"DESIGN AND DEVELOPMENT OF DATA BASE SOFTWARE FOR EDUCATIONAL USE"

by

ROBERT GODFREY
Adv. Dip. Numerical Analysis and Statistics (Salford), M.A.C.S.

A thesis
presented for the degree
of
Master of Science
in the
University of Adelaide.

Based on work performed in the
Computer Centre, South Australian Institute of Technology
in association with the
Department of Computing Science, University of Adelaide.

December, 1984.

DEDICATION

To my father who died while this thesis was being prepared.

SUMMARY

This thesis describes an experiment in the modular construction of data base software for an educational environment.

The thesis commences with a description of the educational uses of data base software and specifies why commercially available software is often not suitable for this environment.

A major review of the database software literature follows. This review examines the hierarchic network, relational and inverted models, and examines the ways in which the user is given independence from physical database storage mechanisms. The data dictionary concept and the role of the Data Base Administrator is discussed followed by a description of different types of user interface languages. The review concludes with the security aspect of database software.

Next the thesis details the objectives, methods and procedures of the software implemented. The software consists of a multi-model database system (hierarchic, inverted and sequential file) with a common query/update language linking the three models.

The query/update language QUILL is then described, followed by the sequential file system SEQUENT, the inverted system INVERSE, and the hierarchic system PYRAMID.

Finally the thesis examines the software developed in retrospect, and also comments on the feasibility of adding other models to the multi-model software.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my thesis supervisor Professor Frank Hirst firstly for giving me the opportunity to undertake this study, secondly for his patience in allowing me time to chart a course around the many pitfalls my research led me into, and finally for his guidance about how to proceed with writing up the work done.

Thanks are due also to my own department head, Professor Bob Northcote, for his encouragement to press on with and to finish the research and then the thesis. I am grateful to him for allowing me time and space to write up this thesis and to those of my colleagues who were allocated extra tasks by him which would otherwise have fallen to me.

The Computer Centre and its staff at the South Australian Institute of Technology are appreciated for providing the computer facilities on which the software was developed.

Andrew Smith of ICL(UK), firstly in Reading and then later in Bracknell is remembered for it was while working under him in 1968 that my interest in database technology was kindled, an interest that has remained strong to this day.

My first wife Barbara is remembered for encouraging me to turn this interest into an actual programme of research.

I express my appreciation to Angela McKay, not just for typing this thesis, but for checking with me daily to see if I had any more pages written. Without her interest and help this thesis would have taken far longer to write.

Finally, to me wife Margaret, a big THANK YOU, for your continuing love, patience and support during the many evenings I spent locked away in my study surrounded by piles of papers.

Robert (Bob) Godfrey.

STATEMENT OF AUTHENCITY

This is to certify that this thesis contains no material which has been accepted for the award of any other degree or diploma in any University. To the best of my knowledge and belief, it contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signed.... ....
(R. Godfrey)

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 The Educational Uses of Database Software

This thesis is concerned with the design of database software for educational users. The term "educational user" used in this thesis is taken to mean the student and their instructors in a tertiary institution.

The term "student" is intended to include both students intending to make a career in computing and also those using a computer as part of some other discipline (e.g. Accountancy, Business Management, Town Planning, etc.)

By "instructor" is meant those lecturers, tutors, etc. actively involved in teaching students about database systems.

The term educational user deliberately excludes the use of a database for administration, research and consulting even though these activities may also be carried out within the tertiary institution.

In the remainder of this thesis the term "user" should be taken to mean "educational user" unless indicated otherwise.

These users require a data base system so that they can:

(1) dissect and/or modify the software to gain an insight into how such software works and to explore its potential;

(2) use the system in a conventional way.

Category (2) above can be further subdivided:

(2a) "Vocational users" who will use the system because it is typical of, or similar to, other such systems that they will meet in the outside world;

(2b) "Non-vocational users" who will use the system simply because it is the most effective tool for their current activity.

The needs of these 3 groups of users can be met in either of two ways:

(a) by using commercially available database software;

(b) by using purpose-built database software that has been specifically designed for educational use.

Hawryzkiewycz (1979) has described a DBMS course using Burroughs DMS-II for practical work.

In broad terms, this thesis is concerned with an experiment using method (b) above.

McDonnell (1981) has described a CODASYL mini-DBMS and an instructional relational algebra (IRA), and the software described in this thesis can be viewed as adding to the range of such systems available to a database instructor.

1.2 The Educational Environment

Bradley (1982) has observed

> "There is a final problem for the data base instructor, about which little can be done in a textbook, and that is the problem of student access to suitable database management systems. I believe that at the present time the CODASYL and Relational approaches are the most important from an educational point of view. Yet it is still rare for an institution to have access to both systems, and there are some that have access to neither."

Gudes (1977) has suggested using a text and Computing Surveys to design meaningful assignments, but this thesis contends that a better approach may be the construction and use of purpose built software.

The major commercially available database systems IMS, IDMS, ADABAS etc. are aimed at the large business enterprise, although they can be used for education (Honkanen 1983). However, it is contended that this software is unsuitable for an educational environment for the following reasons:

1. It is very expensive.
2. It uses large amounts of processor resources.
3. It is designed to be productive and to "meet all needs of all men".
4. It is intended to be used by say 10 - 100 users sharing common data.
5. It is too complex, offering more facilities than can comfortably be taught.
6. It is designed for the long term (even if ad hoc) user. That is, users will use the system, however infrequently, over a period of years.

The typical educational environment for a student machine:

1. has limited money for software purchase (Montgomery, 1980);
2. has processor resource limits geared for small BASIC programs;
3. is selective in matching the "real world" by simplifying and removing or reducing time consuming routine tasks;
4. may have several thousand largely independent users who generally work on their own problems and data;
5. the majority of users will use the system for a limited period, say a term, semester or year, while completing a particular subject. They will not be computer professionals.

1.3 Design Factors

When designing database software for educational use, the following factors need to be considered:

1. It must be easy to learn (typically say in 2-3 hours of class/lab. time).
2. The majority of users will not possess a manual so all error messages need to be clear and non-cryptic.
3. The software must be able to be swapped with other educational users. To this end the sofware should be written in a common standard language.
4. Some users (research fellows and computing majors say) will want to modify and adapt the system to their own ends. The sofware should be built on sound engineering principles using exchangeable/ replaceable modules.
5. The software should contain the essential features of real world systems, but need not contain all such features.
6. The software should be useful both for computing and non-computing majors.

1.4 Summary

The thrust of this thesis is as follows:

- there is a need to teach the use of database software;
- this teaching cannot be carried out satisfactorily without using a DBMS;
- commercially available software is generally not suited to this purpose;
- special purpose software can be built to meet the need for a DBMS.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Introduction

Tsichritzis (1977a) comments thus on DBMS research:

> "Since DBMS is a relatively new discipline many people have converged into it from other areas."

Because of this, DBMS research and literature overlaps with many other computing (and non-computing) areas. These areas include Operating Systems (for I/O processing), Systems Analysis (Database design), Programming Languages (Data types), Artificial Intelligence (Distributed Databases), Software Engineering (Multi-purpose architecture), and Hardware (Database machines). It is difficult therefore to draw a neat boundary around database literature and hence to control the scope and size of any review of that literature.

This review will be confined to those topics of paramount importance to the construction of database software and, in particular, to the educational aspects of this software.

The initial review will consider the more important database models and their place in multi-model database architectures, followed by consideration of database description and the role of the Data Base Administrator. Following this language interfaces are examined, followed by security issues.

2.2 Database Models

2.2.1 Introduction

Tsichritzis (1977:32) defines a data model as "an intellectual tool used to understand the logical organization of data."

Models are used to enable people to think about the nature and processes of the "real world". The model seeks to remove extraneous material and also to simplify the nature of the real world.

The model can be used solely as a design tool or it can be embodied in a database software system. Because the model is needed to serve many purposes some authors define several types of model. Thus Robinson (1981:29-36) defines the following:

- device data model - a device/machine perception in terms of blocks, pages, etc.;
- storage data model - a view in terms of stored records and access mechanisms;
- logical data model - a global view of the data and its inherent logical characteristics (structure, access constraints, integrity constraints, etc.);
- logical data sub-models - a perception in terms of constructs manipulated by high-level languages (fields, records, etc.).

Multiple model approaches to data models require an "architecture" to place the models in the correct relationship with each other, with the users and with the data itself. Thus Robinson's four models are related as shown in Fig. 2.1. He comments that work is still continuing in this area (both at a theoretical and at a practical level) and "it may be some time before agreed definitions of the architecture and its models are reached."

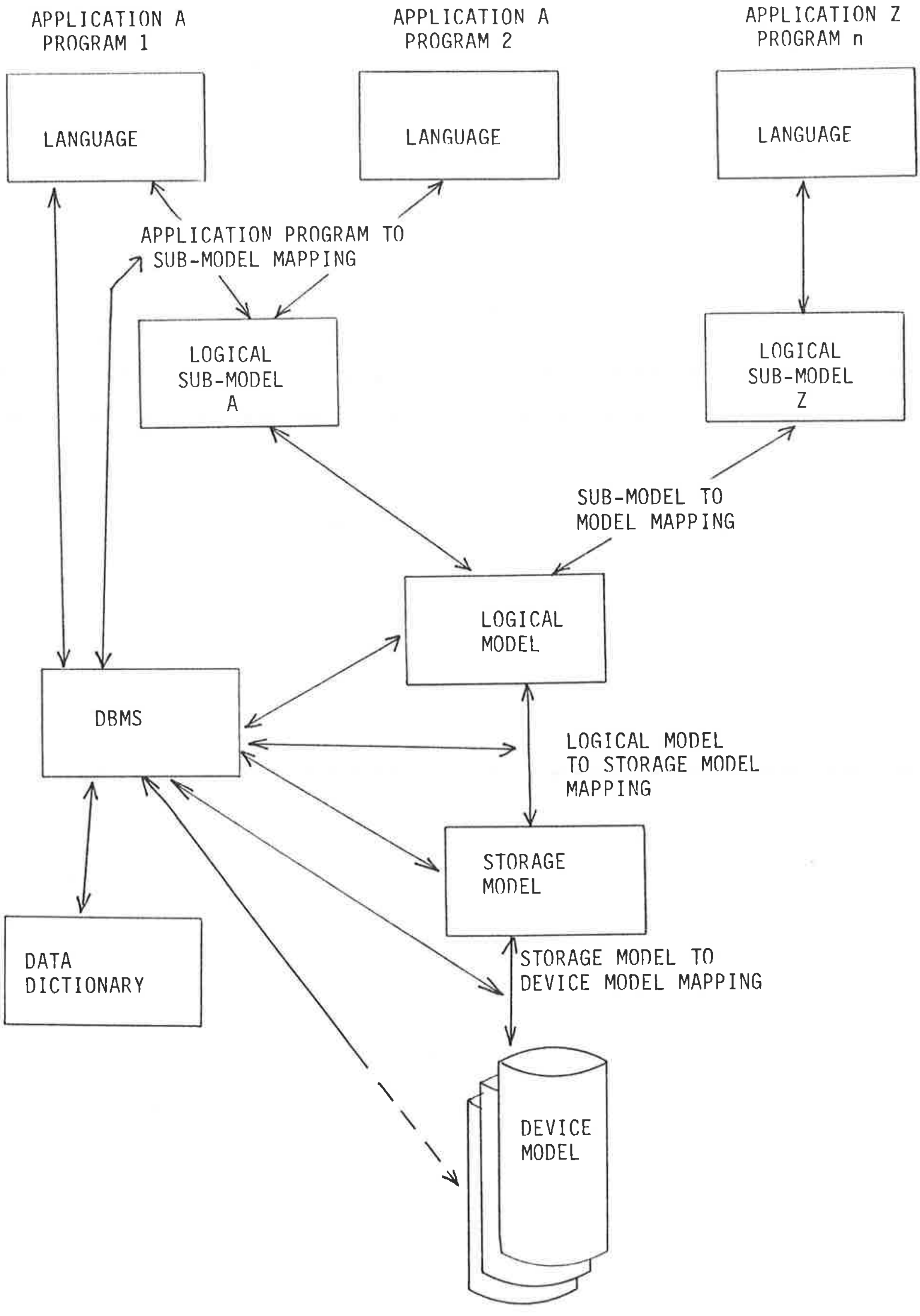


Figure 2.1: Generalised Architecture for a Database System (Robinson)

Date (1977:14) bases his architecture diagram (Fig. 2.2) on the ANSI/X3/SPARC proposals with its three schemas

. External schema - the view of an individual user;

. Internal schema - the way in which the data is stored;

. Conceptual schema - a global view of the data, independent of how it is stored or how it is used.

Tsichritzis (1977:96-97) also uses the ANSI/SPARC architecture. Tsichritzis observes, however, that "most existing commercial DBMS's... combine conceptual and internal schema facilities, and hardly provide any external schema views." The PYRAMID system described in Chapter 7 follows this common approach and combines the internal and conceptual schemas. It does however aim to provide for more than one external view.

Rowe and Stonebraker (1981) describe four options for database architectures (see Figs. 2.3-2.6). They state "We believe these architectures are the only resonable candidates for future DBMS packages."

Option 1 (Fig. 2.3) has a high level interface on top of an intermediate interface (such as CODASYL) where users can access either interface. They give UNIVAC's DMS-1100 as an example of this architecture.

TANDEM's ENFORM is an example of the (Option 2) architecture (Fig. 2.4) where a high level interface sits on top of a low level (e.g. Record Manager) system. Thus programmers can either process files directly (e.g. using COBOL READ/WRITE verbs) or can use say a query/update language to access files.

If the low level system of Option 2 cannot be accessed by the user then Option 3 (Fig. 2.5) results. INGRES is given as an example of this architecture.

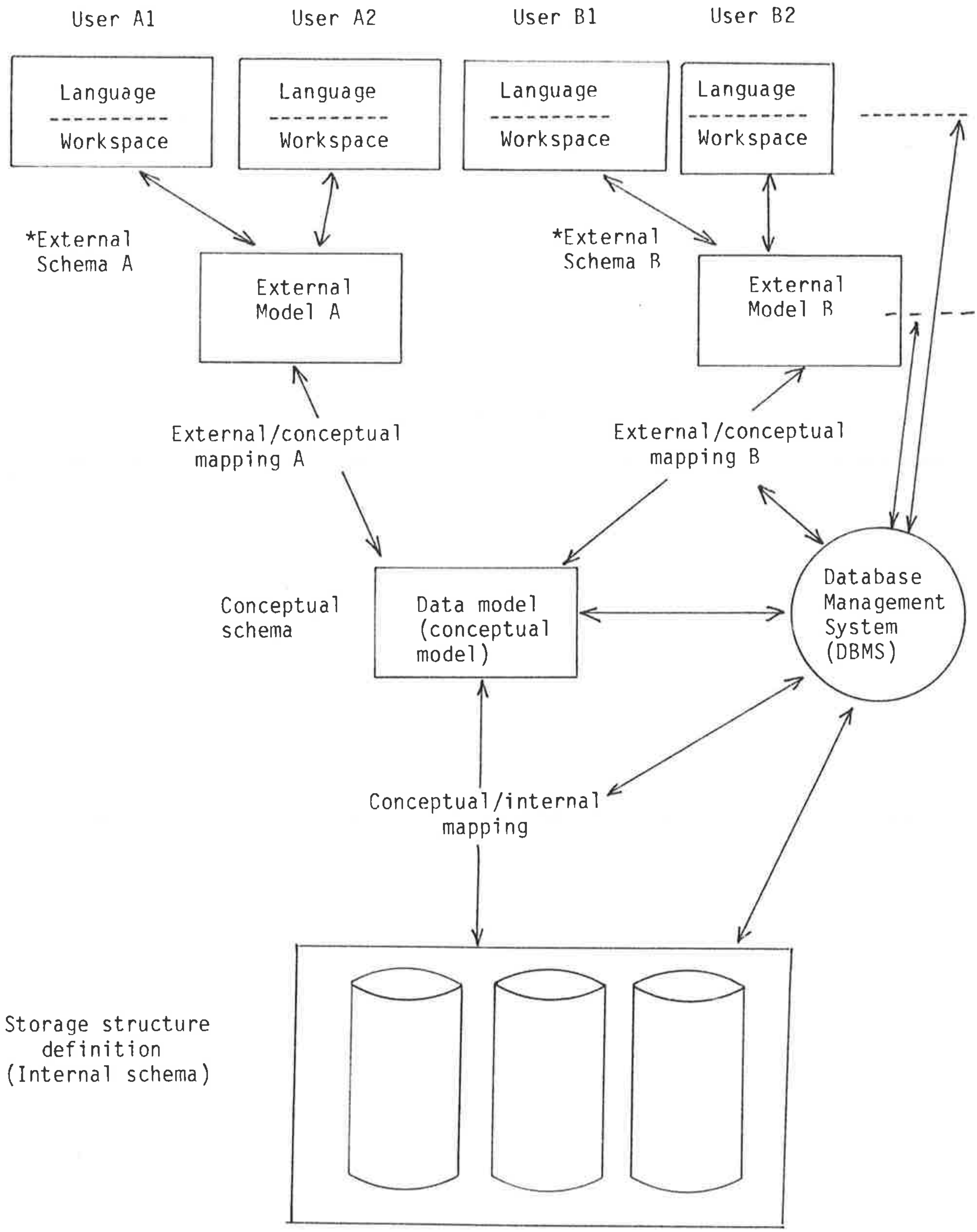


Figure 2.2: An architecture for a database system (Date).

