST>H :               ( | ANY                |      *  | <HT-LIST>H
                      LRNAME | .        R11       |      ** | T(115)
                      LRNAME | ANY      R11  1->  |         | <HTLIST-END>T      FAIL_1(7)
T(115) :               IDENT | ANY      R36  3->  |         | <HTELEMENT-COM>T   FAIL_1(16)
<HTELEMENT-COM>T :       ANY | ,                  | (1)  *  | T(121)
                         ANY | ANY                |         | <HTLIST-END>T
<HTLIST-END>T :            ( | ANY                |         | T(120)
                         ANY | ANY                |         | <COLON2>H
T(120) :                 ANY | )             1->  | (1)     | <HTLIST-END>T      FAIL_1(15)
T(121) :              LRNAME | .        R11       |      ** | T(115)             FAIL_1(7)
T(122) :              LRNAME | ANY           1->  |         | <COLON2>H          FAIL_1(29)
<COLON2>H :              ANY | :                  | (1)  *  | <QUAL-EXP>H
                         ANY | ANY                |         | T(21)
T(125) :              WSNAME | ANY           1->  |         | <PUT>H             FAIL_1(30)
<PUT>H :                 ANY | IDENT    R37       | (1)     | <DOT1>H
                         ANY | WSNAME   R37       | (1)     | <DOT1>H            FAIL_1(31)
<DOT1>H :                ANY | .                  | (1)  *  | <ANAME-SEQ3>H
                         ANY | ANY                |         | <PUT-ORDER>H
<ANAME-SEQ3>H :        IDENT | ANY      R38  1->  |         | <PUT-ORDER>H
                           ( | ANY           1->  |      *  | <ANAME-LIST3>H     FAIL_1(32)
<ANAME-LIST3>H :       IDENT | ,        R38  1->  | (1)  *  | <ANAME-LIST3>H
                       IDENT | )        R38  1->  | (1)     | <PUT-ORDER>H       FAIL_1(32)
<PUT-ORDER>H :           ANY | UP/DOWN            |      *  | T(136)
                         ANY | ANY                |         | T(21)
```

T(136) :	ANY	IDENT	R39	1->	(1)		<PUT-ORDER>H	FAIL_1(33)
T(137) :	IDENT	ANY	R31	1->			<DOT2>H	FAIL_1(34)
	ANY	ANY					<DOT2>H	
<DOT2>H :	ANY	.			(1)	*	<ANAME-SEQ4>H	
	ANY	ANY					T(21)	
<ANAME-SEQ4>H :	IDENT	ANY	R38	1->			T(21)	
	(	ANY		1->		*	T(143)	FAIL_1(32)
T(143) :	IDENT	ANY	R38	1->			<ANAME-LIST4>H	FAIL_1(32)
<ANAME-LIST4>H :	ANY	,			(1)	*	T(143)	
	ANY	)			(1)		T(21)	FAIL_1(32)
T(146) :	IDENT	ANY	R31	1->			T(147)	FAIL_1(35)
T(147) :	ANY	(				**	T(150)	FAIL_1(36)
<AFIELD>H :	ANY	(				**	T(150)	
	ANY	ANY					T(21)	
T(150) :	IDENT	,	R40	1->	(1)	*	<AFIELD+3>H	FAIL_1(37)
<AFIELD+3>H :	KEY	,	R51	1->	(1)	*	T(152)	
T(152) :	ATYPE	ANY	R41	1->			<AFIELD+5>H	FAIL_1(37)
<AFIELD+5>H :	ANY	,			(1)	*	T(155)	
	ANY	)		1->	(1)		<AFIELD>H	FAIL_1(37)
T(155) :	INT/REAL	)	R42	2->	(1)		<AFIELD>H	FAIL_1(37)
<REL/UD/COMP>H :	ANY	ANY	R63			*	<REL/UD/COMP+1>H	
<REL/UD/COMP+1>H :	REL/UD	ANY	R7				T(209)	
	EOS	ANY		1->			<REL/UD/COMP>H	
	STOP	ANY					EXIT	
	WSNAME	ANY	R7			*	T(168)	
	READ/LIST	ANY	R7				T(162)	FAIL_2(38)