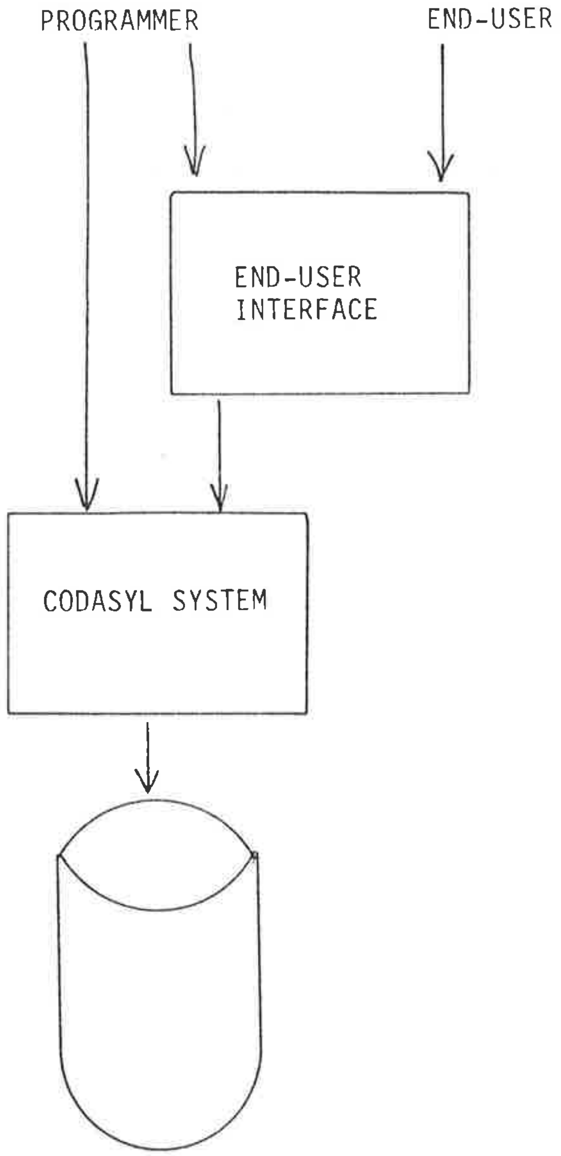


Figure 2.3: DBMS Architecture (Option 1)

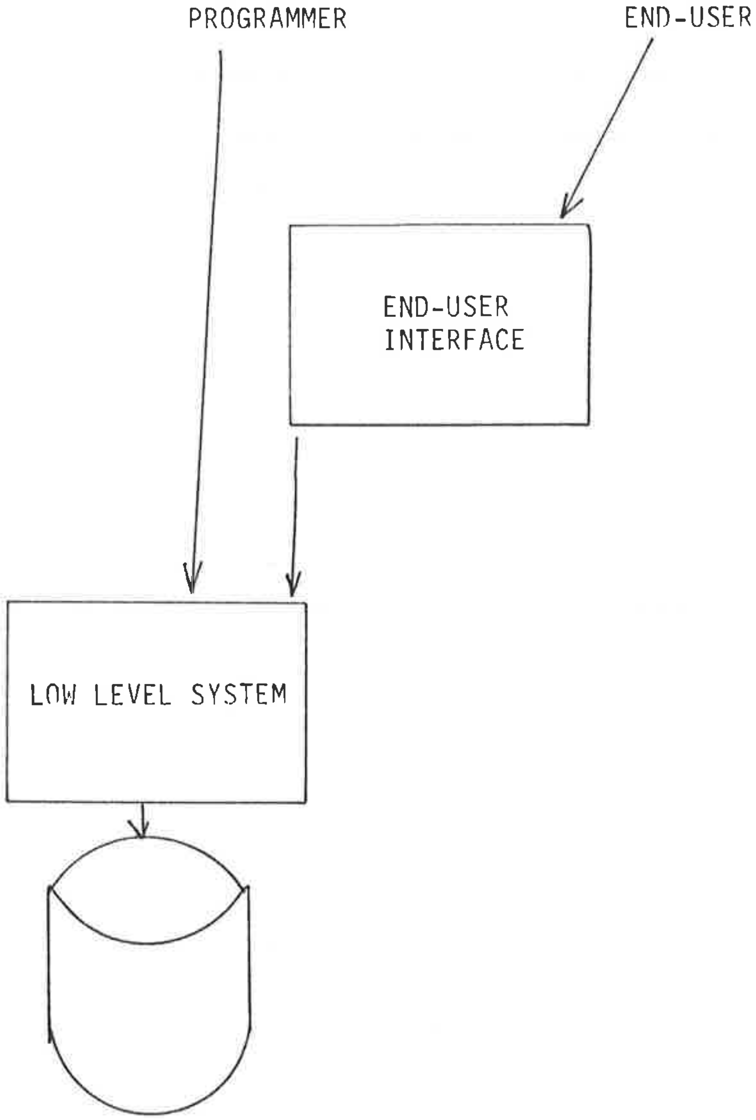


Figure 2.4: DBMS Architecture (Option 2)

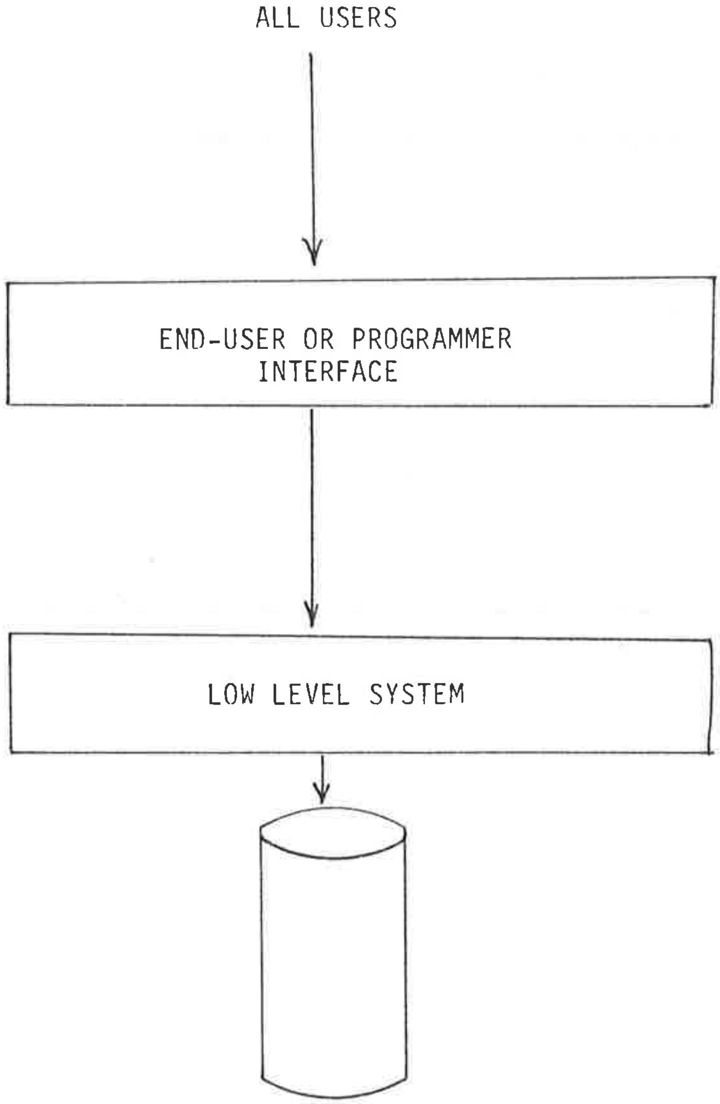


Figure 2.5: DBMS Architecture (Option 3)

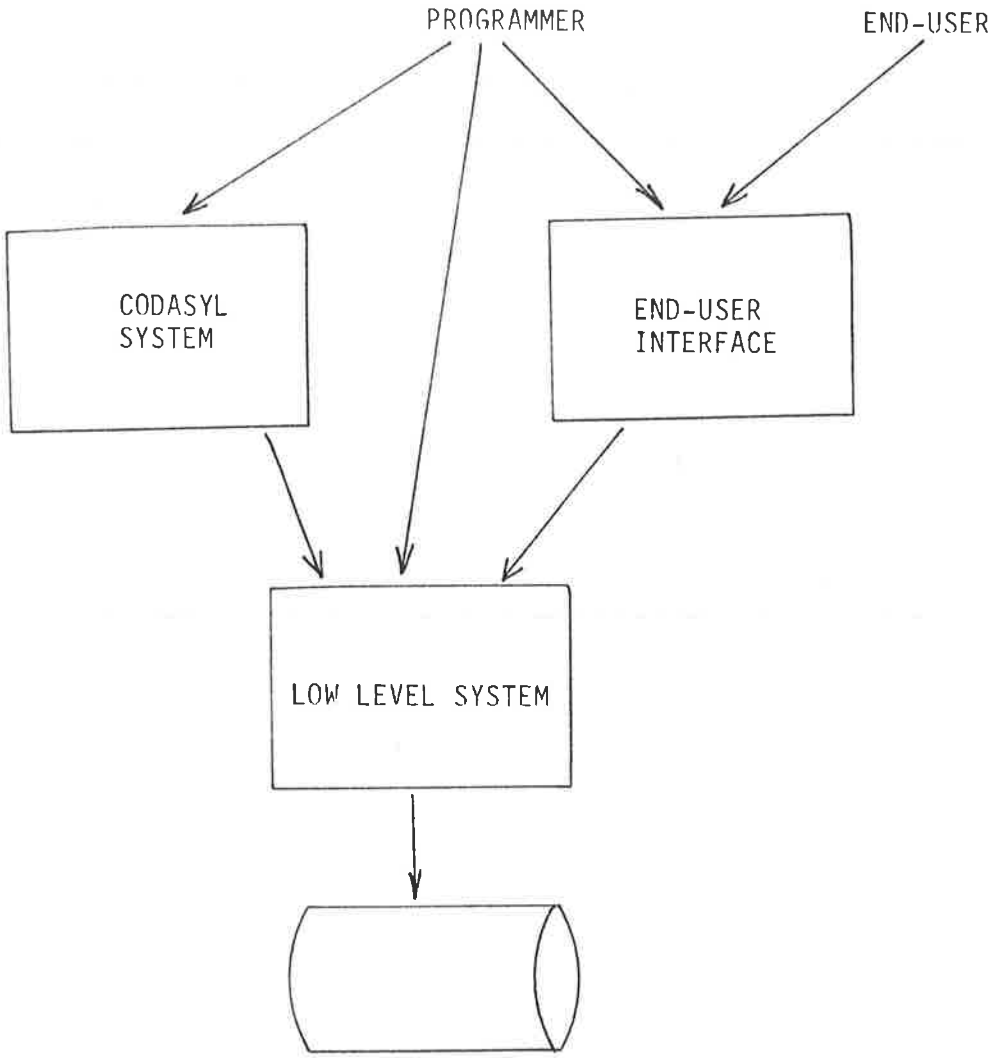


Figure 2.6: DBMS Architecture (Option 4)

The fourth architecture (Fig. 2.6) has an intermediate level interface (e.g. CODASYL) and a high level interface on top of a low level interface. Rowe and Stonebraker could not find any example of this architecture. They considered an alternative to Option 4 in which the end-user interface interfaces not with the low level interface but with the intermediate interface. They did not consider this alternative in great detail as in their view it offers approximately the same advantages and disadvantages as the original Option 4.

Option 4 is clearly the most complex but it does offer the greatest flexibility in terms of user interfaces. Accordingly, the architecture chosen is basically this option with the exception that the CODASYL model is replaced with a variety of different database models and the end-user interface being the QUILL query language. This is shown in Fig. 2.7. Not all operations are possible at all levels but an attempt has been made to permit some operations at all three levels to enable students to use and hence appreciate the differences between the various levels.

The use of multiple intermediate interfaces (PYRAMID, INVERSE and SEQUENT described in Chapters 5, 6 and 7) is motivated by the very different advantages and disadvantages of each model to certain groups of users. To select only one model is to deny or at least deter some users from the system. The use of such "coexistence" or "multi-model' architectures have been extensively advocated (Tsichritzis, 1977a; Hawryszkiewycz, 1980; Deen, 1980 and 1981; Sockut, 1981; Champine, 1979; Zaniolo, 1979; Mercz, 1979).

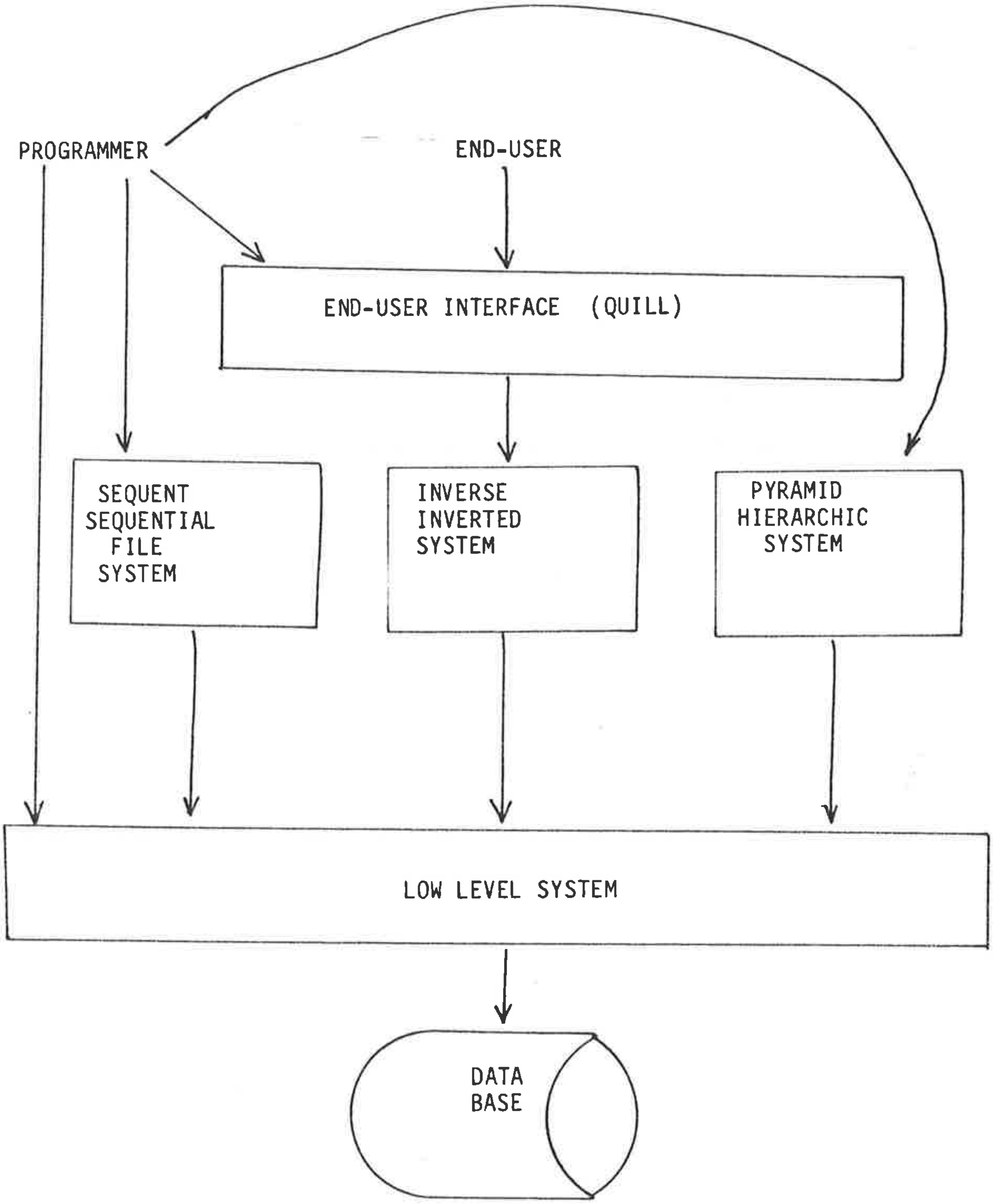


Figure 2.7: QUILL Architecture

Kroenke (1983) relates "six common useful models" using the diagram below

HUMAN (Logical) ⟷ MACHINE (Physical)

Semantic Data Model (SDM)	Entity-Relationship Model (E-R)	Relational Data Model	CODASYL DBTG Model	DBMS-Specific Model
ANSI/X3/SPARC				

He ranks five of the models as being oriented towards human meaning or machine specifications, with the sixth (ANSI/X3/SPARC) in a class of its own. He does not rank the hierarchic model (in his view hierarchic = IBM's DL/I) or the (non-CODASYL) network model but includes these specific implementations in the DBMS-specific models (including ADABAS, SYSTEM200, TOTAL, IMAGE). If one of these products is to be used Kroenke recommends using the SDM (McLeod 1978) or similar model to develop the logical database design and then transferring this design into a physical design for the available DBMS. Vetter (1981: 72-92) uses the E-R model for this purpose. Many use the normalisation parts of the relational model for this design process but Codd (1980) has pointed out that the relational model is more than a data structure (flat files) but includes the relational algebra operators and some integrity rules.

Kent (1978) groups the hierarchical, network and relational model as variations of the traditional record model and notes "an increasingly visible trend away from record oriented data models towards models which might generally be called semantic nets, or graph structured models." This visibility is "everywhere except in current commercial database processing."

As this thesis is concerned with database software, the topic of logical database design will not be pursued further. The concern here is for physical database design using one of the commercially implemented models, it being assumed that one of the logical design models having already been used as proposed by Kroenke (1983) or Vetter (1981).

2.2.2 Hierarchic Model

The hierarchic model is clearly the poor relation when compared to the network and relational models. It lacks the theoretical nicety of the latter, and can be viewed as a subset of the former. The hierarchic model is important however, if only because (Robinson, 1981) "people use them", and the software implementations are proven (Atre, 1980).

The hierarchy is a common structure (Tsichritzis 1976) in everyday life and the model is easier to understand than the other two models. Clemons (1981) believes "that an external schema facility is best based on hierarchies." Lien (1981) also proposed that a hierarchical view of relational databases may be preferable to the view of a relational database as a series of projections of one universal relation.

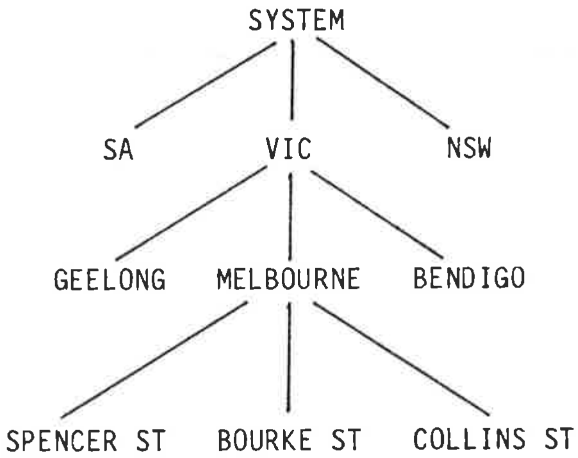
Kroenke's (1983) observation that "hierarchic data model" and "DL/I" are synonymous has already been referred to. Tsichritzis (1977b), while not explicitly saying so, nevertheless writes as if the two are the same. Date (1977:55-58) however treats hierarchies independently of IMS's DL/I. Perhaps the strongest critic of the narrow approach is Bradley (1982):

> "Because of the fairly wide use of IMS, some authors have contented themselves with a description of IMS instead of describing the hierarchical approach in general. We believe this to be an undesirable strategy from an educational point of view...."

The hierarchic model views data as records connected via 1:n relationships in an inverted tree. Each record occupies a node of the tree and can own zero or more records but apart from the root can be owned by one and only one record. The root node at the top of the tree has no owners.

Consider the hierarchy of record types

This hierarchy has zero or more states, each owning zero or more cities. Each city owns zero or more streets. Thus a typical instance of this hierarchy might be



It is often convenient to conceptualize a virtual record say "system" to own the instances of the root record type.

The major disadvantage of the hierarchic model is its clumsy handling (Atre, 1980) of the two way relationships found in networks. Thus given a requirement to process the triad of records: CUSTOMER, ORDER, PRODUCT a hierarchic model must select one of the hierarchies below

CUSTOMER
↓
ORDER
↓
PRODUCT

PRODUCT
↓
ORDER
↓
CUSTOMER

Bradley's "hierarchical conceptual database" would select one of these as the primary hierarchy and then derive a secondary hierarchy to convert the network conceptual database to a hierarchical conceptual database. This is done by adding another link record into the database (CUST-ORD) as in Fig. 2.8. The two primary hierarchies

CUSTOMER
↓
CUST-ORD

PRODUCT
↓
ORDER

are also linked by the secondary hierarchy

CUSTOMER
↓
CUST-ORD/ORDER
↓
PRODUCT

The most widely used hierarchic database system is IBM's Information Management System (IMS) (see Date, 1977) which divides its database into "segments". There is a "root" segment type with the other segment types being dependent segment types. Each "parent" segment type has at least one "child" segment type.

MRI's System 2000 (Cohen 1978) is based on an inverted list in a hierarchically structured database. In a System 2000 database "index",

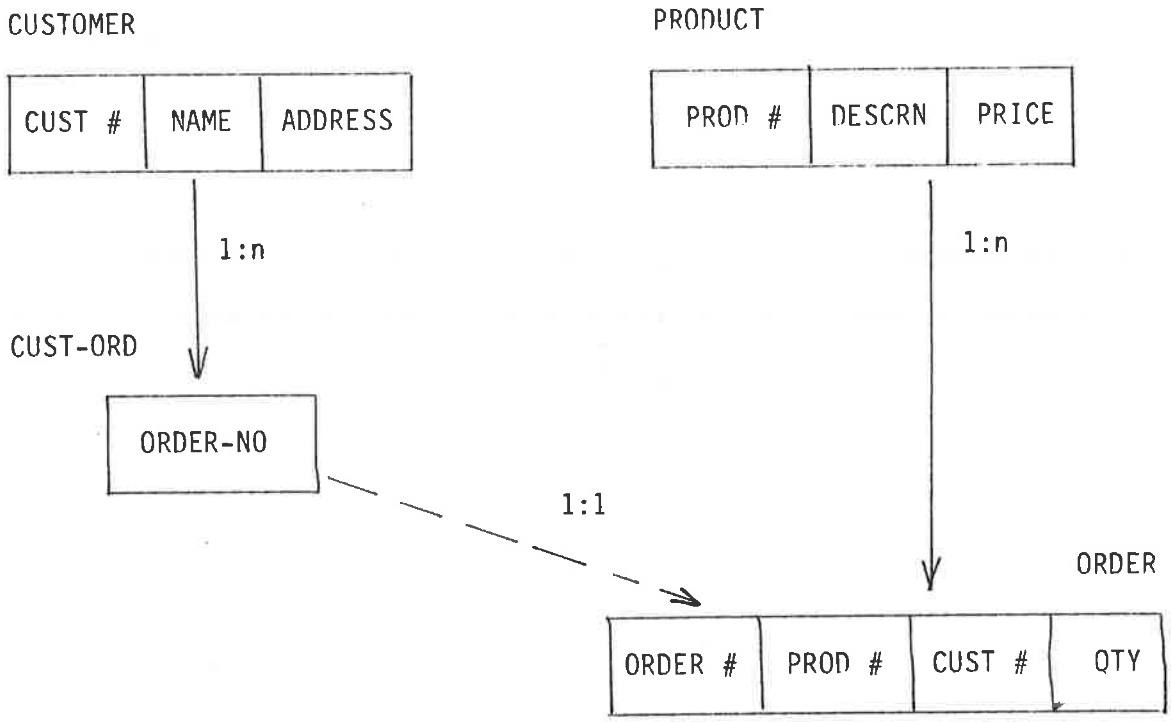


Figure 2.8: Primary and Secondary Hierarchies

"structure data" and "content data" exist on separate files. The term "repeating group" is used to denote a type of "dataset" (record) consisting of a number of "elements" (fields). Each data set not the root repeating group has one and only one "parent" one level above it. An "ancestor" will occur at each higher level above any data set which is not the root repeating group. All data sets which trace their ancestry to a common data set are considered "descendants" of that data set whether they occur immediately below or at deeper levels. Data sets which share a common parent are "siblings".

2.2.3 Network Model

The network model has been used as the basis for the CODASYL database proposals, and while this is the most important use of the model, other network implementations (e.g. TOTAL) are also of importance. Reference has already been made to Kroenke's (1983) view that any non-CODASYL network model is a DBMS specific model. Atre (1980: 109-123) is not as explicit but treats the terms "CODASYL model" and "network model" synonymously. Tsichritzis (1977: 136-184), however considers the CODASYL model to be a restricted form of the more general network model. The relationships in a network model can be 1:1, 1:N or N:M. However (CODASYL, 1971) requires all relationships to be potentially 1:N.

This 1:N relationship is fundamental to the CODASYL proposals and most other network DBMS's. If two record types (by STATE, CITY) are connected by a 1:N relationship from STATE to CITY then each STATE record can be connected with many CITY records. Conversely each CITY record can only be connected with one STATE record. The STATE record is said to be the "owner" of a "set" of CITY records and the CITY records are said to be "members" of the set. This set construction can be used to create both hierarchies and networks (CODASYL 1971, Olle 1973).

Tsichritzis (1977) considers the problems of modelling N:M relationships within the CODASYL model. Thus if an N:M relationship (Fig. 2.9) exists between say STATE and COMPANY then an intermediate record type (MANUFACTURES say) is required along with two links MANUFACTURES IN between STATE and MANUFACTURES, and IS MANUFACTURED between COMPANY and MANUFACTURES (Fig. 2.10).

The CODASYL DataBase Task Group (CODASYL 1971) proposals have been used as the basis for many commercial DBMS's (Cullinane's IDMS, DEC's DBMS-11, UNIVAC's DMS 1100, Burroughs DMS-II etc.). Fry (1976) gives some of the history of the CODASYL proposals, starting with G.E.'s I-D-S, through the (CODASYL 1969) report and further reports in 1971, 1973, 1975, 1976. A further major CODASYL report followed in 1978 (Caelli 1979). Each of these reports have been developments and refinements of the work of various CODASYL committees.

The CODASYL database is described in the "schema" which defined all record formats and set constructions in the database. A sub-schema defines the user view of a single application. Although a Device Media Control Language (DMCL) to handle file and device assignments was mentioned (but not defined) the architecture was essentially of two levels. By 1978 however following the ANSI/SPARC three level architecture the 1978 CODASYL proposals revised their architecture to fall into line with this newer concept. The 1978 CODASYL architecture is shown in Fig. 2.11 (Caelli 1979). The sub-schema and schema correspond to the ANSI/SPARC External and Conceptual Schemas respectively, with the DSDL matching the Internal Schema.

The CODASYL user accesses the database using a host language Data Manipulation Language. Comprehensive examples of programs using DML can be found in BCS (1971) and Dee (1973).

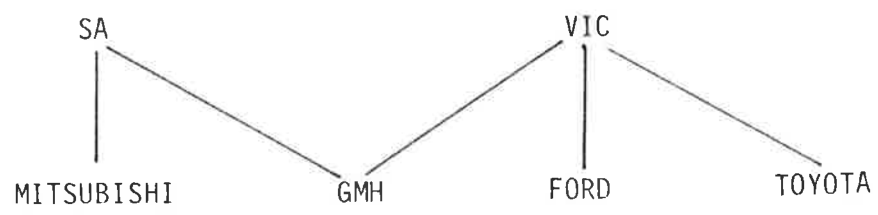


Figure 2.9: N:M Relationship

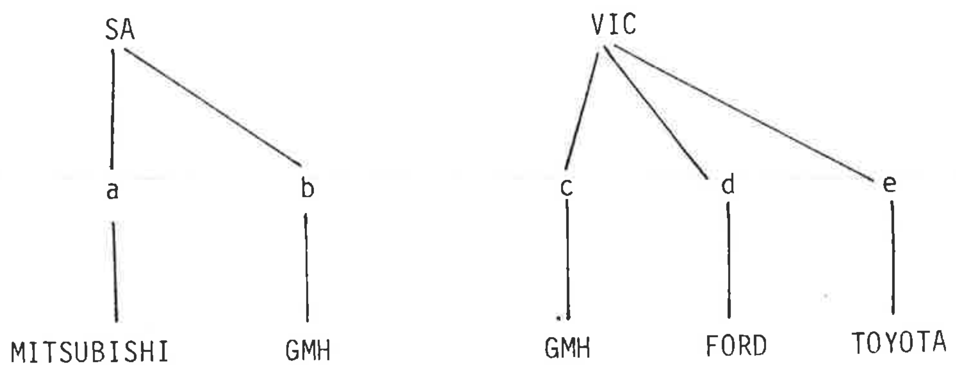


Figure 2.10: 1:N Relationship

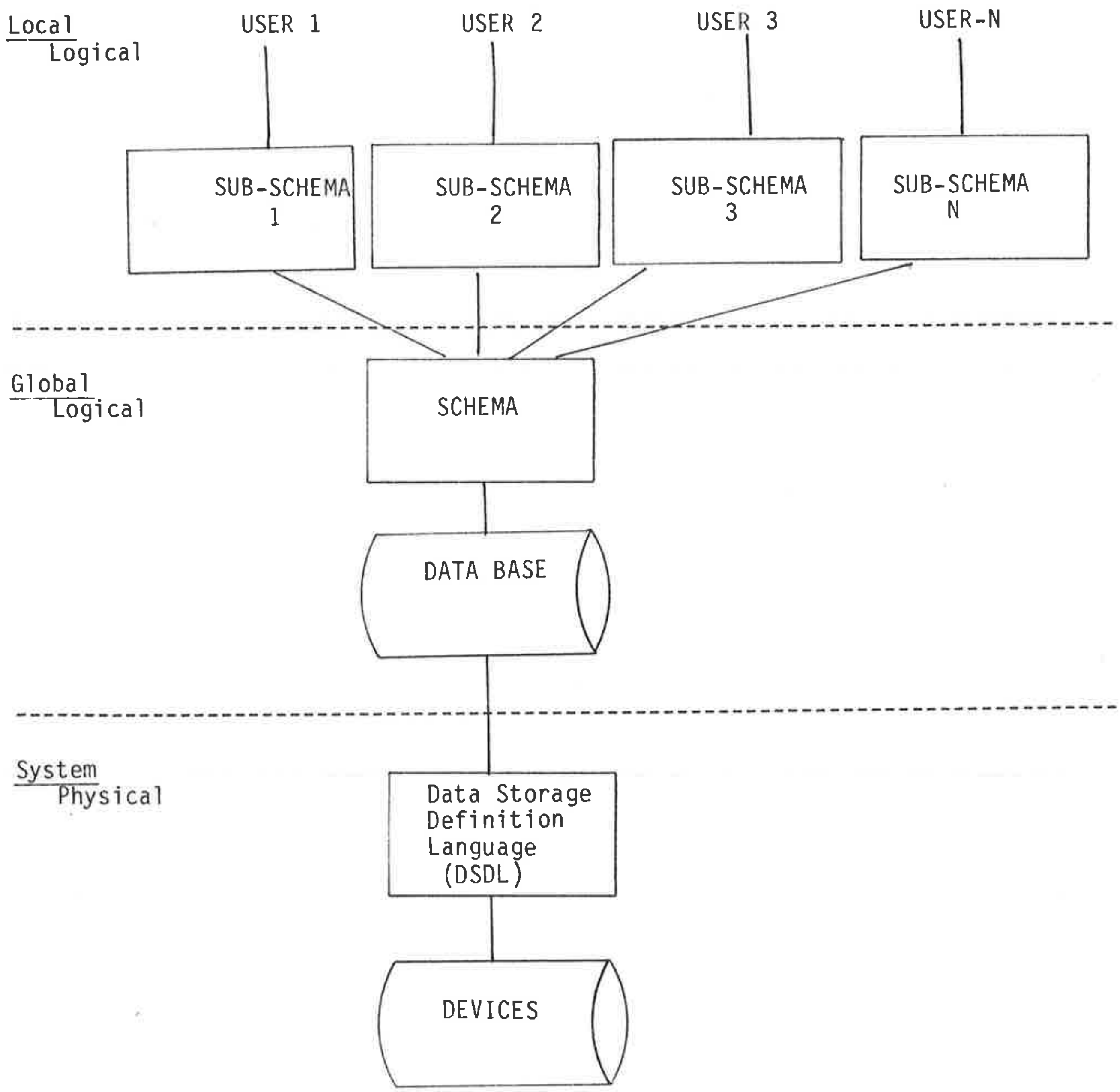


Figure 2.11: CODASYL 1978 Data Base Architecture

The central DML statement is FIND which locates a record in the database. (GET is used to retrieve fields from located records.)

Thus in a COBOL host program the statements

```
ACCEPT PART-NO.
FIND PART
GET PART; PART-NO, PRICE, DESCRIPTION.
DISPLAY DESCRIPTION, PRICE.
```

would locate and retrieve fields from a specific PART record.

2.2.4 Relational Model

While the network model has been the basis for most of the commercially available DBMS's, the relational model has been the subject of the greatest research.

Although some of the ideas had been known for some years, Codd (1970) was the first person to give structure to the concepts. In later material (Codd 1971 a,b,c: 1974, 1979, 1980) these ideas were refined. In the meantime several others had added to the wealth of literature on the subject. Chamberlin (1976) and Kim (1979) give comprehensive bibliographies of much of this work.

In his original paper (1970) Codd applies elementary relation theory to two problems - "data independence" and "data inconsistency". He cited as two important advantages of the relational model to be firstly that it did not need any additional pointers or the like, and secondly that it forms a sound basis for treating derivability, redundancy and consistency.

There are two main thrusts to the work on relational databases. Firstly the structure of the relations themselves and their "normalization"; secondly the development of a Relational Algebra and Calculus to manipulate the relations. Many authors have ignored the second thrust and treated relational data bases merely as a so-called "flat file" model. Codd (1980) takes them to task for this with the observation:

> "This is like trying to understand the way the human body functions by studying anatomy but omitting physiology."

He defines a data model thus:

> "1. a collection of data structure types (the building blocks of any database that conforms to the model):
>
> 2. A collection of operators or inferencing rules, which can be applied to any valid instances of the data types listed in 1., to retrieve or derive data from any parts of those structures in any combinations desired;
>
> 3. a collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes or both... these rules may sometimes be expressed as insert-update-delete rules."

The basic data structure for a relational database is the relation. Relations are normally shown as arrays, though this is not essential (Codd 1970).

Three sample relations (C, P and O) are shown in Fig. 2.12. Each relation "closely resembles a traditional sequential file" (Date 1977).

The rows of the relations are called "tuples" and their order is immaterial. The ordering of columns is significant and this significance is partly conveyed by labelling it with the name of a "domain" (Codd 1970). There is confusion in the literature over the use of the terms "domain" and "attribute" to refer to a column. Kroenke (1883: 243) just refers to attributes and many people follow this style. However the most useful distinction between the two terms is perhaps given by Date (1977) and Deen (1977). They define an attribute to refer to the column and the domain to be the set of values that can appear in the column. Both column and attribute can be named. As both Deen and Codd (1970) have pointed out, a relation may have two columns from the same domain (but being different attributes, e.g. father's age, mother's

C (Customer)

C#	CNAME	CITY	STATUS
1	Smith	Adelaide	1
2	Jones	Melbourne	1
3	Wilson	Adelaide	2

P (Part)

P#	DESN	PRICE
1	DESK	250
2	CHAIR	140
3	TABLE	180
4	BOOKCASE	100

O (Orders)

C#	P#	QTY
1	1	5
1	3	4
2	1	1
2	2	3
2	3	2
2	4	4
3	2	6

Figure 2.12: Customer, Part and Order Relations

age). Codd notes that many current DBMS's do not provide for two or more identical domains and hence for most purposes attribute and domain can be used synonomously.

The production of normalized relations was dealt with first by Codd (1970) when "first" normal forms were dealt with. Subsequently (Codd 1971a) "second" and "third" normal forms were introduced to make relations easier to understand and control. In his 1971 paper Codd stated that use of third normal form would "significantly extend the life expectancy of application programs." The rather abstract paper (Codd 1971a) was followed by a tutorial discussion (Codd 1971c). Each of these higher normal forms make database operations more consistent than operation on lower normal forms.

For a time it was considered that third normal form was the highest possible or desirable form. However, Fagin (1977) formalized the notion of a "fourth" normal form and Date (1977) mentions the independent work of Zaniolo in this field. Fagin (1979) continued work and the "fifth" normal form was born. Ling (1981) has suggested an improvement to third normal form. Kent (1983) summarises the development of these five normal forms.