T(162) :	ANY	(			(1)	*	<IDENT-LIST>H	FAIL_2(39)
<IDENT-LIST>H :	VAR	ANY	R44	1->			<IDENT-LIST+3>H	
	WSNAME	ANY	R44	1->			<IDENT-LIST+3>H	
	IDENT	ANY	R44	1->			<IDENT-LIST+3>H	FAIL_2(40)
<IDENT-LIST+3>H :	ANY	,			(1)	*	<IDENT-LIST>H	
	ANY	)			(1)		T(210)	FAIL_2(40)
T(168) :	.	IDENT	R46	1->	(1)		T(169)	FAIL_2(41)
T(169) :	ANY	=	R43			**	<RIGHT-PART>H	FAIL_2(42)
<ARITH-EXP>H :	+/-	ANY	R56			*	<ARITH-EXP+1>H	
<ARITH-EXP+1>H :	(	ANY				*	<ARITH-EXP>H	
	INT/REAL	ANY	R47	1->			<EXP>T	
	VAR	ANY	R48	1->			<EXP>T	
	WSNAME	ANY		1->		*	T(175)	FAIL_2(43)
T(175) :	.	IDENT	R46	1->	(1)		<EXP>T	FAIL_2(41)
<EXP>T :	PREFIX(-)	ANY	R49	1->			<EXP+1>T	
<EXP+1>T :	*//	ANY	R50	1->			<EXP+2>T	
<EXP+2>T :	ANY	*//				**	<ARITH-EXP+1>H	
	+/-	ANY	R50	1->			<EXP+4>T	
<EXP+4>T :	ANY	+/-				**	<ARITH-EXP+1>H	

	(	ANY					T(183)	
<EXP+6>T :	=	ANY	R50	1->			T(210)	
T(183) :	ANY	)		1->	(1)		<EXP>T	FAIL_2(15)
<FIX_1>T :	RANGE	ANY	R58				<POST-RANGE>H	
	DROP	ANY		1->			<FIX_1+5>T	
	NEW	ANY		1->			<FIX_1+5>T	
	PUT	ANY		1->			<FIX_1+5>T	
	ANY	ANY		1->			<FIX_1>T	
<FIX_1+5>T :	RANGE	ANY					<POST-RANGE>H	
	ANY	ANY					<POST-GET>H	
<FAIL_1>T :	ANY	EOS					<FIX_1>T	
	ANY	ANY			(1)		<FAIL_1>T	
<FIX_2>T :	HOLD	ANY	R59				<REL/UD/COMP>H	
	1ELTSTK	ANY					<POST-GET>H	
	ANY	ANY		1->			<FIX_2>T	
<FAIL_2>T :	ANY	EOS					<FIX_2>T	
	ANY	ANY			(1)		<FAIL_2>T	
<FIX_3> :	1ELTSTK	ANY	R59				<POST-GET>H	
	ANY	ANY		1->			<FIX_3>T	
<FAIL_3>T :	ANY	EOS					<FIX_3>T	
	ANY	ANY			(1)		<FAIL_3>T	
<FAIL_4>T :	RANGE	ANY	R58				<POST-RANGE>H	
	GET	ANY		1->			<POST-RANGE>H	
	HOLD	ANY		1->			<POST-RANGE>H	
	DELETE	ANY		1->			<POST-RANGE>H	
	ANY	ANY		1->			<POST-GET>H	
<FAIL_5>T :	HOLD	ANY					<REL/UD/COMP>H	
	1ELTSTK	ANY					<POST-GET>H	
T(209) :	ANY	EOS	R1	3->			<POST-GET>H	FAIL_2(5)
T(210) :	ANY	EOS	R61	1->			<STATEMENT>T	FAIL_2(5)
T(211) :	REL-OP	ANY				*	<NUMBER>H	FAIL_1(25)
<NUMBER>H :	+/-	ANY	R56			*	T(213)	
T(213) :	INT/REAL	ANY	R57	5->			<GET-ORDER>H	FAIL_1(26)
<POST-GET>H :	ANY	ANY	R63			*	<POST-GET+1>H	
<POST-GET+1>H :	PUT	ANY	R7			*	T(125)	
	WSNAME	ANY	R7			*	T(168)	
	EOS	ANY		1->			<POST-GET>H	
	ANY	ANY					<SESSION>H	
<RIGHT-PART>H :	STR	ANY	R47	1->			<EXP+6>T	
	ANY	ANY					<ARITH-EXP>H	

A P P E N D I X   E

IMPLEMENTATION OF THE TRANSLATOR

The translator has been implemented on the Northumbrian Universities' Multiple Access Computer (NUMAC) IBM 360/67, owned jointly by the Universities of Durham and Newcastle-upon-Tyne, and Newcastle-upon-Tyne Polytechnic. NUMAC provides both interactive and batch processing facilities and has 1024K bytes of main memory.

The translator is invoked in terminal mode by the MTS * command

£SOURCE CLN6:ALPHA

and in batch mode by the command

£SOURCE CLN6:B.ALPHA .

In batch mode the coding tables 'generated' by the translator are directed to *sink*.

---

* MTS, the Michigan Terminal System, is the major operating system in use in NUMAC.

A P P E N D I X   F

EXAMPLES OF TABLES PRODUCED

BY THE TRANSLATOR