The normalization concept is now an accepted part of the process of database design, not just for relational databases but also for hierarchic and network systems.

However, reference has also been made to the necessity to consider the Relational Algebra and Calculus and their place in the relational model. Both are techniques for manipulating databases, the first a lower level procedural language, and the second a high level non-procedural language.

The relational algebra was introduced by Codd (1970). The two principal operators introduced at this time were the "project" and "join" operators.

Projection is basically the extraction of one or more columns of a relation and then the elimination of any duplicate tuples that result. Referring back to Fig. 2.12, if we project relation C over the attribute CITY, we obtain a relation containing Adelaide and Melbourne, in other words all city names in the relation.

Join is basically the merging of two relations using an attribute from one to cross-reference to one or more tuples in another relation. It accomplishes what in the hierarchical and network models is often achieved by inter-record links. To join relations O and P over the attribute PART# effectively creates a new relation like O but with the appropriate DESN and PRICE fields appended to each tuple.

The relational algebra was extended (Codd 1971b) to include the division and restriction operators. The concept of combining several operators to form a relational algebra expression was also introduced. Thus to find the identity of any customer with orders for all parts, first project P over P# to form relation Q (just containing P#) and then divide O by Q. Date (1977: 117) gives a similar example.

Both the above operations can be combined in a single arithmetic expression.

The problem for programmers with the relational algebra lies with its non-navigational approach. While it is relatively easy to take an expression and say what it will do, it is much harder to have a need and then write an expression to satisfy that need. A parallel could perhaps be drawn with mathematics here - if mathematics appeals to a student then its use seems natural and simple, if the reverse is true then while the student may be able to follow a worked example, they may not be able to solve problems for themselves.

The relational calculus (Codd 1971b) is an attempt to help overcome this problem and is further addressed by Codd (1974). The former paper gives an algorithm for translating a calculus expression written in DSL ALPHA into a semantically equivalent sequence of operations in the relational algebra. Codd envisages a great variety of languages for accessing databases and considers the completeness of such languages for accessing a relational database. He divided such "data sublanguages into calculus and algebra related languages (see Fig. 2.13).

The form of expression for the calculus given by Codd (1971b) is based on mathematical symbols, but Date (1977) gives examples based on SEQUEL which are easier for non-professionals to follow. Using Fig. 2.12 again, to find all status 1 customers in Adelaide one would write

```
SELECT C#, CNAME
FROM   C
WHERE  CITY = 'ADELAIDE'
AND    STATUS = 1
```

Again the initial feeling against the relational calculus was based more on its mathematical form of expression than on its potential usefulness. More user-friendly versions are now readily available - AQL (Antonacci 1978), SQUARE (Boyce 1975), BSOL (Baxter 1978), CASDAL (Su 1978), REMOTE-OBE (Combes 1980) to name but a few.

Again for a long time System R (Astrahan 1979 and 1980, Chamberlin 1981) was the only well known commercial implementation of a relational package. The market is now "flooded" with such products - INGRES (Stonebraker, 1976), ORACLE (   ), RAPPORT (Logica, 1982) and many others. Brodie (1981) lists 75 vendor systems. In Canning's (1982) words "Relational Database Systems are here".

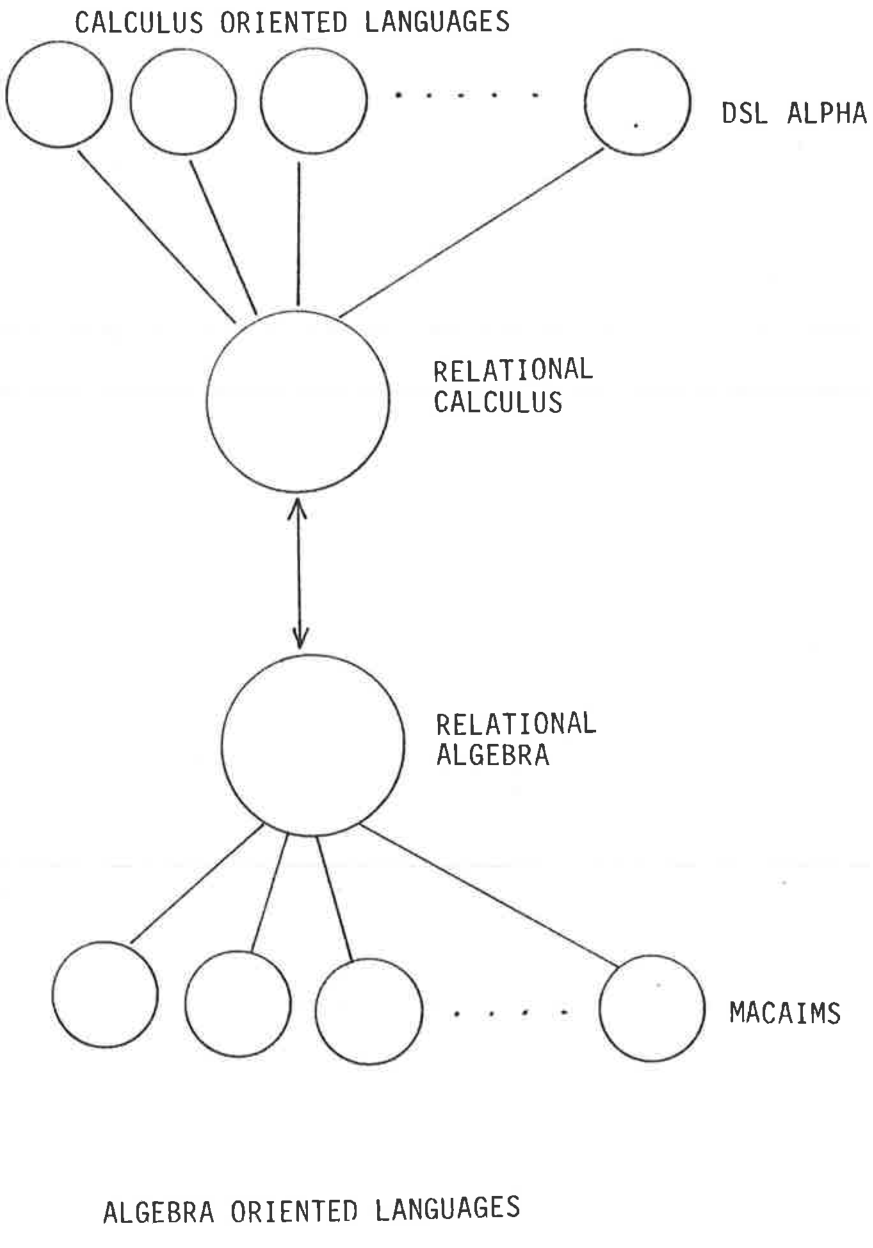


Figure 2.13: Comparison Scheme for Data Sublanguages (Codd 1971b)

2.2.5 Inverted model

Data can be thought of as points in n-dimensional space. In three dimensions a useful view of data is shown in Fig. 2.14 below

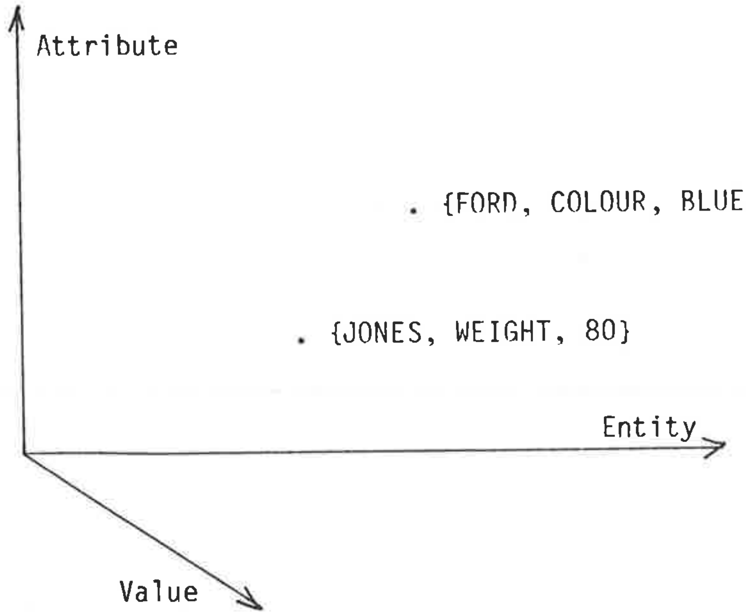


Figure 2.14

With some data a fourth dimension must be considered, that of time. Thus the attribute "weight" for entity "Jones" may have the value "80" at present but over time this may vary.

Disc and tape storage devices have one dominant dimension, based around the block concept. These devices read and write blocks and it makes sense to store commonly associated data within the same block. In geometric terms it is thus necessary to project the data points onto one of the axial planes. Thus one of the four dimensions of data is represented by blocks on a file. In traditional file systems this blocking is based on the entity dimension.

Within blocks it is usual to allocate different parts of each block to one of the other dimensions. For example within each block a particular field is used to represent the attribute dimension.

A third dimension is typically represented by some binary pattern within a field. Traditionally the value dimension is treated in this way.

The time dimension is typically represented (if at all) by either holding archival files or by having multi-valued attributes (e.g. holding 12 monthly sales figures in an inventory record).

So entrenched have these representations become that many users unfailingly select this representation for all files.

An alternate representation of data based on "inverted files" rejects this traditional method of holding data within files. It organises data primarily by attribute instead of entity.

Inverted files have been used as the basis for many databases although there is no clear cut agreement in the literature as to whether they constitute a "database model" or merely a "file organiuzation" to be used in implementing a model. Atre (1980: 280-287), Kroenke (1983: 53), Deen (1977: 174) are in the first group, while Bradley (1982:151), Tschritzis (1977b: 218-221) and Date (1977: 34) take the latter view. Whether as a data model or a file organization, inverted files are of great importance in retrieval intensive database applications, and this importance alone is strong enough for them to be considered here as a database model.

Bird (1978) cites two major strengths of inversion: rapid retrieval by multiple keys, and the ability to evaluate queries without reference to the primary file. On the other side Bird places three weaknesses: the complex file structure, the increased storage requirements and the complexity of the file maintenance process.

Inverted files have been used as the basis of database systems both in the information retrieval field and for more general applications. PRIOR (ICL 1968), PEARL (Carter 1969), ROBOT (Burns 1975) are early examples of such systems, and SYSTEM 2000 (Cohen 1978) and ADABAS (Software AG 1980) more recent examples.

Cardenas (1975), McDonell (1976, 1977), Hill (1978a, 1978b), Bird (1978), and Johnson (1982) have all analysed the performance of inverted indexes (or Associate Key Lists) while Liu (1976) has described algorithms for searching inverted files.

Inverted files can be held solely as an inverted file (e.g. PEARL) but more usually there is a main file and an index. Updating of such dual files presents a problem - some systems (ADABAS for example) maintain both in parallel, while other systems (e.g. PRIOR) have maintained only the main data file and then inverted it at intervals. As Bird (1978) points out, this latter technique is only useful for relatively static databases. Chapter 6 discusses the use of this technique for just such a "static" database (used for planning).

A second major difficulty with inverted systems is the handling of inter-record relationships. In some systems they are handled by system pointers while in many databases they are simply ignored or not implemented. This latter approach can be defended in two ways - firstly because many databases are homogeneous in nature and the handling problems are basically due to size and not complexity; secondly because the distinction between attribute and relationship is somewhat arbitrary.

Kent (1978) admits "I don't know why we should define "attribute" as a separate construct at all." He gives as an example two "facts":

. Henry Jones works in Accounting;

. Henry Jones weighs 175 pounds.

Both facts are relationships connecting entities "Henry Jones" and "Accounting" and "175 pounds" respectively. Both facts can clearly be represented by attributes or as relationships themselves having attributes:

. Henry Jones has worked in Accounting since 1970;

. Henry Jones has weighed 175 pounds since 1970.

2.3 Data Description

2.3.1 Introduction

Databases are usually described in a Data Description Language (DDL) and this description is held in a Dictionary. The Dictionary (or Directory) is a core file of most database systems and contains descriptions of the various files, records and fields in the database. Thus ADABAS has its ASSOCIATOR file (Software AG 1980) and SYSTEM 2000 has a Data Base Definition File (Tsichritizis 1977; 293). While the names are many and varied the purpose of each of these Dictionary Directory files is similar.

The data dictionary has assumed an importance both within and also external to DBMS and it is even suggested (Canning, 1981) that for some small organisations the Data Dictionary alone (without its associated DBMS) may meet most needs.

Associated with the data dictionary is the concept of the Data Base Administrator (DBA) function which has the task of maintaining the dictionary and controlling the organisation and use of the database.

The data dictionary and its associated DDL have been developed in many situations to the status of a systems design tool (BCS 1977; Bourne 1979) but this aspect of their use is beyond the scope of this thesis.

There are many different techniques for setting up the Dictionary, the three most common of which are:

Form Filling

Conversational

Data Description Language (DDL)

2.3.2 The Form Filling Approach

In this approach the Dictionary is set up by filling in forms and these forms are input to the computer and used to enter data descriptions into the dictionary.

This is a fairly simple technique and is suitable for relatively unsophisticated users. The major disadvantage of the approach is that the user has to have a supply of the forms to fill in or at least know the exact format of the input data. The system may thus be unsuitable for the casual user.

While the original input forms can be used as a visible form of the data dictionary, this is often fairly bulky and a more suitable form of documentation is often provided by a Dictionary Print Program. Alternatively the print can be produced as a by-product of the original input process.

2.3.3 The Conversational Approach

In this approach the Dictionary is set up by running an on-line conversational style program. The program asks the user a series of questions and from the responses builds up the data descriptions in the Dictionary.

Like the form filling approach this is suitable for unsophisticated users. In addition because the user merely has to respond to questions this approach is also suitable for first time users with no prior training.

The major disadvantages of this approach is the verbosity of the dialogue as the user becomes more experienced, and in addition a change to the data description can often only be made by repeating the entire conversation. This latter problem can be overcome by introducing an intermediate stage where some Data Description Language (DDL) is generated (see 2.3.4) and this in turn is compiled into the Dictionary. Minor changes can now be implemented by editing the DDL using a Text Editor and then re-compiling the DDL.

Typical of this approach is the Automatic Design Tool (ADT) of Datatrieve (DEC 1982). Using this tool the user is asked a series of questions and from the responses the ADT package builds up a set of DDL. Subsequent modifications are made by editing the DDL and more sophisticated users can go direct to DDL to describe their data.

The SEQUENT system described in Chapter 5 uses an interface similar in style to ADT but places the data description directly in the dictionary.

2.3.4 The Data Description Language Approach

In this approach the Dictionary is set up by compiling a purpose built Data Description Language (DDL).

In general this approach is best suited to systems complex enough to require a Data Base Administrator. Because of the complexity of the languages they are generally unsuitable for unsophisticated users.

PLUTO "layout strings" (ICL 1969) are an early example of the use of data description language. The string

H24NAMH26ADDR04S02MSLR12S02BALZ

describes a record with a 24 character name file (NAM) followed by up to 4 lines of an address field ADD (each of 26 characters) followed by up to 12 2 byte monthly sales figures (MSL) and finally a 2 byte balance field (BAL).

This layout string was stored in front of each PLUTO Master File and was used by PLUTO routines to access fields by name.

A more modern instance of this approach (DATATRIEVE) was referred to in the previous section, but by far the best known version of this approach is the CODASYL DBTG Schema DDL (CODASYL 1971), and this has been the principal inspiration in the development of the INVERSE and PYRAMID DDL's described in Chapters 6 and 7.

2.4 Data Base Administrator

Concurrent with the development and growing use of databases there has been a recognition that the database is a resource (Davenport 1980) that needs to be managed and this is the role of the Data Base Administrator (DBA).

Lyon (1976) points out:

> "While the nature of the DBA can be expressed in general terms, there is no universal definition of a DBA; it is unique to the enterprise."

The role of the DBA covers the following:

- design of the database;
- physical creation of the database;
- maintenance and use of the database;
- optimization of the database.

In a teaching environment the balance between the activities will be different to the emphasis placed on them in the outside world.

The performance optimization of the database is crucial in the outside world but in a teaching situation databases are rarely large enough to justify much effort in this direction.

Similarly the concern with the maintenance of the database is likely to be less strong than in the outside world. For many teaching situations the database will only be used in a retrieval mode. Where updates are used they will tend (being generally hypothetical transitions) to be small in volume and used for illustration. Rarely will updating be a major problem.

The key problems of database administration in a teaching environment are:

- what sort of database is needed - in terms of database model, record contents, inter-record structure etc.;
- where is the data to come from - so that the database looks real.

For the systems described in Chapters 6 and 7 (INVERSE and PYRAMID) it is assumed that usually the role of DBA will be undertaken by a member of the teaching staff. They will design the database, decide how it is to be used, and then build the database.

Only for the SEQUENT system (Chapter 5) would it be normal for the student to perform all functions including data definition when using the QUILL language as a stand-alone query language.

2.5 Data Manipulation Facilities

2.5.1 Introduction

Mayne (1981) defines three types of data manipulation facilities

- Host Language DML
- Report Writers
- Query Update Languages

He observes that the latter two are often combined and called a self-contained language.

Peat (1982) defines data manipulation facilities in terms of the users of those facilities rather than by Mayne's use of names describing the style and features of the language. Thus Peat refers to "programmer interface" and "end-user facilities".

The QUILL Query/Update language described in Chapter 4 has some report writer features. Mayne would thus call it a self-contained language and Peat by the term "end-user facilities". Within this thesis the term "end-user facilities" and "programmer interface" will be treated as synonyms for "self-contained language" and "host language DML" respectively.

Most (but not all) general purpose database software systems start with a host language interface and they may then add a query language at a later date.

This developmental life-cycle emerges from a primary concern with the representation of data and relationships rather than with user processing of that data. It seems almost as if the query language interface is seen as the "icing on the cake".

Thus Olle (1973) records that the CODASYL DBTG specifications do not define a query language and that they were not intended to do so. This was not because the DBTG did not believe in such capabilities, but because they saw these facilities as being on a different level from the CODASYL DML.

The CODASYL (1969) report states

> "The objectives of the Data Base Task Group in developing its proposals was to make it easier and more efficient for <u>programmers</u> to store and retrieve data...."

They went on to say

> "It is important to note that the Data Base Task Group's proposals are oriented to the programmer. It is <u>not</u> an inquiry language intended for the non-programmer...."

The CODASYL (1971) report makes the same point when it states

> "It is important to note that the Data Manipulation Language specified in this document is not designed as a universal processing language and indeed that it is not a self contained language. Rather it is an enhancement of COBOL and it can thus be categorised as a host language system. As such its level of procedurality is about equal to that of COBOL and thus it is appropriate for use in programming that large class of problems for which COBOL is the most used and most suitable language."

A status report (CODASYL 1979) on end user facilities has not yet been followed up.

Thus these database systems were clearly geared to COBOL-like programming. They failed to draw the distinction that while COBOL may be the most used language, it was not necessarily the most suitable. Recent developments in the so-called "Fourth Generation Languages" (Ashton 1982) demonstrate that other languages may be more suitable for large classes of problems.

While some systems such as RIQS (Borman 1976) only provide the self-contained interface and CODASYL (1971) only specifies a programmer interface, most database systems provide both facilities. Thus the PYRAMID system described in Chapter 7 offers both QUILL and a host language interface.

2.5.2 Host Language DML

Host Language Data Manipulation languages use a standard host programming language (e.g. COBOL, FORTRAN, PL/I) to perform all but database I/O. The database I/O is performed by causing the user programs DML commands to invoke the particular DBMS software.

In its simplest form the host DML command takes the form of a CALL to a library procedure. For example a COBOL program using ADABAS (Peat 1982: 189-204) would say

```
CALL "ADABAS" USING CONTROL-BLOCK,
                    FORMAT-BUFFER,
                    RECORD-BUFFER,
                    SEARCH-BUFFER,
                    VALUE-BUFFER.
```

The control block contains amongst other things a command code and the lengths of the other buffers.

The format buffer contains a description of the layout of the record buffer which is filled up by say a READ command. A value of "AA,5X, AB,3,V" specifies that the record buffer is to be laid out as below.

	5 spaces	
AA value 8 bytes packed		AB value 3 bytes unpacked

The search buffer specifies the record selection criteria and the value buffer contains the values used to particularize the selection expression. Thus a search buffer containing "AA,D,AB" and a value buffer with the hexadecimal value F1F2F3F4 F5F6F7F8002C will locate those records containing the AA value of 12345678 <u>and</u> the AB value of +2.

The ADABAS call interfaces with ADAMINT which is a custom module created by the Data Base Administrator (Cohen 1978). A similar technique and interface is employed by the PYRAMID system described in Chapter 7.

Some database systems provide an alternative way of writing DML which avoids the direct use of the call mechanism. The host source including the DML statements is passed through a preprocessor to convert the DML statements into host language CALL statements. While DMS 1100 and IDMS have a preprocessor, IMS and TOTAL do not (Mayne 1981). The PYRAMID system described in Chapter 7 has no preprocessor, but Chapter 8 describes how such a feature could easily be added.

2.5.3 End-User Access to Databases

Benbasat (1981) reports that it is estimated that for 95% of human/machine interactions, people costs are greater than machine costs and that actions that reduce human costs and simplify the human interface will have the greatest impact on the growth of the computer industry. This has led to the development of a whole range of end-user languages of which query languages are perhaps the most important.

While most computer professionals would recognise a Query Language if they saw one, most formal definitions, while nonetheless correct, are somewhat superficial.

Reisner (1981) defines them as "a special-purpose language for constructing queries to retrieve information from a database of information stored in the computer."

Tagg (1981) defines a Query Language as being "a high-level language, suitable for non-programming users, and oriented towards ad hoc retrieval of data with fast response."

Samet (1981) gives the definition "a high-level computer language which is primarily oriented towards the retrieval of data held on files or databases." Samet also gives what he acknowledges to be a less formal, but more satisfactory, way of telling if a package is a query language by examining certain features of the package.

Paraphrasing Samet's list in Table 2.1, there are 6 basic attributes that can be examined for features appropriate or inappropriate in a query language.

A query-update language is an extension of the query language concept that permits the user to update as well as retrieve information. In what follows the term "query language" will be taken to refer to either of the above concepts unless otherwise qualified.

Query languages are normally intended to be used by non-professional programmers. In general they have a limited number of fairly high-powered functions.

Robinson (1981) divides query language functions into the following categories:

Attribute	Appropriate	Inappropriate
Data Retrieval	On line Ad hoc Not predefined	Batch Predefined Evaluated repeatedly
Prime Users	Little or no DP experience	Specialists who build systems for others
Style of language	Specify WHAT is wanted, often in a single statement	Specify HOW to do the task
Data entry or maintenance	Limited	Unlimited
Amount of data displayed at a time	Few lines/records	Large volumes
Performance	Response and speed of development more important than run time efficiency	Run time efficiency is important

Table 2.1

- Retrieval
- Update
- Phonetic Search
- Graphics
- Boolean Operators
- Conditional Operators
- Relational Operators
- Statistical Functions
- Mathematical Functions

He divides "retrieval" into six sub-categories: Single Record, Record Collection, Combination, Quota, Grouping and Total. Single records is based on primary key, while record collection is the selection of groups of records based on conditional and boolean operators. Combination retrieval is the ability to use the output of one query as the input for another. All three of these features are available in the QUILL query language described in Chapter 4, although there are restrictions on the use of combinations retrieval in that a "hit file" has to be produced as an intermediate stage and this file then interrogated separately. Of the last three of Robinson's six retrieval functions only one is implemented in QUILL (see Chapter 4), that being total retrieval, the ability to print the entire database. Quota retrieval, which places restrictions on the volume of output, is not implemented. It is perhaps more suited to bibliographic searching, although it does have applications in accounting ("list the 10 largest outstanding debts"). Grouping retrieval collects records together with a common domain value and hence implies a sorting process. The only way to achieve this using QUILL is to produce a hit file, sort it, and then carry out a series of queries on the hit file for each value of the sorted attribute.

Yu (1978) classifies queries into three classes: Exact Match, Partial Match and Closest Match. In an "exact match" the query specifies particular values of a set of attributes that match exactly one record, for example "employee-number = 1234". A "partial match" query also specifies particular values and attributes but it is expected that many records will meet the criteria, for example "sex = male and age > 21". In a "closest match" query the search is for records which match some but not necessarily all of the chosen attributes. This type of query is found in bibliographic searches and also in searches of say criminal records. The QUILL query language provides no facilities for closest match, but concentrates on partial match. Exact match can clearly be viewed as a subset of partial match, but it is not considered here as of great importance.

Robinson defines update as being a process of changing parts of the database based on some retrieval selection process. He observes that many query languages do not permit update, and that in others (e.g. SYSTEM 2000) update is restricted to batch mode. He further states that update features are often achieved in a rather clumsy manner and are often not provided in the first version released but are added later. The QUILL language provides update facilities in a limited way, the limit being imposed more by the non-procedural nature of the language than by any implementation problems.

Phonetic searching and graphics, while desirable features, are not implemented in QUILL as they are considered to be outside the scope of the system developed.

QUILL does provide for Robinson's boolean and conditional operators, but does not have a feature for his "don't care" string matching as, apart from any customer name searching the facility is more useful for bibliographic databases.

There has been no attempt to implement the relational operators of selection, projection, join and division etc., because the mode selected for the QUILL language (see Chapter 4) precludes their implementation.

QUILL provides the add, subtract, multiply and divide operators, but does not provide exponentiation. The design objectives of the language do not permit unary minus and parentheses to be implemented.

The statistical functions provided in QUILL are SUM and AVERAGE. No mathematical functions are included - in Robinson's words they "are not an essential feature of a query language".

Most query languages require that the user views their data in a particular way from a whole range of possible views (Tagg 1983).

This conceptual view, or data model (Reisner 1981) may be thought of in several ways:

1. a single table - a file;
2. a set of tables or relations;
3. a hierarchy or tree structure;
4. a network model or graph structure.

The model chosen for the QUILL language is the single table model.

It should be stressed that this data model or conceptual view need not be the way that the data is stored. In Chapters 5, 6 and 7 it is shown that a number of different internal or physical views can be mapped onto this relatively simple conceptual view.

Set the task of describing a computer technique to solve a problem, solutions advanced tend to fall into two distinct groups. For example, suppose a group of students is asked to say how they would find the average salary of females in a payroll file.

Students with programming skills would tend to give an answer like:

1. Read the first record.
2. If it is female add the salary to a total and add 1 to a count.
3. Read the next record. If there is one go to Step 2.
4. If there are no more records divide the total by the count.
5. Print the answer.

There would be variations - some suggesting opening and closing files, some clearing the total and count (often at the wrong step!), and others putting the end of data test at some other point. Nevertheless all very similar descriptions.

Students without programming skills would by contrast tend to produce answers like:

"Find all the females, add up their salaries and divide by the number of females."

Again there will be variations on this theme, but the techniques here are quite different in style from the programmer solutions.

Thus faced with a need to allow non-programmers to access a database, two broad directions can be followed. One can teach the user to think and write programs in a procedural fashion (say using top-down design, structured code etc.) or alternatively instead of moving the user closer to the computer language the language is made more "natural" to the user's style of expression and thought. If the latter course is chosen then a so-called non-procedural language is likely to result. This user-oriented language is also likely to have more powerful functions (but often less flexibility) than conventional languages.

Thus using COBOL the following procedure division code might be produced.

```
PROCESS-QUERY.
     MOVE ZERO TO TOTAL, COUNT.
     OPEN INPUT PAYROLL-FILE,
     MOVE "YES" TO MORE-DATA.
     PERFORM READ-AND-PROCESS-DATA UNTIL MORE-DATA = "NO".
     DIVIDE TOTAL BY COUNT GIVING AVERAGE ROUNDED.
     MOVE AVERAGE TO EDITED-AVERAGE.
     DISPLAY EDITED-AVERAGE.
     CLOSE PAYROLL-FILE.
     STOP RUN
READ-AND-PROCESS-DATA.
     READ PAYROLL-FILE AT END MOVE "NO" TO MORE-DATA.
     IF MORE-DATA = "YES"
          IF SEX = "F"
               ADD SALARY TO TOAL
               ADD 1 TO COUNT.
```

Using a language like RIQS (Borman 1976) the following code might be produced.

```
BEFORE SEARCH LET T1 = 0 LET T2 = 0
BEGIN SEARCH IF #SEX = "F" LET T1 = T1 + #SALARY
       LET T2 = T2 + 1
AFTER SEARCH LET AVERAGE = T1/T2
       PRINT AVERAGE.
```

Alternatively, using the QUILL language the user could code

```
WHERE SEX = F AVERAGE AGE.
```

Query languages are often described as "procedural" or "non-procedural" but comparing the three programs above it can be seen that RIQS is less procedural than COBOL but more procedural than QUILL. It is inappropriate then to talk of "procedural" and "non-procedural" as though these terms are the two discrete values in a binary scale. Welty (1981) has commented that procedurality can be thought of as a continuous measure. To this end Welty has proposed a "procedurality metric" by which query languages may be ranked for procedurality.

Haskell (1980) lists as the advantages of non-procedural programming languages:

- they can be given machine independent semantics;
- programs can be executed in many different orders;
- program proving is simpler.

Expanding on the last point, Haskell goes on to argue that the proof for any procedural program involves transforming the program into a non-procedural equivalent form which is then proved correct. There is no known direct proof method for procedural programs.

However, as Haskell points out, all non-procedural languages compromise their semantics when dealing with system functions such as I/O. Thus users of the non-procedural language QUILL described in Chapter 4 need to be aware that in the program

```
WHERE AGE <21 PRINT NAME, SALARY
              ADD 50 TO SALARY.
```

the ADD statement is evaluated before the PRINT.

Thus Haskell concludes that "so far it has not been possible to design a system employing such a language which is entirely non-procedural."

Miller (1981) has documented an experiment in which he gave 6 different problems of varying complexity to a group of non-programmers. He analysed the responses for completeness and for the content categories of expressions (e.g. actions, attribute testing, transfer of control etc.). He found that there was very little explicit control or data definition/declaration in natural language when compared to programming languages. He concluded that there are

> "fundamental, almost incompatible, differences between natural and programming specifications of procedures. ... Changing so firmly entrenched a manner of speech is akin to asking people to change the way they walk or talk."

Benbasat (1981), Welty (1981), and Schneiderman (1978) have described similar research. Welty notes, however, that people more often write difficult queries correctly when using a procedural rather than a non-procedural language.

Thus the use of a non-procedural query language can be seen to be of value to non-programmers to help them handle simple requests of a database.

This development of languages to be more natural to the user has fostered a whole field of research in Artificial Intelligence and Natural Languages. Most of the early attempts at Natural Language are widely perceived as having failed or to be impossible (Hill 1972) but more recent results are impressive (Kaplan 1982). Using Artificial Intelligence Corporation's INTELLECT Kaplan gives the following examples.

ARE THERE ANY PEOPLE WORKING AS SECRETARIES
AND EARNING A SALARY OF $15,000 OR MORE?

GIVE ME A SORTED LIST OF NAMES OF ALL
THE VICE PRESIDENTS IN CHICAGO OR LOS ANGELES.

Njissen (1983) has also stated that INTELLECT or similar natural language interfaces are the direction in which all database access should be heading, and both Harris (1978) and Hendrix (1978) have described natural language database interfaces.

2.6 Security

Drake (1971) lists the three general ways in which a file can be damaged

- unauthorised access;
- erroneous or incomplete update;
- system malfunctions.

While there is general agreement on the above subdivision, there are considerable variations in the use of labels for each category. Thus Drake uses the terms "security" or "privacy" merely to apply to the first of the above, and Tsichritzis (1977) adopts the same use for the term "security". Date (1977) however uses "security" to refer to all three, as does Kroenke (1983).

Deen (1977) refers to authorisational operation and physical security to refer to the three types of "data protection".

This thesis adopts the convention that security is concerned with protecting a database from both unauthorized use and also unintentional destruction. The term privacy will be used for unauthorized access, even though this term is used by some to apply to the rights of human individuals, and even though others may prefer to talk about access controls, authorisation checks, confidentiality etc.

Recovery is the term used to describe processes to rebuild the database after system or program failure.

Two major privacy features are typically provided by DBMS's (Peat 1982). They are passwords and encyphering.

Passwords can be applied to various clauses in the DDL, with the implicit assumption that unless the password is quoted access to the protected clause is to be denied. The CODASYL (1971) report is perhaps the best known use of this technique. It has a multi-level system of both simple passwords and more complex procedures.

The PYRAMID system (see Chapter 7) uses passwords as in CODASYL to achieve Bonczek's (1977) "Security by view" - that is that the Database Administrator set up access routines that can only access parts of the database and the user can only look at their allocated view. The INVERSE system also provides this security by view through its selective indexing mechanisms.

Encyphering techniques are used for highly sensitive data. They have not been considered necessary either to discuss further here or to implement.

Verhofstad (1978) states

> "No single recovery techniques or series of recovery techniques can cope with every possible failure."

He describes six possible kinds of recovery:

- recovery to the correct state;
- recovery to a correct past state;
- recovery to a possible previous state;
- recovery to a valid state;
- recovery to a consistent state;
- crash resistance (e.g. after failure return to the prior state is automatic).

Verhofstad goes on to list seven categories of recovery, restart and maintenancy of consistency:

- salvation program - rescues information still recognizable - used as a last resort;
- incremental dumping - taking of back up copies;
- audit trail - recording sequences of actions on files (before and after images);
- differential files - main file is unchanged, differential file holds changes;
- backup/current version - traditional file cycling;
- multiple copies - all copies identical except during update - file marked by "back list" when updating in progress;
- careful replacement - duplicates data at the moment of update.

Verhofstad links the six kinds of desired recovery to the seven recovery techniques in a cross-reference matrix.

The only technique to recover files to the correct state is the audit trail or journal. For this reason the INVERSE system in Chapter 6 produces an audit trail journal. The use of the incremental dump technique can also be used to reduce the amount of audit trail information required to be kept. The audit trail journal contains both before and after entries (see Drake 1971, Fossum 1974, and Verhofstad 1978).

It is possible that if the INVERSE linked lists are corrupted then the situation could be improved by a purpose-built salvation program.

Harder (1979) discusses the possibility of optimizing logging and recovery in database systems.

Verhofstad (1979) has proposed that the security techniques implemented may vary at different levels of multi-level database systems.

Fossom (1974) describes the database integrity features of Univac's DMS 1100 system, including its locking and deadlock mechanisms, the rollback and quick, long and selective recovery features.

Dadam (1980) has analyzed the special problems of recovery in a distributed database and suggested checkpoint techniques that although more complex than for a central database are nevertheless necessary.

Kaunitz (1981) provides a similar but less extensive review to that of Verhofstad (1978).

2.7 Summary

This chapter has attempted to review a selection from the literature that bears on the design and construction of educational database software. The software described in Chapters 4 through 7 has been designed mostly because of, but also occasionally in spite of, the ideas found in the literature. The rejection, often reluctant, of useful ideas has usually been made on the grounds of expediency - that the construct is of limited application; is difficult to teach; is too greedy on resources; or is more difficult to implement than some alternative, though more restricted facility.

The selection of database models to be implemented has been made on expedient grounds. It has to be conceded that of the four major database models dealt with (hierarchic, network, relational and inverted) that the selection of the first and last only and the decision not to implement network and relational models is less than perfect. The network model is however often used in a hierarchic fashion for student exercises and not much is lost in implementing this subset of network facilities. The choice between relational and network/hierarchic models is more difficult (Simsion 1981, Michaels 1976, Sockut 1981). At the current time the network model is more widely used, but there is clearly a trend to the relational model. Nevertheless the decision to select a navigational model rather than the relational model is based on current market-place popularity. This choice is looked at in retrospect in Chapter 8.

The decision to implement the inverted model was much easier - it has clear advantages for retrieval intensive applications - e.g. land use databases, bibliographic databases etc.

The simplification of the three level ANSI/X3/SPARC architecture in favour of a two level architecture in the pyramid system in Chapter 7 is defended on the basis that most commercial DBMS's follow the same path. The choice still permits a sufficient measure of data independence to be implemented.