```
RANGE MATERIAL M

GET W (M.P#)
```

SYMBOL TABLE

```
1. MATERIAL          39     0
2. M                 40     0
3. W                 37     0
4. P#                37     0
```

WORKSPACE TABLE

```
21. W
```

CODING TABLES

```
RA_TAB
     RNAME    LRNAME    QUANT    WCOMP
       1        2         0        0

ST_TAB        1

WN_TAB        21

Q_TAB         0

GT_TAB
     FUNCT    ALIST    ALISTPTR    LRNAME    ANAME
       0        0         0          2         4

FN_MT_TAB

DT_TAB

DTOV_TAB

WP_TAB

OR_TAB
```

```
RANGF MATERIAL M

GET W (M.P#, M.PNAME, M.COH) : (M.COH < 100)

SYMBOL TABLE
------------
 1. MATFRIAL                       39        0
 2. M                              40        0
 3. W                              37        0
 4. P#                             37        0
 5. PNAME                          37        0
 6. QOH                            37        0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
     1.        100

CODING TABLES
-------------
 RA_TAB
      RNAME     LRNAME     QUANT     WCOMP
        1          2         0         1

 ST_TAB         1

 WN_TAB        21

 Q_TAB          0

 GT_TAB
      FUNCT     ALIST     ALISTPTR     LRNAME     ANAME
        0         0          0           2          4
        0         0          0           2          5
        0         0          0           2          6

 FN_MT_TAB
      FUNCT   ALIST   ALISTPTR   RNAME   LRNAME   ANAME   RELOP   WCOMP   TCOMP   CONTYPE   CONLEN   CONPTR
        0       0        0         1       2        6      124      1       1        1         0        1

 DT_TAB

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE MATERIAL M

GET W (6) (M.P#, M.PNAME, M.QOH) : (M.QOH <100)  UP M.P#  DOWN M.WEIGHT

SYMBOL TABLE
------------
  1. MATERIAL                    39         0
  2. M                           40         0
  3. W                           37         0
  4. P#                          37         0
  5. PNAME                       37         0
  6. QOH                         37         0
  7. WEIGHT                      37         0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
    1.        100

CODING TABLES
-------------
 RA_TAB
      RNAME     LRNAME     QUANT     WCOMP
        1          2          0         1

 ST_TAB          1

 WN_TAB          21

 Q_TAB           6

 GT_TAB
      FUNCT     ALIST     ALISTPTR     LRNAME     ANAME
        0         0          0            2         4
        0         0          0            2         5
        0         0          0            2         6

 FN_MT_TAB
      FUNCT   ALIST   ALISTPTR   RNAME   LRNAME   ANAME   RELOP   WCOMP   TCOMP   CONTYPE   CONLEN   CONPTR
        0       0        0         1        2        6      124      1       1        1         0        1

 DT_TAB

 DTOV_TAB

 WP_TAB

 OR_TAB
      ORDER     LRNAME     ANAME
       206         2          4
       106         2          7
```

```
RANGE FACTORY F

RANGE ORDER O SOME

GET W (F.SNAME, F.ADDRESS) : (F.S# = O.S#) & (O.P# = 5)

SYMBOL TABLE
------------
  1. FACTORY                  39          0
  2. F                        40          0
  3. ORDER                    39          0
  4. O                        40          2
  5. W                        37          0
  6. SNAME                    37          0
  7. ADDRESS                  37          0
  8. S#                       37          0
  9. P#                       37          0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
    1.          5

CODING TABLES
-------------
 RA_TAB
      RNAME    LRNAME    QUANT    WCOMP
        1        2         0        1
        3        4         2        1

 ST_TAB        1

 WN_TAB        21

 Q_TAB         0

 GT_TAB
      FUNCT    ALIST    ALISTPTR    LRNAME    ANAME
        0        0         0          2         6
        0        0         0          2         7

 FN_MT_TAB
      FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME  RELOP  WCOMP  TCOMP  CONTYPE  CONLEN  CONPTR
        0      0       0        3       4      9     324     1      1       1        0       1

 DT_TAB
      LRNAME1   ANAME1   LRNAME2   ANAME2   RELOP   WCOMP   TCOMP
        2         8         4         8      324      1       1

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE FACTORY F

RANGE FACTORY E SOME

GET W (F.S#) : (E.SNAME = 'IRONWORKS') & (F.ADDRESS = E.ADDRESS)

SYMBOL TABLE
------------
  1. FACTORY              39        0
  2. F                    40        0
  3. E                    40        2
  4. W                    37        0
  5. S#                   37        0
  6. SNAME                37        0
  7. ADDRESS              37        0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 CHARS
    1. IRONWORKS

CODING TABLES
-------------
 RA_TAB
      RNAME    LRNAME    QUANT    WCOMP
        1        2         0        1
        1        3         2        1

 ST_TAB          1

 WN_TAB          21

 Q_TAB           0

 CT_TAB
      FUNCT    ALIST    ALISTPTR    LRNAME    ANAME
        0        0         0          2         5

 FN_MT_TAB
      FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME  RELOP  WCOMP  TCOMP  CONTYPE  CONLEN  CCNPTR
        0      0       0        1       3       6     324     1      1       5        9       1

 DT_TAB
      LRNAME1   ANAME1   LRNAME2   ANAME2   RELOP   WCOMP   TCOMP
        2         7         3         7      324      1       1

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE FACTORY F

RANGE ORDER O ALL

GET W (F.S#) : (F.S# ¬=O.S#) | (O.P# ¬=3)
```

SYMBOL TABLE

1. FACTORY	39	0
2. F	40	0
3. ORDER	39	0
4. O	40	1
5. W	37	0
6. S#	37	0
7. P#	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

F831

1.	3

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1
3	4	1	1

ST_TAB 1

WN_TAB 21

Q_TAB 0

GT_TAB

FUNCT	ALIST	ALISTPTR	LRNAME	ANAME
0	0	0	2	6

FN_MT_TAB

FUNCT	ALIST	ALISTPTR	RNAME	LRNAME	ANAME	RELOP	WCOMP	TCOMP	CONTYPE	CONLEN	CONPTR
0	0	0	3	4	7	624	1	2	1	0	1

DT_TAB

LRNAME1	ANAME1	LRNAME2	ANAME2	RELOP	WCOMP	TCOMP
2	6	4	6	624	1	1

DTOV_TAB

WP_TAB

OR_TAB

```
RANGE FACTORY F

RANGE TASK T ALL

RANGE ORDER O SCME

GET W (F.SNAME) : (F.S# = O.S#) & (O.J# = T.J#)
```

SYMBOL TABLE

1. FACTORY	39	0
2. F	40	0
3. TASK	39	0
4. T	40	1
5. ORDER	39	0
6. O	40	2
7. W	37	0
8. SNAME	37	0
9. S#	37	0
10. J#	37	0

WORKSPACE TABLE

21. W

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1
3	4	1	1
5	6	2	1

ST_TAB 1

WN_TAB 21

Q_TAB 0

GT_TAB

FUNCT	ALIST	ALISTPTR	LRNAME	ANAME
0	0	0	2	8

FN_MT_TAB

DT_TAB

LRNAME1	ANAME1	LRNAME2	ANAME2	RELOP	WCOMP	TCOMP
2	9	6	9	324	1	1
6	10	4	10	324	1	1

DTOV_TAB

WP_TAB

OR_TAB

```
RANGE TASK T
RANGE FACTORY F
RANGE ORDER O SOME
GET W (T.J#, T.JNAME, F.ADDRESS) : (T.J# = O.J#) & (O.S# = F.S#)
```

SYMBOL TABLE

1. TASK	39	0
2. T	40	0
3. FACTORY	39	0
4. F	40	0
5. ORDER	39	0
6. O	40	2
7. W	37	0
8. J#	37	0
9. JNAME	37	0
10. ADDRESS	37	0
11. S#	37	0

WORKSPACE TABLE

21. W

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1
3	4	0	1
5	6	2	1

ST_TAB 1

WN_TAB 21

Q_TAB 0

GT_TAB

FUNCT	ALIST	ALISTPTR	LRNAME	ANAME
0	0	0	2	8
0	0	0	2	9
0	0	0	4	10

FN_MT_TAB

DT_TAB

LRNAME1	ANAME1	LRNAME2	ANAME2	RELOP	WCOMP	TCOMP
2	8	6	8	324	1	1
6	11	4	11	324	1	1

DTOV_TAB

WP_TAB

OR_TAB

```
RANGE ORDER O

RANGE FACTORY F SOME

GET W1 (O.J#) : (O.S# = F.S#) & (F.SNAME = 'PAINTWORKS')

SYMBOL TABLE
------------
 1. ORDER                   39        0
 2. O                       40        0
 3. FACTORY                 39        0
 4. F                       40        2
 5. W1                      37        0
 6. J#                      37        0
 7. S#                      37        0
 8. SNAME                   37        0

WORKSPACE TABLE
---------------
 21. W1

CONSTANT TABLE
--------------
 CHARS
     1. PAINTWORKS

CODING TABLES
-------------
 RA_TAB
      RNAME    LRNAME    QUANT    WCOMP
        1        2         0        1
        3        4         2        1