The choice between the conversational and DDL approaches to data description was also relatively easy, each being used where most appropriate (Chapters 5,6,7).

Academic staff have always had a coordinating and control role in student exercises, but with the use of databases the demand for them to act in this way is more necessary. Some consideration needs then to be given to the role of the Data Base Administrator and for any activity to decide where the boundary between academic and student should lie.

The concentration on an end-user language (QUILL) as the main data manipulation language echoes the comment by Lawrence (1979):

> "It is believed that in this area (ad hoc enquiries) that the most significant benefit of a DBMS is realised."

However, having concentrated on the end-user side, the needs of programmers has to be met with a host-language DML. Stamen (1981) has set forth some evaluation criteria for database languages.

The important (and growing) importance of security has been recognised and both privacy using Bonczeks "Security by View" and the now fairly standard audit trail features have been implemented.

Finally, Peat (1982) makes the following comment on the selection of a DBMS.

> "It should be recognised that no DBMS is 'better' than another, rather that each has its strengths and weaknesses. The object of the selection process is to find the system with the most advantages and fewest disadvantages for the envisaged EDP environment."

Thus the system described in this thesis should be judged on its advantages and disadvantages for tertiary-level education and not on its use as a commercially viable DBMS.

The major advantage of the described system is its low use of resources (both money and central memory), its simple interfaces, and adaptability to other hardware systems.

The major disadvantage is its restricted range of facilities, mostly to ensure low memory utilisation and simplicity of user interface. Again following Peat (1982):

> 'The power ... of commands is in general directly proportional to their complexity."

CHAPTER 3

METHODS AND PROCEDURES

3.1 Major Objectives

The fundamental aim in developing the educational software described in this thesis is that the student user who in later life has to use a commercial DBMS should when using the various facilities of this commercial system be able to say in effect "Ahah! I've used that sort of feature before".

To this end the software should contain in microcosm examples of most of the features found in real world systems. Reference was made to many of these features in Chapter 2 but the more important ones are repeated here.

The software should have the following features:

- it should provide physical and logical data independence;
- it should provide both a programmer and an end-user interface;
- more than one data model should be supported;
- use of resources, especially main memory, should be kept to a minimum;
- security features including privacy locks and journal files should be provided;
- the software must be capable of being taken apart and rebuilt (with some modules replaced) by, say, a student interested in software construction;
- the software must be able to be transferred from the development machine and operating system to a target user machine.

Later chapters (4, 5, 6 and 7) describe the end-user language QUILL; the stand-alone query system SEQUENT; the inverted system INVERSE and the hierarchical system PYRAMID which were built to meet the stated aims.

3.2 Choice of Programming Language

The software developed during the preparation of this thesis was written for a CDC Cyber 173 using the NOS Operating System and subsequently some of it was transferred to a DEC VAX 750 using the VMS Operating System.

Three major programming languages were available to code the system's modules; FORTRAN, PASCAL and COBOL.

FORTRAN was not used because it lacks any convenient data structure for describing records.

PASCAL has a good data structure for describing records, and its block structure and parameter passing mechanisms are good features for writing compilers. A serious drawback however is its lack of sophisticated input output such as indexed sequential files.

Eventually it was decided to write all the software in COBOL. As Evans (1982) and Triance (1978) have reported, COBOL has a number of weaknesses, but this thesis advances the view that the effect of these weaknesses need not be great, and in addition COBOL has many compensating strengths.

Evans lists the following as some of the weaknesses of COBOL.

1. It has no block structure and this makes structured programming difficult.
2. It is verbose.
3. It has no local data items.
4. Internal and external call mechanisms are different.
5. It cannot pass parameters in its internal call mechanism.

Weaknesses (1) and (3) can be overcome by adhering to particular coding standards, for example by heavy use of the PERFORM...UNTIL construction, avoiding PERFORM...THRU, using GOTO only for abort activities, and by reserving data items for specific purposes.

Weakness (2) is in part necessary so that COBOL programs are easy to read and hence maintain.

In addition COBOL has certain strengths:

- it has a well defined standard (ANSI 1974) and compilers for this standard are found on most mainframe computers;
- as COBOL is the target language for the code generators described in Chapter 7, and is the intended host language for the system, then the use of COBOL makes it possible by bootstrapping to use the current system to add new subsystems.

Wallis (1982) observes that ease of portability has been less important in the development of COBOL standards than the desire to provide permissive standards. Each COBOL standard has a life of five years, and it is not the case that each successive standard incorporates its predecessor as a subset.

The "freedom" to add extra features leads to problems in that a data name used in a legal ANS standard program is a non-standard reserved word in a compiler to which the program is transferred (Fenton 1978). A typical example of this problem was found when test program "CRCUST" (see Appendix 5) was transferred from the CYBER to the VAX. The dataname RECORD-NAME which was acceptable on the CYBER was rejected by the VAX compiler.

Wallis (1982) states that because many COBOL features are left to be "implementor defined" and further that there is substantial freedom to pick and choose features for subsets, the portability of COBOL has been seriously compromised. Thus the 1974 standard specifies a nucelus and eleven modules of the standard, each of which modules can be implemented at different levels. There are thus more than 100,000 versions of "standard" 1974 COBOL.

Similar problems exist with FORTRAN, particularly with respect to character handling. Thus Fenton (1978) says about both COBOL and FORTRAN "no two compilers accept precisely the same language. Indeed no compiler accepts the standard, the whole standard and nothing but the standard."

However, whilst accepting that some COBOL compilers have non-standard features, Norman (1978) has observed

"... experience has shown that the best results are obtained when the (COBOL) language is used in a disciplined way."

Part of the discipline is the selection of the original compiler to develop the software. Fisher (1978) in ranking eleven COBOL compilers for portability ranks the top 3 as:

1. IBM - extremely good.
2. CDC - very good and strictly according to the standards (non-ANSI flag good).
3. DEC - very good (System/10).

The U.S. Navy (1978) ranks the CDC COBOL Compiler, Version 4.2 as the most portable and comments that it is "virtually perfect".

The choice then of the CDC COBOL compiler, while not guaranteeing portability, does offer perhaps better prospects than any other language and compiler.

The COBOL Environment Division is and always will be a problem (Fisher, 1978).

The advice of Fisher has been followed that "the only 'reliable' data type is DISPLAY". This data type has been used wherever possible, and an attempt has been made to avoid use of data types that are dependent on the word length of the CYBER.

All code in the system has been written and tested using the CDC COBOL-5 compiler (CDC 1978). The compiler option ANSI=AUDIT has been used to verify that constructs not included in the ANSI standard (ANSI 1974) are rejected by the compiler. Thus the code should be used on other computer systems with minimal conversion effort.

Because it is intended that students may dissect and/or modify the code, the following coding conventions have been adopted to make the code easier to follow.

1. All names are as self-explanatory as possible, even at the expense of verbosity.
2. The code has been laid out in accordance with the top-down design of each software program. Thus the paragraphs of each program are coded top-down, left-to-right. For example, given the paragraph hierarchy of Fig. 3.1, the order of the paragraphs is A, B, C, D, E, F and finally G. An exception is made in the case of paragraphs called more than once. These are placed at the end of that part of the hierarchy in which they are used. Thus in the hierarchy shown in Fig. 3.2, the order of the paragraphs is A, B, C, D, E, F, G and finally the common paragraph H.

3.3 Software transfer

Mention has already been made that the software described here was developed on a CDC CYBER 173 and then transferred to a VAX 750. The software consists of about a dozen large COBOL subprograms which are linked in various combinations to form the various software programs. There are in total over 7000 lines of code. Accordingly, while structure diagrams and sub-program diagrams are included within this thesis, the 150 or so pages of software compilation listings are not. It is felt that to include the code would add little to an understanding of what has been achieved. Further, nobody should attempt to implement major software packages by keying in copies of code from an appendix. If any potential user requires the code it is available both on magnetic tape and also on the diskettes on which it was successfully transferred from the CYBER to the VAX.

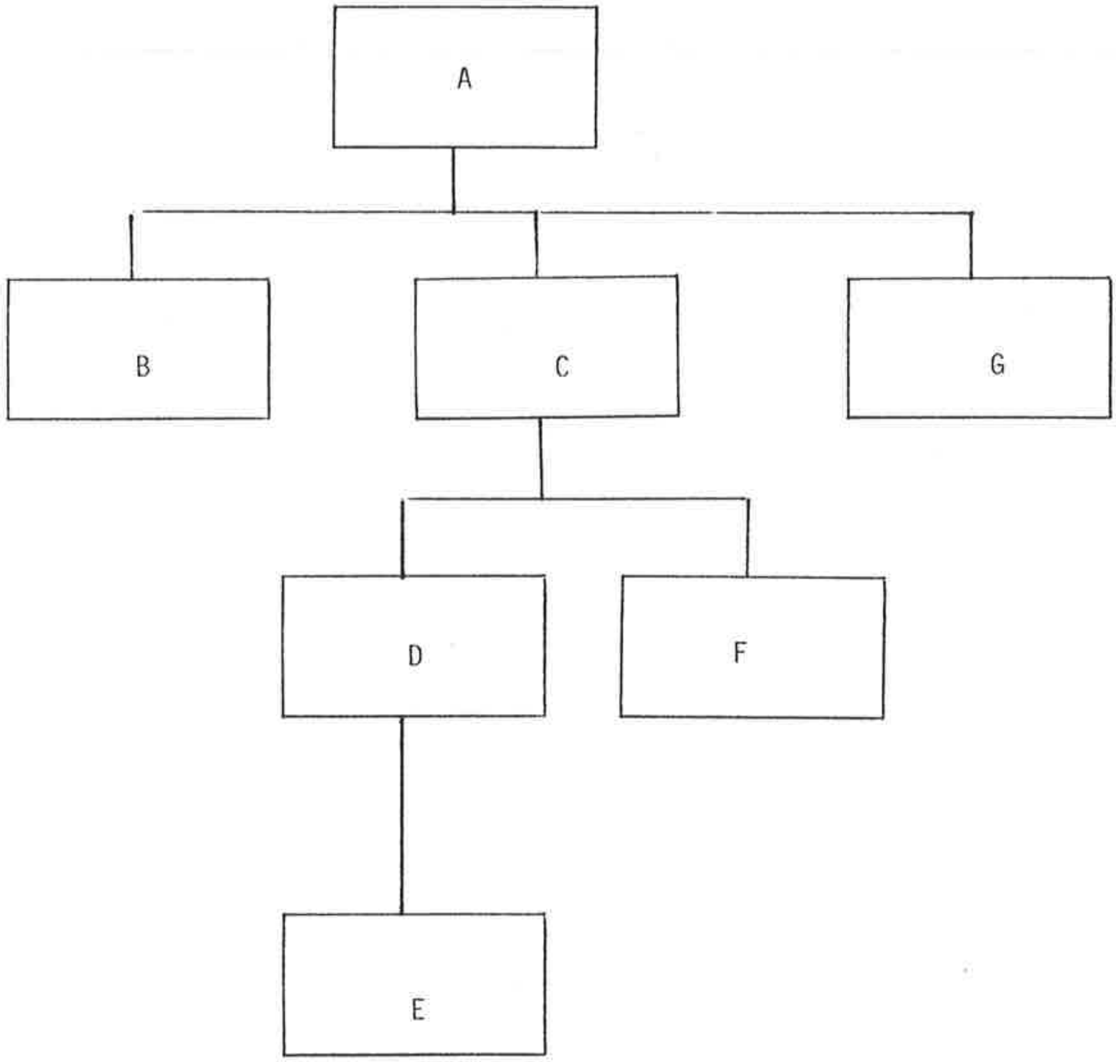


Figure 3.1: Typical module hierarchy (1)

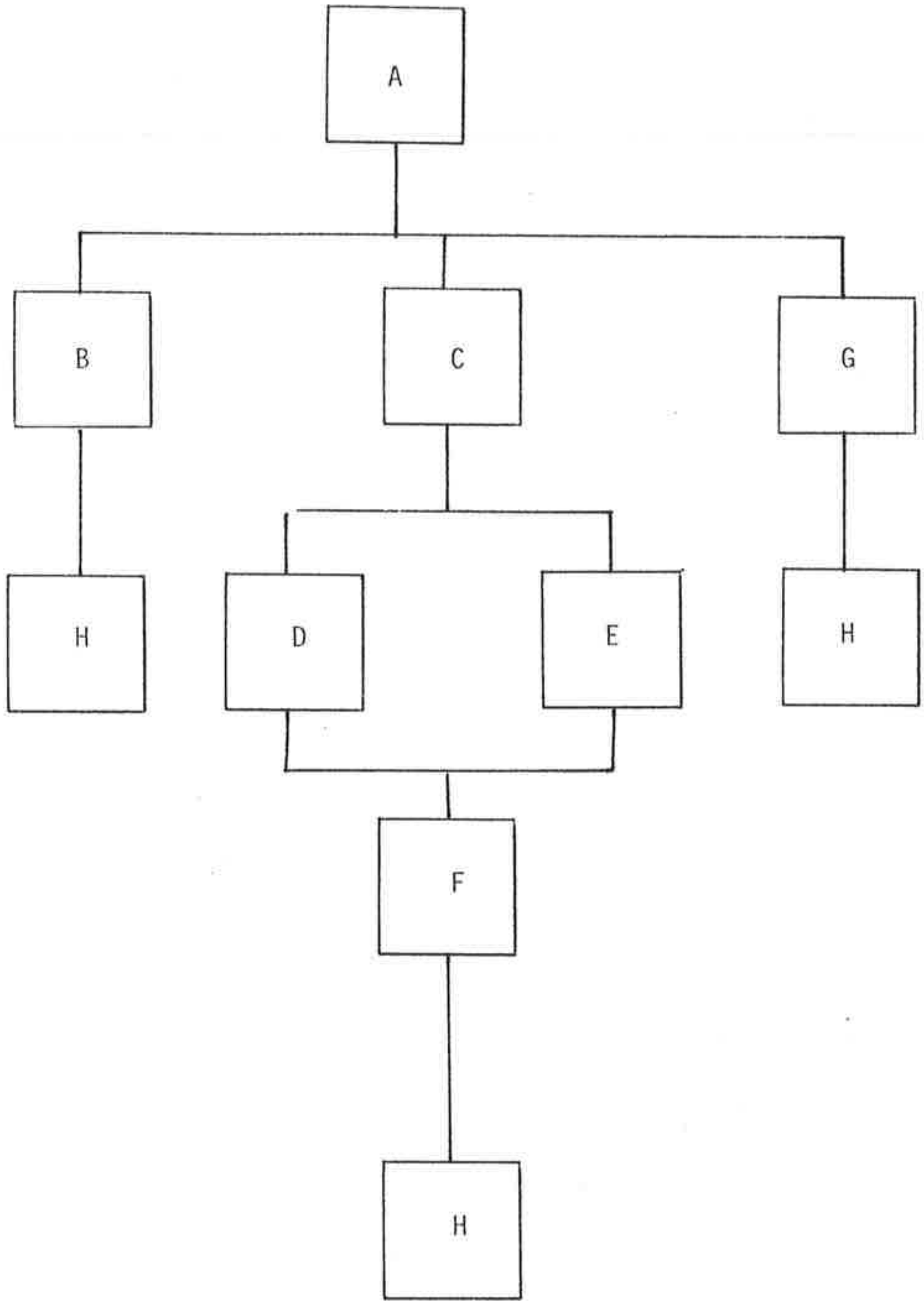


Figure 3.2: Typical module hierarchy (2)

3.4 Use of Examples

The various software facilities developed for this thesis are described in Chapters 4 through 7. Each feature of the software is described through examples. The selection of examples has attempted to steer a middle path between the two extremes of a single complex all-embracing example and a disjointed set of simpler examples particularly suited to the feature being discussed. The former approach would allow a consistent thread to be maintained but the use of certain features for the single application may defy reality and stretch credibility. The latter approach enables an easier case to be made for any specific feature but tends to obscure the integrating nature of any particular database.

3.5 The Lexical Analyzer

Most of the programs in the software are in fact compilers. This subprogram is a central part of all such compilers in the system. It is invoked by the calling sequence

```
          CALL "LEXAN"
             USING FUNCTION, SYMBOL, SYMBOL-TYPE,
                 NUMERIC VALUE.
```

The basic purpose of the Lexical Analyzer is to read source lines, break them down into symbols, and present the symbols one at a time to the calling program.

Symbols may be separated by any number of spaces. They must be wholly contained on one source line.

The symbol types processed are:

String - any sequence of characters enclosed by quotes ("   "). The maximum length of a string is 64 characters.

Identifier - any sequence of characters from the set A through Z, 0 through 9 and hyphen (-). The first character must be a letter. A hyphen can only appear between two other non-hyphen identifier characters. The maximum length of an identifier is 20 characters.

Number - a string of decimal digits, 0 through 9, with a leading optional sign (+) or (-) and an optional decimal point (.). If the decimal point appears it must not be either the first or the last character of the number.

Letter - a single character from the set A through Z .

The input parameter is FUNCTION which can take the following values:

Spaces - the next symbol (irrespective of type) is returned. The parameter SYMBOL-TYPE as set to "STRING", "IDENTIFIER", "NUMBER" or "SEPARATOR" as appropriate. The value in SYMBOL is the characters of the string (not including the quotes), the identifier or the separator. For a number SYMBOL contains the character by character value as it appears in the source text, and NUMERICVALUE contains the actual signed value as an 18 digit number with 9 decimal places. A separator is a single character which is not A through Z, 0 through 9, "space", "+", "-" e.g. a punctuation character.

LETTER - if the next non-space character is a letter then this is returned, otherwise a space is returned in SYMBOL.

LIST - starts listing the source from the next source line.

NOLIST - stops printing the source after the current source line.

LENGTH - NUMERIC-VALUE is assumed to contain the character position on the source line where unpacking of symbols is to cease. The default value is 73.

FINISHED - the source file is closed and symbol processing finishes.

The output parameter SYMBOL-TYPE is set to one of the following values:

IDENTIFIER

STRING

LETTER

NUMBER

SEPARATOR

In all compilers the mode statement

$$\underline{\text{MODE}}\ \text{IS} \left\{ \begin{array}{l} \underline{\text{BATCH}} \\ \\ \underline{\text{INTERACTIVE}} \end{array} \right\}$$

establishes the processing mode for the compilation. If the clause is not specified then MODE IS INTERACTIVE is assumed.

In batch mode source lines are read from the system file "INPUT" and are echoed on system file "OUTPUT" along with any appropriate compilation errors and/or messages. Any compilation error found during the compilation will cause the entire compilation to fail after syntax and semantic checking has been completed.

In interactive mode the system files "INPUT" and "OUTPUT" are assumed to be an interactive terminal. No echoing of source lines takes place, and any compilation errors are assumed to be immediately corrected and hence the compilation is not aborted.

CHAPTER 4

QUILL QUERY LANGUAGE

4.1 Introduction

The QUILL Query/Update language is the high-level or end-user interface to the system. The language is designed to be used by non-programmers in an interactive fashion, although it can also be used by programmers and can also be run as a batch system.

The design principles for the language are those suggested by Bonczek (1977):

- the language is independent of the database;
- programming expertise is not required to access the database;
- the language is non-procedural;
- the language is easily extendable.

The independence of the language from the database is such that the same language is used to access three fundamentally different types of database - a sequential file, an inverted database and a hierarchic database.

Each of these three internal physical views is mapped onto a single conceptual view, or data model (Reisner, 1981). For QUILL this conceptual view is of a single table or file with each record of the file containing the same fixed format fields. The language allows the user to manipulate the database through this conceptutal view and mapping routines translate these activities into the operations required in the particular database.

Programming expertise is not required to access the database as using QUILL the user can retrieve data, produce reports and (depending upon the particular physical database) can update data. Thus for a whole range of data processing tasks the QUILL language can be used rather than a conventional programming language such as COBOL.

The QUILL language is non-procedural and using the procedurality metric of Welty (1981) the language is much closer to the non-procedural extremity of the procedural ↔ non-procedural scale than most query languages. The QUILL query or statement is specified as a series of actions and these actions can be written by the user in any order, with all such combinations being by definition semantically equivalent and hence producing the same result.

The language is easily extendable such that since its original conception and implementation various different physical database models have been accessed via QUILL, and in addition several arithmetic operations have been added without any significant changes being made to the existing code.

4.2 Language details

Operations using QUILL consist of a sequence of statements. The statements are actioned individually so that when used interactively input of statements alternate with actioning those statements.

Each statement takes the form

WHERE search-predicate action-1 --- action-n.

The search predicate may be a simple or a complex boolean expression and the actions consist of printing, displaying, updating, totalling and extracting specified fields from the selected records. The actions may (except for printing page control) be written in any order without affecting the result of the statement. The full syntax for the language is given in Appendix 1.

The facilities of the QUILL language are shown in the following examples.

WHERE SEX = M PRINT AGE.

will print on the line printer all records with the SEX field containing M (males).

A more complex boolean expression may be given

WHERE SEX = M AND AGE < 21 DISPLAY NAME.

which will display on the screen the names of all records with both a SEX value of M and an AGE value less than 21.

Where 3 or more conditions are given the question of operator precedence is raised. AND and OR are treated as of equal precedence, and parentheses are also allowed to indicate the order of evaluation.

WHERE SEX = M AND (AGE < 18 OR AGE > 64)...

will retrieve say males not aged between 18 and 64 inclusive.

The < and > can also be written LESS THAN, GREATER THAN as in the following example

WHERE AGE IS GREATER THAN 64 ...

The negation operator can be used as in

WHERE AGE NOT > 64 ... or in

WHERE AGE IS NOT GREATER THAN 64 ... etc.

For the equal and not equal tests two or more values can be OR'd together in the same condition. For example, the QUILL user can write

WHERE SEX NOT = M OR F DISPLAY NAME, SEX.

which will display the NAME and SEX values of any record not correctly classified as M (male) or F (female).

While it is sensible to allow the user to write

WHERE GRADE = 2 OR 3 ...

it is clearly not sensible to allow

WHERE AGE <30 OR 35 ...

and therefore only = and NOT = can be followed by multiple values.

One of the design features of the language is that character values may be, but need not be, enclosed in quote characters. This allows the user to avoid the unnatural string concept unless embedded spaces or special characters appear in the value. The user can thus write

WHERE TITLE = IOLANTHE DISPLAY AUTHOR.

WHERE TITLE = "PIRATES OF PENZANCE" DISPLAY AUTHOR.

WHERE CATEGORY = COLOUR OR AGE DISPLAY ID-NO.

To permit the last of these three examples causes problems in the interpretation of the symbol OR. Consider the query

WHERE CATEGORY = COLOUR OR AGE < 10 DISPLAY ID-NO.

Either the QUILL query syntax analyzer must look ahead; or the OR must be interpreted as connecting this condition, or connecting two values for a single condition. To resolve this problem the last of these interpretations has been used and thus in the last example given a syntax error is given on encountering the < symbol as the symbol AGE has been taken to be a test value for CATEGORY. This example can be rewritten

WHERE (CATEGORY = COLOUR) OR (AGE < 10) DISPLAY ID-NO.

and the ambiguity is resolved.

The actions PRINT and DISPLAY follow the "tabular" and "list" structures of Samet (1981). Thus the statement WHERE SEX = M PRINT AGE, NAME will produce the following style of output in a printer file

```
SMITH          27
JONES          43
WILSON         17
```

whereas the statement WHERE SEX = M DISPLAY AGE, NAME will produce the following style of output on the screen

```
AGE      =    27
NAME     =    SMITH

ENTER S TO STOP DISPLAY.  PRESS RETURN
```

Thus PRINT is intended for high volume printed output, and DISPLAY for low-volume on-line output.

In the action PRINT A, B, C the fields may be separated by spaces, commas or the AND symbol. If desired the field list may be enclosed in parenthese as in PRINT (A,B,C). This latter form can overcome the ambiguity between the actions PRINT A B DISPLAY C where DISPLAY is taken as the key word of an action and thus A and B are printed and C is displayed. However PRINT (A B DISPLAY C) will treat all of A, B, DISPLAY and C as field names.

Returning to the action PRINT A, B, C the three fields are printed by default with two spaces between them. It is possible to over-ride this default as in the action PRINT A SPACE 5 B SPACE 6 C.

If the number of characters to be printed exceeds one line then a fresh line is started with the first field that cannot fit onto the current line.

Headings can be printed by the use of the HEADING action. Thus the statement

```
WHERE AGE > 17 PRINT AGE SPACE 3 SEX
SPACE 3 NAME
HEADING "AGE   SEX   NAME".
```

will produce output of the form

```
AGE      SEX      NAME
18        M       SMITH
21        F       JONES
19        M       WILSON
```

Headings are assumed to start at line 1 column 1 unless otherwise specified. Greater control can be obtained by the use of line and/or column numbers as in the statement

```
WHERE AGE > 17 PRINT AGE NAME
HEADING "AGE NAME" ON LINE 1
HEADING "--- ----" ON LINE 2
HEADING " " ON LINE 3.
```

will produce output of the form

```
AGE      NAME
---      ----

18       SMITH
21       JONES
19       WILSON
```

The statement

```
WHERE AGE > 17 PRINT SPACE 20 NAME
HEADING "NAME" AT COLUMN 21
HEADING " " ON LINE 2.
```

will produce a column of names in column 21 as below

```
NAME

SMITH
JONES
WILSON
```

The CONTROL action can be used to set up to control the page and display screen layouts. Thus the actions

```
CONTROL PAGE WIDTH 120
CONTROL PAGE LENGTH 50
CONTROL PAGE NUMBER 100...
```

will print 50 120-character lines per page (including headings) and will number pages at column 100 of line 1 of each page heading.

Other controls available are for example

```
CONTROL DISPLAY WIDTH 75
CONTROL DISPLAY DEPTH 20
```

While these report writer features are probably sufficient for most student use, more sophisticated reports can be produced by using QUILL to produce an extract file, and then processing this extract file using a conventional program or report writer utility. For example the QUILL user can write

```
WHERE SEX = M EXTRACT NAME AGE SALARY.
```

and a file will be produced with the selected fields (and no others) for all males in the database.

QUILL is also able to process simple update operations using the ADD, SUBTRACT, MULTIPLY, DIVIDE, INCREASE, DECREASE and SET actions.

The ADD arithmetic operation has the same syntax as COBOL, thus

WHERE AGE = 18 ADD 30 TO WAGE

adds 30 to the WAGE field for all those records with the AGE field equal to 18.

The selection of records is performed prior to the update operation, thus the QUILL statement

WHERE GRADE = 3 ADD 1 TO GRADE

will result in all selected records having a grade of 4. Thus no records will have the value 3 after this statement.

The MULTIPLY arithmetic operation has a different syntax from COBOL

WHERE AGE = 18 MULTIPLY WAGE BY 1.05.

COBOL uses the form MULTIPLY 1.05 BY WAGE adopting the convention that the last field name receives the result. Thus in COBOL the statements ADD A TO B and MULTIPLY A BY B both place the result in B.

In QUILL, however, each arithmetic operation involves a single variable and a literal, with the result being placed in the variable. The ambiguity of COBOL is thus avoided (along with some of the power of COBOL) and in QUILL the more natural form of the MULTIPLY syntax can be employed.

The INCREASE arithmetic operation is for some end-users a more natural form of expression than ADD or MULTIPLY.

Consider the following QUILL update statements

WHERE AGE < 18 INCREASE SALARY BY 15%

compared to the equivalent statement

WHERE AGE < 18 MULTIPLY SALARY BY 1.15.

The QUILL interpreter processes both of these statement identically and this allows the user to choose the (to them) more natural form of expression.

Again consider

WHERE AGE < 18 INCREASE SALARY BY 500

compared to the equivalent statement

WHERE AGE < 18 ADD 500 TO SALARY.

The DECREASE operation is an alternative to SUBTRACT or MULTIPLY. Thus the QUILL update statement

WHERE COST < 18 DECREASE PRICE BY 10%.

is interpreted identically to

WHERE COST < 18 MULTIPLY PRICE BY 0.90.

and the staterment

WHERE COST < 18 DECREASE PRICE BY 5.

is the same as

WHERE COST < 18 SUBTRACT 5 FROM PRICE.

The final arithmetic operation is the SET action. The QUILL statement

```
WHERE AGE = 17 SET SALARY TO 8000
               SET GRADE TO X.
```

will replace the current value of the SALARY and GRADE fields with 8000 and X respectively.

When an arithmetic action and an output action are combined in the same statement, the order in which the actions are defined (by the QUILL language) to be carried out is of significance. Consider the statements

```
WHERE SALARY < 10000 INCREASE SALARY BY 1000
                     PRINT NAME, SALARY.

WHERE SALARY < 10000 PRINT NAME, SALARY
                     INCREASE SALARY BY 1000.
```

If these two statements are required to be semantically equivalent, then in both cases either the print or the increase action must be performed first, and the QUILL system in fact chooses the latter option, performing arithmetic before output. Thus the above two statements may print salaries that no longer meet the selection criteria of the search predicate.

Continuing this theme, a further problem arises when several arithmetic actions appear in the same statement. Thus consider the statements

```
WHERE A = 10 ADD 1 TO B MULTIPLY C BY 3.

WHERE A = 10 ADD 1 TO B MULTIPLY B BY 3.
```

The actions in the first statement are clearly order independent, while those of the second are not. For this reason QUILL restricts arithmetic operations to one per field in any statement, even if the arithmetic actions are commutative. However

WHERE A = 10 ADD 3 TO B SUBTRACT 1 FROM B

can be clearly rewritten with a single action ADD 2 TO B and so these multiple commutative actions are transformed into a single action.

4.3 Implementation

The QUILL language is implemented in the source module QLSCE and this module communicates via a standard COBOL CALL-interface with the SCAN module to access the database (Figure 4.1).

The SCAN module exists in two versions

- SCANSQ for sequential files and hierarchic databases;
- SCANIV for inverted databases;

The call to the SCAN module in the QLSCE code is as follows

```
CALL "SCAN" USING SEARCH-FUNCTION,
                  CONDITION-COUNT,
                  CONDITIONS,
                  VALUE-COUNT,
                  TEST-VALUES,
                  RETRIEVE-LIST-LENGTH,
                  RETRIEVE-FIELDS,
                  BUFFER,
                  SEARCH-STATUS.
```

The SEARCH-FUNCTION can take the OPEN, CLOSE, FIND, GET, PUT. The function OPEN and CLOSE are used to open and close the database. FIND is used to initialise the search process for a new query. For some SCAN modules (e.g. SCANIV) the searching and selection of records is done here, while for others (e.g. SCANSQ) the data is merely (re-)positioned at the start. The GET function presents the calling routine with a single record matching the search criteria, while PUT returns an updated record.

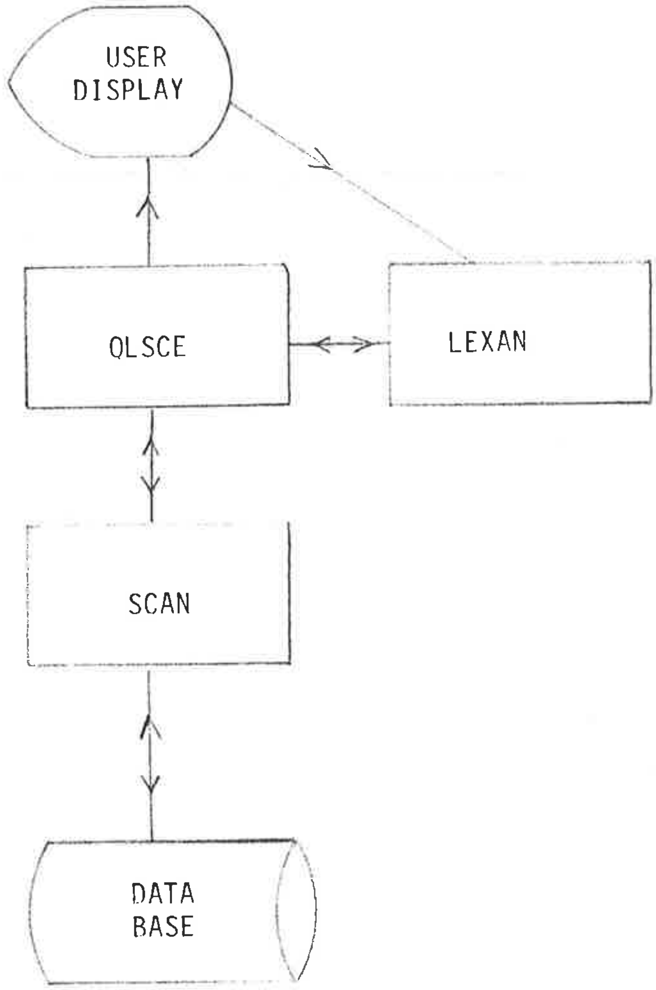


Figure 4.1: The QUILL system chart.

The CONDITIONS are a table with one entry for each condition of a search predicate. Each entry in the table has 6 components: LEVEL, CONNECTOR, FIRST-VALUE, NO-OF-VALUES, TEST-FIELD and TEST-TYPE. The LEVEL is an integer representing the depth of a condition within nested parentheses. A value of 1 indicates a condition not enclosed in parentheses, 2 within a single pair, 3 within a double pair, etc. The CONNECTOR is used in the second and subsequent entries in the table to connect the entry to its predecessor. It can take the value "A" for AND or "O" for OR. Thus using LEVEL and CONNECTOR nested queries of arbitrary complexity can be specified. FIRST-VALUE is the relative address within the TEST-VALUES of the one or more values (specified by NO-OF-VALUES) that the TEST-FIELD is to be compared to. Finally TEST-TYPE can take the values "EQ", "NE", "LT", "LE", "GT" or "GE" representing "equal", "not equal", "less than", "less than or equal to", "greater than", and "greater than or equal to". Only EQ and NE may have NO-OF-VALUES greater than 1. After the design of this table driven system was completed, a similar but less powerful tabular technique was found to be described by Cagan (1973).

The TEST-VALUES are a table of values (both numeric and character) that particular fields are to be tested against.

The RETRIEVE-FIELDS are a table with each entry having 4 components: RETRIEVE-FIELD-NAME, RETRIEVE-FIELD-POSITION, RETRIEVE-FIELD-LENGTH and RETRIEVE-FIELD-TYPE. The RETRIEVE-FIELD-NAME is filled in for each field to be retrieved, and the SCAN module returns the position, length and type of the retrieved field. The position is a relative character positio (1...n) within BUFFER.

Finally SEARCH-STATUS is normally set to spaces, but is set to "NO MORE" by SCAN when no more records can be returned. Any other value of SEARCH STATUS indicates an error.

From the structure diagrams for QLSCE (Appendix 2) it can be seen that the basic action is to process a number of statements, and that each statement consists of the two steps: "get statement" and "action statement".

"Get statement" consists of "get conditions" and "get actions". "Get condition" scans the boolean expression for the search predicate and from it builds up the CONDITIONS and TEST-VALUE tables. "Get actions" processes the action clauses of the statement and records these details in various action lists: retrieve list, arithmetic list, sum list, print list, display list and extract list.

"Action statement" locates and then retrieves records from the data base using the scan module. It then moves through the action lists in the order arithmetic, sum, display, print and extract and carries out the appropriate action. This action sequence is thus not dependent upon the order of specification of the clauses in the statement.

CHAPTER 5

SEQUENTIAL FILE QUERIES (SEQUENT)

5.1 Introduction

The QUILL language can be used as a stand-alone query language. In this mode of operation (called SEQUENT) the user can process files using conventional programming techniques and intersperse these operations with the use of the query language.

There are two stages to this process (see Fig. 5.1). First a Dictionary file must be set up describing the field formats of the records in the file, and secondly the QUILL query language is run using both the users file and the previously created Dictionary. On the CYBER these two activities are controlled by the SEQUENT CCL procedure.