 ST_TAB       1

 WN_TAB       21

 Q_TAB        0

 GT_TAB
      FUNCT    ALIST    ALISTPTR    LRNAME    ANAME
        0        0         0          2         6

 FN_MT_TAB
      FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME  RELOP  WCOMP  TCOMP  CONTYPE  CONLEN  CONPTR
        0      0       0        3      4       8     324     1      1       5       10       1

 OT_TAB
      LRNAME1   ANAME1   LRNAME2   ANAME2   RELOP   WCOMP   TCOMP
         2         7        4         7      324      1       1

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE FACTORY F
RANGE W1 X ALL
RANGE ORDER O SOME
GET W2 (F.SNAME, F.ADDRESS) : (F.S# = O.S#) & (O.J# = X.J#)
```

SYMBOL TABLE

1. FACTORY	39	0
2. F	40	0
3. X	40	1
4. ORDER	39	0
5. O	40	2
6. W2	37	0
7. SNAME	37	0
8. ADDRESS	37	0
9. S#	37	0
10. J#	37	0

WORKSPACE TABLE

21. W1
22. W2

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1
21	3	1	1
4	5	2	1

ST_TAB 1

WN_TAB 22

Q_TAB 0

GT_TAB

FUNCT	ALIST	ALISTPTR	LRNAME	ANAME
0	0	0	2	7
0	0	0	2	8

FN_MT_TAB

DT_TAB

LRNAME1	ANAME1	LRNAME2	ANAME2	RELOP	WCOMP	TCOMP
2	9	5	9	324	1	1
5	10	3	10	324	1	1

DTOV_TAB

WP_TAB

OR_TAB

```
RANGE ORDER O

GET W (O.S#) : (O.J# = 25) OR (O.J# = 47) & (O.P# = 7)
```

SYMBOL TABLE

1. ORDER	39	0
2. O	40	0
3. W	37	0
4. S#	37	0
5. J#	37	0
6. P#	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

FB31

1.	25
2.	47
3.	7

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1
1	2	0	2

ST_TAB 1

WN_TAB 21

Q_TAB 0

GT_TAB

FUNCT	ALIST	ALISTPTR	LRNAME	ANAME
0	0	0	2	4

FN_MT_TAB

FUNCT	ALIST	ALISTPTR	RNAME	LRNAME	ANAME	RELOP	WCOMP	TCOMP	CONTYPE	CONLEN	CONPTR
0	0	0	1	2	5	324	1	1	1	0	1
0	0	0	1	2	5	324	2	1	1	0	2
0	0	0	1	2	6	324	2	1	1	0	3

DT_TAB

DTOV_TAB

WP_TAB

WCOMP1	WCOMP2	OPER
1	2	120

OR_TAB

```
RANGE FACTORY F

RANGE CRDER O ALL

GET W COUNT(F.S#) : (F.S# ¬= O.S#) | (O.P# ¬= 5)

SYMBOL TABLE
------------
 1. FACTORY                    39        0
 2. F                          40        0
 3. ORCER                      39        0
 4. O                          40        1
 5. W                          37        0
 6. S#                         37        0
 7. P#                         37        0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 F831
    1.        5

CODING TABLES
-------------
 RA_TAB
      RNAME     LRNAME      QUANT      WCOMP
        1          2          0          1
        3          4          1          1

 ST_TAB         1

 WN_TAB         21

 Q_TAB          0

 GT_TAB
      FUNCT     ALIST     ALISTPTR    LRNAME     ANAME
      217         0          0          2          6

 FN_MT_TAB
      FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME   RELOP   WCOMP   TCOMP  CONTYPE  CONLEN  CCNPTR
        0      0       0        3      4       7      624      1       2       1        0       1

 DT_TAB
      LRNAME1   ANAME1   LRNAME2   ANAME2   RELOP    WCOMP    TCOMP
         2         6        4         6      624       1        1

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE TASK T

GET W (T.JNAME) : TOP(4,T.PRIORITY)

SYMBOL TABLE
------------
  1. TASK                          39          0
  2. T                             40          0
  3. W                             37          0
  4. JNAME                         37          0
  5. PRIORITY                      37          0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
     1.          4

CODING TABLES
-------------
 RA_TAB
      RNAME    LRNAME     QUANT      WCOMP
        1        2          0          0

 ST_TAB          1


 WN_TAB          21


 Q_TAB           0


 GT_TAB
      FUNCT    ALIST    ALISTPTR   LRNAME    ANAME
        0        0         0         2         4

 FN_MT_TAB
      FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME   RELOP    WCOMP    TCOMP  CONTYPE  CONLEN  CCNPTR
       121     0       0        1       2      5       0        0        0       1        0       1

 DT_TAB


 DTOV_TAB


 WP_TAB


 OR_TAB
```

```
RANGE ORDER O

GET W (O.P#, O.J#, (TOTAL(O, (P#,J#), QUANTITY) )

SYMBOL TABLE
------------
  1. ORDER                            39          0
  2. O                                40          0
  3. W                                37          0
  4. P#                               37          0
  5. J#                               37          0
  6. QUANTITY                         37          0

WORKSPACE TABLE
---------------
 21. W

CODING TABLES
-------------
 RA_TAB
        RNAME     LRNAME     QUANT     WCOMP
          1          2         0         0

 ST_TAB          1

 WN_TAB          21

 Q_TAB           0

 GT_TAB
        FUNCT     ALIST     ALISTPTR     LRNAME     ANAME
          0         0          0           2          4
          0         0          0           2          5
        518         4          4           2          6
          0         5          0           0          0

 FN_HT_TAB

 DT_TAB

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE ORDER O

GET W (O.P#, O.J#, ICOUNT(O. (P#,J#), S#) )

SYMBOL TABLE
------------
 1. ORDER                     39        0
 2. O                         40        0
 3. W                         37        0
 4. P#                        37        0
 5. J#                        37        0
 6. S#                        37        0

WORKSPACE TABLE
---------------
 21. W

CODING TABLES
-------------
 RA_TAB
      RNAME    LRNAME     QUANT     WCOMP
        1        2          0         0

 ST_TAB          1


 WN_TAB          21


 Q_TAB           0


 GT_TAB
      FUNCT    ALIST    ALISTPTR   LRNAME    ANAME
        0        0         0         2         4
        0        0         0         2         5
      218        4         4         2         6
        0        5         0         0         0

 FN_HT_TAB


 DT_TAB


 DTOV_TAB


 WP_TAB


 OR_TAB
```

```
RANGE CRDER O

GET W (O.P#) : ICOUNT(O, P#, J#) > 3

SYMBOL TABLE
------------
  1. ORDER                     39        0
  2. O                         40        0
  3. W                         37        0
  4. P#                        37        0
  5. J#                        37        0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
    1.        3

CODING TABLES
-------------
 RA_TAB
     RNAME    LRNAME    QUANT    WCOMP
       1        2         0        0

 ST_TAB        1

 WN_TAB        21

 Q_TAB         0

 GT_TAB
     FUNCT    ALIST    ALISTPTR    LRNAME    ANAME
       0        0         0          2         4

 FN_MT_TAB
     FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME  RELOP  WCOMP  TCOMP  CONTYPE  CONLEN  CONPTR
     218      4       0        1      2       5     224     0      0       1        0       1

 DT_TAB

 DTOV_TAB

 WP_TAB

 OR_TAB
```

```
RANGE TASK T

HOLD W (T.JNAME) : (T.J# = 20)
```

SYMBOL TABLE

1. TASK	39	0
2. T	40	0
3. W	37	0
4. JNAME	37	0
5. J#	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

FB31
1. 20

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1

ST_TAB 4

WN_TAB 21

HT_TAB

LRNAME	ANAME
2	4

FN_HT_TAB

FUNCT	ALIST	ALISTPTR	RNAME	LRNAME	ANAME	RELOP	WCOMP	TCOMP	CONTYPE	CONLEN	CONPTR
0	0	0	1	2	5	324	1	1	1	0	1

DT_TAB

DTOV_TAB

WP_TAB

OR_TAB

```
W.JNAME = 'CAM'
```

SYMBOL TABLE

1. JNAME	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

CHARS
1. CAM

CODING TABLES

ST_TAB 38

ASS_TAB

OP1	OP2	OPER	IOP1	IOP2
21	1	129	2	2
-1	1	324	4	5

```
UPDATE
```

WORKSPACE TABLE

21. W

CODING TABLES

ST_TAB 105

```
 RANGE TASK T

 GET W (T) : (T.J# = 6)

SYMBOL TABLE
------------
 1. TASK                         39         0
 2. T                            40         0
 3. W                            37         0
 4. J#                           37         0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
     1.          6

CODING TABLES
-------------
 RA_TAB
      RNAME     LRNAME     QUANT     WCOMP
        1         2          0         1

 ST_TAB          1

 WN_TAB          21

 Q_TAB           0

 GT_TAB
      FUNCT     ALIST     ALISTPTR     LRNAME     ANAME
        0         0          0           2          0

 FN_MT_TAB
      FUNCT   ALIST   ALISTPTR   RNAME   LRNAME   ANAME   RELOP   WCOMP   TCOMP   CONTYPE   CONLEN   CONPTR
        0       0        0         1        2       4      324      1       1        1        0        1

 DT_TAB

 DTOV_TAB

 WP_TAB

 OR_TAB


 RANGE TASK T

 DELETE T : (T.J# = 6)

SYMBOL TABLE
------------
 1. TASK                         39         0
 2. T                            40         0
 3. J#                           37         0

WORKSPACE TABLE
---------------
 21. W

CONSTANT TABLE
--------------
 FB31
     1.          6

CODING TABLES
-------------
 RA_TAB
      RNAME     LRNAME     QUANT     WCOMP
        1         2          0         1

 ST_TAB          2

 LRN_TAB         2

 FN_MT_TAB
      FUNCT   ALIST   ALISTPTR   RNAME   LRNAME   ANAME   RELOP   WCOMP   TCOMP   CONTYPE   CONLEN   CCNPTR
        0       0        0         1        2       3      324      1       1        1        0        1

 DT_TAB

 DTOV_TAB

 WP_TAB
```

W.J# = 7

SYMBOL TABLE

1. J# 37 0

WORKSPACE TABLE

21. W

CONSTANT TABLE

FB31
1. 7

CODING TABLES

ST_TAB 38

ASS_TAB

OP1	OP2	OPER	IOP1	IOP2
21	1	129	2	2
-1	1	324	4	1

PUT W TASK

SYMBOL TABLE

1. TASK 37 0

WORKSPACE TABLE

21. W

CODING TABLES

ST_TAB 7

WN_TAB 21

RN_TAB 1

AL_TAB

OR_TAB

RANGE ORDER O

HOLD W (O.QUANTITY) : (O.P# = 4) & (O.J# = 5) & (O.DATEDUE = '1.4.74')

SYMBOL TABLE
----

1. ORDER	39	0
2. O	40	0
3. W	37	0
4. QUANTITY	37	0
5. P#	37	0
6. J#	37	0
7. DATEDUE	37	0

WORKSPACE TABLE
----

21. W

CONSTANT TABLE

FB31

1.	4
2.	5

CHARS

1. 1.4.74

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1

ST_TAB 4

WN_TAB 21

HT_TAB

LRNAME	ANAME
2	4

FN_MT_TAB

FUNCT	ALIST	ALISTPTR	RNAME	LRNAME	ANAME	RELOP	WCOMP	TCOMP	CONTYPE	CONLEN	CONPTR
0	0	0	1	2	5	324	1	1	1	0	1
0	0	0	1	2	6	324	1	1	1	0	2
0	0	0	1	2	7	324	1	1	5	6	1

DT_TAB

DTOV_TAB

WP_TAB

OR_TAB

W.QUANTITY = W.QUANTITY - 40

SYMBOL TABLE
----

1. QUANTITY	37	0

WORKSPACE TABLE
----

21. W

CONSTANT TABLE

FB31

1.	40

CODING TABLES

ST_TAB 38

ASS_TAB

OP1	OP2	OPER	IOP1	IOP2
21	1	129	2	2
21	1	129	2	2
-2	1	222	4	1
-1	-3	324	4	4

UPDATE

WORKSPACE TABLE
----

21. W

CODING TABLES

ST_TAB 105

RANGE MATERIAL M

HOLD W (M.QOH) : (M.P# = 4)

SYMBOL TABLE

1. MATERIAL	39	0
2. M	40	0
3. QOH	37	0
4. P#	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

FB31
1. 4

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	1

ST_TAB 4

WN_TAB 21

HT_TAB

LRNAME	ANAME
2	3

FN_MT_TAB

FUNCT	ALIST	ALISTPTR	RNAME	LRNAME	ANAME	RELOP	WCOMP	TCOMP	CONTYPE	CONLEN	CONPTR
0	0	0	1	2	4	324	1	1	1	0	1

DT_TAB

DTDV_TAB

WP_TAB

OR_TAB

W.QOH = W.QOH + 40

SYMBOL TABLE

1. QOH	37	0

WORKSPACE TABLE

21. W

CONSTANT TABLE

FB31
1. 40

CODING TABLES

ST_TAB 38

ASS_TAB

OP1	OP2	OPER	IOP1	IOP2
21	1	129	2	2
21	1	129	2	2
-2	1	122	4	1
-1	-3	324	4	4

UPDATE

WORKSPACE TABLE

21. W

CODING TABLES

ST_TAB 105

```
RANGE TASK T

DELETE T
```

SYMBOL TABLE

1. TASK	39	0
2. T	40	0

CODING TABLES

RA_TAB

RNAME	LRNAME	QUANT	WCOMP
1	2	0	0

ST_TAB 2

LRN_TAB 2

FN_NT_TAB

DT_TAB

DTOV_TAB

WP_TAB

```
RANGE ORDER O

RANGE FACTORY F SOME

RANGE TASK T SOME

DELETE O : (F.SNAME = 'STEELWORKS') & (O.S# = F.S#) & (O.J# = T.J#) & (T.JNAME = 'CAG')

SYMBOL TABLE
------------
  1. ORDER                 39        0
  2. O                     40        0
  3. FACTORY               39        0
  4. F                     40        2
  5. TASK                  39        0
  6. T                     40        2
  7. SNAME                 37        0
  8. S#                    37        0
  9. J#                    37        0
 10. JNAME                 37        0

CONSTANT TABLE
--------------
 CHARS
     1. STEELWORKS
     2. CAG

CODING TABLES
-------------
 RA_TAB
       RNAME     LRNAME     QUANT     WCOMP
         1          2         0         1
         3          4         2         1
         5          6         2         1

 ST_TAB          2


 LRN_TAB         2


 FN_MT_TAB
       FUNCT  ALIST  ALISTPTR  RNAME  LRNAME  ANAME  RELOP  WCOMP  TCOMP  CONTYPE  CONLEN  CONPTR
         0      0       0        3       4      7     324     1      1       5       10       1
         0      0       0        5       6     10     324     1      1       5        3       2

 DT_TAB
       LRNAME1   ANAME1   LRNAME2   ANAME2   RELOP   WCOMP   TCOMP
          2        8         4        8      324      1       1
          2        9         6        9      324      1       1

 DTOV_TAB


 WP_TAB
```

PUT W TASK

SYMBOL TABLE

1. TASK	37	0

WORKSPACE TABLE

21. W

CODING TABLES

ST_TAB	7
WN_TAB	21
RN_TAB	1
AL_TAB	
OR_TAB	

```
PUT W MATERIAL.(P#, PNAME, QOH)

SYMBOL TABLE
------------
  1. MATERIAL                  37        0
  2. P#                        37        0
  3. PNAME                     37        0
  4. QOH                       37        0

WORKSPACE TABLE
---------------
  21. W

CODING TABLES
-------------
  ST_TAB        7


  WN_TAB       21


  RN_TAB        1


  AL_TAB
          2
          3
          4

  OR_TAB
```

```
PUT W MATERIAL.(P#, PNAME, QOH)  UP P#

SYMBOL TABLE
------------
  1. MATERIAL                        37          0
  2. P#                              37          0
  3. PNAME                           37          0
  4. QOH                             37          0

WORKSPACE TABLE
---------------
  21. W

CODING TABLES
-------------
  ST_TAB          7

  WN_TAB         21

  RN_TAB          1

  AL_TAB
          2
          3
          4

  OR_TAB
       ORDER        LRNAME      ANAME
        206           0           2
```

DROP TASK

SYMBOL TABLE

1. TASK 37 0

CODING TABLES

ST_TAB 3

RN_TAB 1

AL_TAB

```
DROP MATERIAL.(WEIGHT, QOH)

SYMBOL TABLE
------------
  1. MATERIAL                          37          0
  2. WEIGHT                            37          0
  3. QOH                               37          0

CODING TABLES
-------------
 ST_TAB          3


 RN_TAB          1


 AL_TAB
        2
        3
```

```
NEW SUPPLIER (S#, KEY, CHAR, 5)(SNAME, CHAR, 15)(ADDRESS, CHAR, 20)

SYMBOL TABLE
------------
  1. SUPPLIER                      37        0
  2. S#                            37        0
  3. SNAME                         37        0
  4. ADDRESS                       37        0

WORKSPACE TABLE
---------------
 21. W

CODING TABLES
-------------
 ST_TAB        8

 RN_TAB        1

 ATT_TAB
     ANAME     KTYPE     ATYPE     ALEN
       2         1         5         5
       3         0         5        15
       4         0         5        20

  PUT W SUPPLIER

SYMBOL TABLE
------------
  1. SUPPLIER                      37        0

WORKSPACE TABLE
---------------
 21. W

CODING TABLES
-------------
 ST_TAB        7

 WN_TAB       21

 RN_TAB        1

 AL_TAB

 OR_TAB
```

```
READ (I, J)

SYMBOL TABLE
------------
  1. I                                 37         0
  2. J                                 37         0

VARIABLE TABLE
--------------
 31. I
 32. J

CODING TABLES
-------------
 ST_TAB        9


 IC_TAB
       31
       32
```

```
LIST (W, I, J)

SYMBOL TABLE
------------
  1. W                          37          0
  2. I                          37          0
  3. J                          37          0

CODING TABLES
-------------
 ST_TAB        109


 ID_TAB
         1
         2
         3
```