5.2 Dictionary Creation

Because users of this facility are more likely to be less sophisticated users than the users of the Inverted and Hierarchical databases, it is essential that the setting up of the Dictionary should be as simple as possible. Thus the use of a Data Description Language is avoided and instead data is described to an on-line conversation style program.

The CYBER SEQUENT CCL procedure call

SEQUENT, DEFINE

invokes the Dictionary Set-Up program SBUILD and initiates the interactive dialogue.

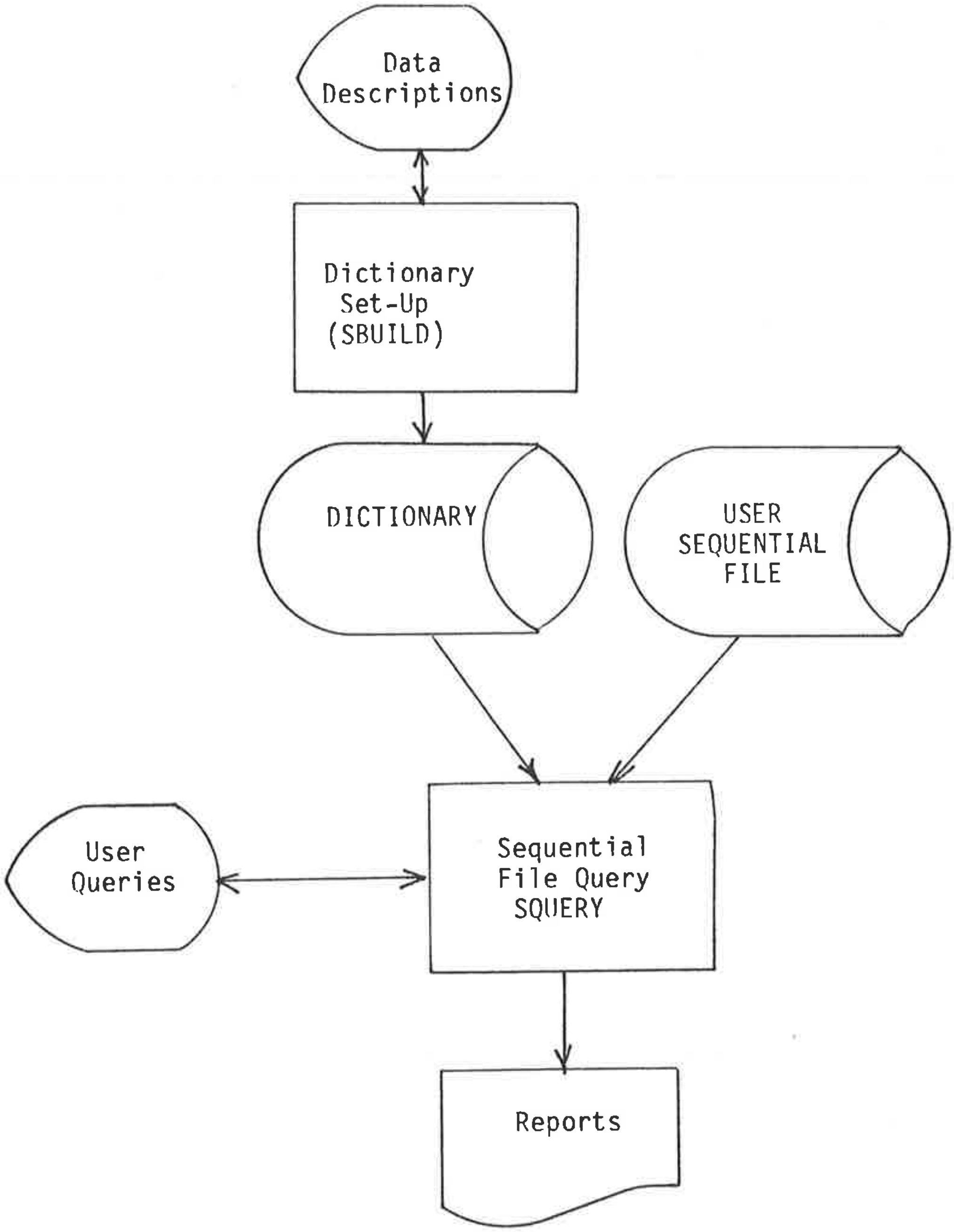


Figure 5.1: SEQUENT System Structure

For example, consider the following sequential file record layout

Columns	Contents
1 - 4	4 digit employee-number
5	Sex (M or F)
6	Marital Status (S, M, W, D)
7 - 9	Hourly pay rate $.¢¢
10 - 29	Surname) Name
30 - 33	Initials) Name
34 - 53	Maiden Name
54 - 80	Not used

The full dialogue of the Dictionary Set-Up program is included in Appendix 3. Some extracts from this dialogue are shown below so that the facilities of the Dictionary Set-Up program may be discussed.

A numeric field (e.g. hourly pay rate) is set up using the following dialogu

```
ENTER FIELD NAME
? PAY-RATE
ENTER FIELD TYPE - C(CHARACTER) OR N(NUMERIC)
? N
ENTER LENGTH OF FIELD (3 DIGITS)
? 003
ENTER NUMBER OF DECIMAL PLACES (1 DIGIT)
? 2
ENTER FIELD POSITION (4 DIGITS FROM 0001)
? 0007
```

At each stage of the above dialogue the response is validated, and if an error is detected then an opportunity is given for the user to repeat their response.

When all responses have been made the information keyed in is echoed to the user and they are asked to confirm whether or not they wish to add the field to the Dictionary. For the example above this confirmation dialogue is as follows:

```
FIELD   NAME         PAY-RATE
FIELD   TYPE         NUMERIC
FIELD   LENGTH          3
DECIMAL PLACES          1
FIELD   POSITION        7

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY
? Y
```

A character field follows the same pattern as that shown above for a numeric field. The only difference is that "decimal places" are not asked for in the dialogue or echoed in the confirmation.

There is no restriction on how the record is broken up into fields other than that all names are unique. In particular a part of the record may be redefined. Thus columns 9 to 32 of the record can be described twice, once as NAME, and then effectively redefined as SURNAME and INITIALS. This allows users to write queries of the form

```
WHERE SURNAME = SMITH PRINT NAME
```

or

```
WHERE YEAR-BORN < 43 PRINT DATE-OF-BIRTH
```

Another use of this facility allows users to process alternative record descriptions. Thus for the record described above the field "maiden name" may only be present for married women and could be used as below

```
WHERE SEX = F AND MARITAL-STATUS = M
   PRINT EMPLOYEE-NUMBER, MAIDEN-NAME.
```

5.3 Sequential File Queries

SEQUENT queries can be invoked in two ways

```
SEQUENT, QUERY          (on-line)
SEQUENT, QUERY, I=DATA  (batch from file DATA)
```

The module structure of the query program SQUERY is shown in Fig. 5.2. The QLSCE module is the standard query language module for the QUILL query language. The same module is used for query programs IQUERY (Inverted database) and PQUERY (Hierarchic files). Likewise the Lexical Analyzer module LEXAN is common to all three query programs.

The module SCANSQ is common both to programs SQUERY and PQUERY. (Program IQUERY contains a different module SCANIV which is described in Chapter 6.)

SCANSQ performs the record selection defined by the call from QLSCE (see Chapter 4, Section 3).

Module SCANSF is called firstly to open the file (and read the field descriptions from the dictionary), and secondly to read the next record from the file.

Because SCANSQ operates in a read-only mode, any update operation specified in QUILL is passed down by QLSCE to SCANSQ, but is then ignored. Since however QLSCE carries out all updating and printing from its own buffers, printed output will appear to have been updated. Thus

```
WHERE SALARY < 8000 ADD 500 TO SALARY
                    PRINT NAME, SALARY.
```

will print the update salary and not the original salary. The file, however, will not have been changed.

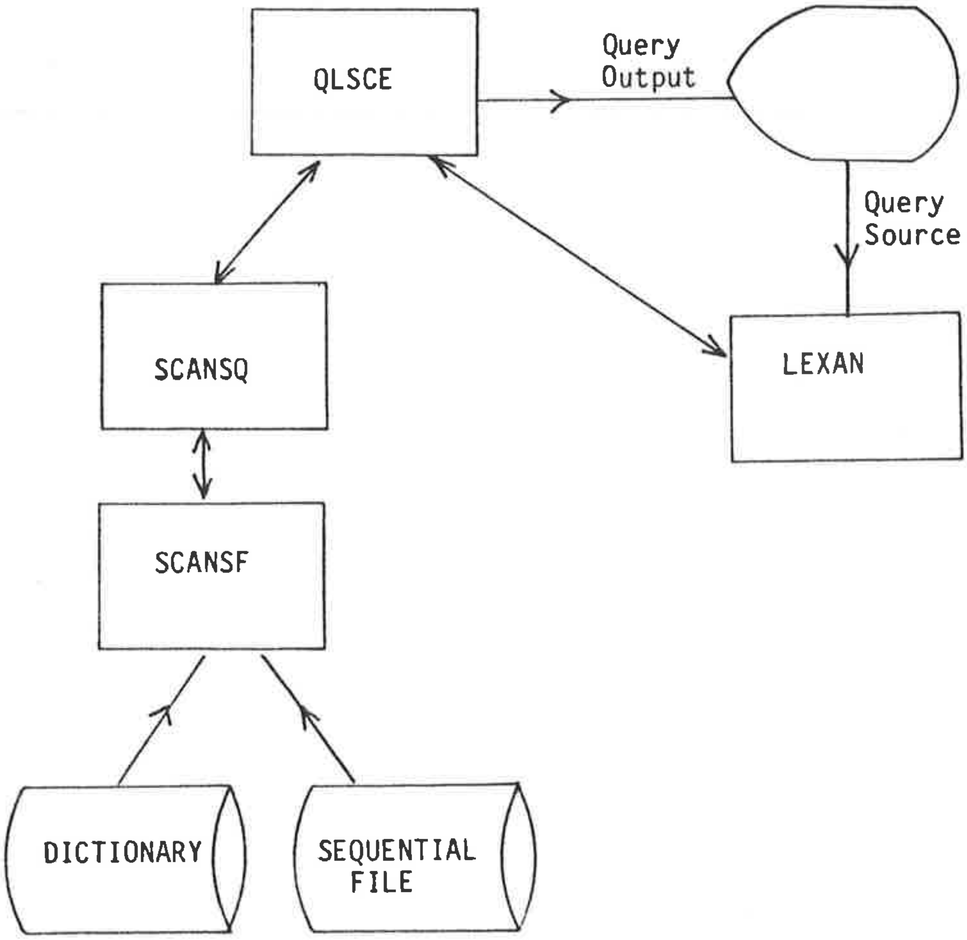


Figure 5.2: SEQUENT Query Program SQUERY Module Structure

CHAPTER 6

INVERTED DATABASE SYSTEM (INVERSE)

6.1 Introduction

An INVERSE database consists of a single data file coupled with one or more inverted index files.

The data file can be used as a stand-alone file or using the QUILL query language it can be accessed through one of the index files. Each of the index files includes both a Data Dictionary describing selected fields of the user records and indexes to some of these selected fields. There may be several such index files, each one representing a different user view in the multi-user system.

There are two basic components of this system (see Fig. 6.1). First the index file is created, and second the QUILL language is used to interrogate and update the data file through the index file. On the CYBER both activities are controlled by the INVERSE CCL procedure.

A typical application for which the INVERSE system is suited is Financial Planning or Town Planning where a large database is to be browsed over say a period of 3/4 weeks. During this period of ad hoc enquiries it is expected that the database will not change so that a frozen (but nevertheless reasonably up-to-date) view of the enterprise can be used to plan management decisions. Appendix 4 contains just such an example from the Town Planning area. Some examples from that database, and also from a personnel database are used as illustrations within this chapter.

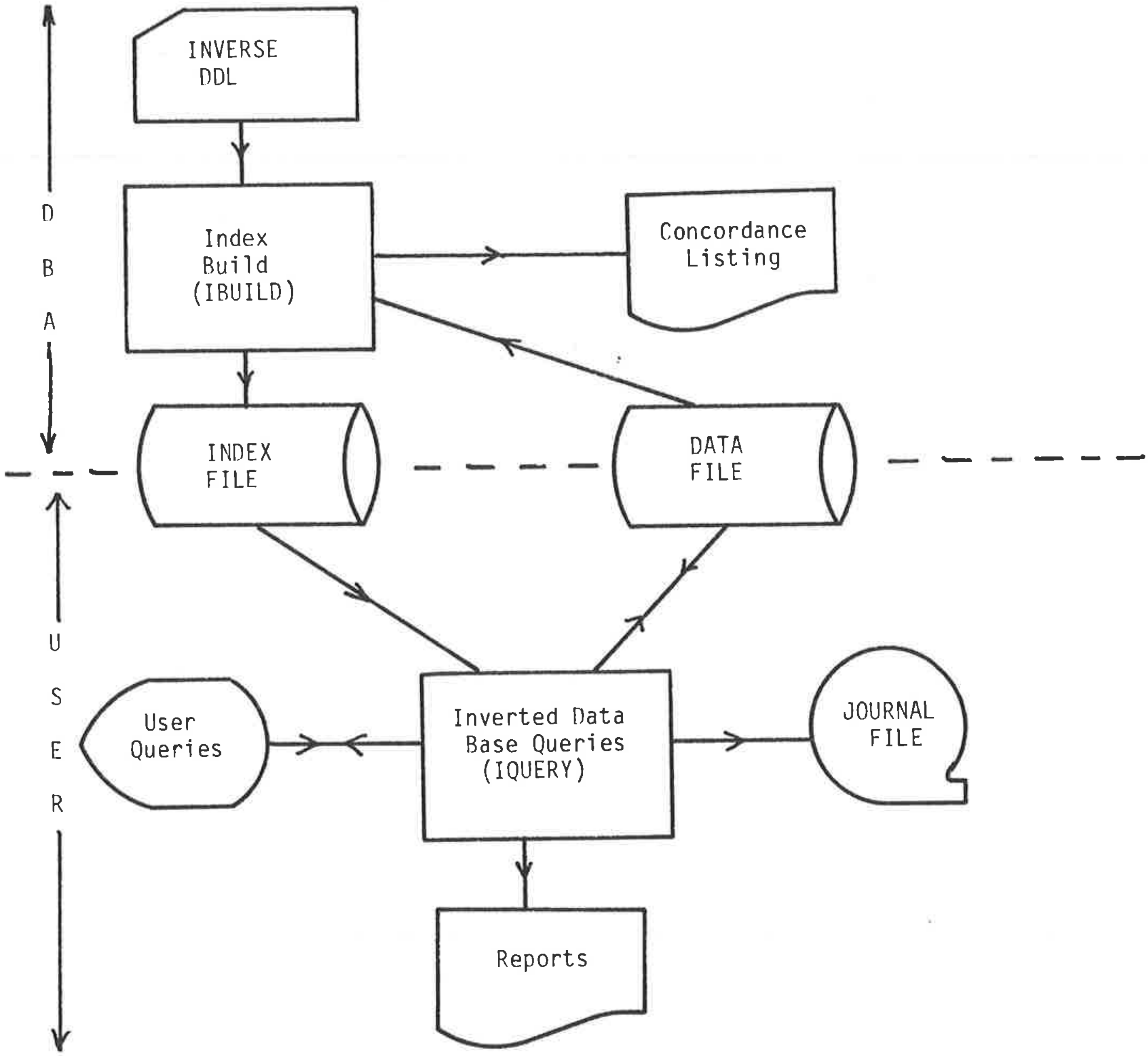


Figure 6.1: INVERSE System Structure

6.2 INVERSE Data Description Language

Because users of the INVERSE system are likely to be more sophisticated than some of the users of the SEQUENT system described in Chapter 5, the setting up of the data dictionary parts of the index file is accomplished using a Data Description Language (INVERSE DDL) rather than using a conversation style dialogue. This is necessary because the recording of data description and index building is integrated in a single process, and this process needs to be redone whenever the indexed fields are changed. For example, in the example used in Appendix 4 this update and re-indexing is carried out monthly.

The CYBER INVERSE CCL procedure call

INVERSE, BUILD, I = data

invokes the Index Build program IBUILD which reads the DDL and from it constructs the index.

A part of the DDL given in full in Appendix 4 is shown below.

```
INVERT ALL RECORDS.
PRINT SUMMARY.
INDEX FIELD NAME IS ZONING-CODE
      POSITION IS 205 TYPE IS ALPHA LENGTH IS 3.
FIELDNAME IS FRONTAGE POSITION IS 50
      TYPE IS NUMERIC LENGTH IS 5.
```

The formal syntax of the language is given in Appendix 1.

The INVERT statement controls the selection of records for inversion. As shown in the example above all records in the file can be accessed through the index but by using the form INVERT FROM m TO n then only the records with ordinal numbers m through n are indexed. The records of the data file are held in the ANSI-COBOL Relative file organisation where each record is identified in serial order by an ordinal number starting from 1. If the data file is loaded sorted by some prime search key then by a judicious use of the values of m and n a view can be built in which preliminary selection by the prime key can be done while building the index. The user of the view need not then select on this prime key using QUILL but need only concern themselves with other subordinate search keys. For example, the town planning database of Appendix 4 is sorted by LGA (Local Government Area number) because it is known that each group of users of the database (in say one subject or course) will restrict their searches to a few adjacent LGA's as part of some assignment or project activity. Thus while the database consists of some 400000 records for 100 or so LGA's, each query can be constrained to a few hundred (for small geographic areas) up to perhaps a few thousand records. A second use of the INVERT FROM m TO n feature is to set up pilot indexes for testing and demonstration purposes. Thus the INVERT statement controls the "breadth" of the index (see Fig. 6.2).

The "depth" of the indexing is controlled by the FIELD statements. The Data Base Administrator (DBA) has the option of simply recording the nature and position of a field (so that it can be printed for selected records) or they can specify that an index is to be built for the field. The prefix INDEX on a FIELD statement identifies those fields for which indexes are to be built and on which record selection can be carried out.

The FIELD NAME clause identifies the field name that can be used by the query language user. The POSITION clause specified the character number (from 1) of the start of the field and the TYPE clause specifies whether the field is ALPHA or NUMERIC. The LENGTH clause specifies

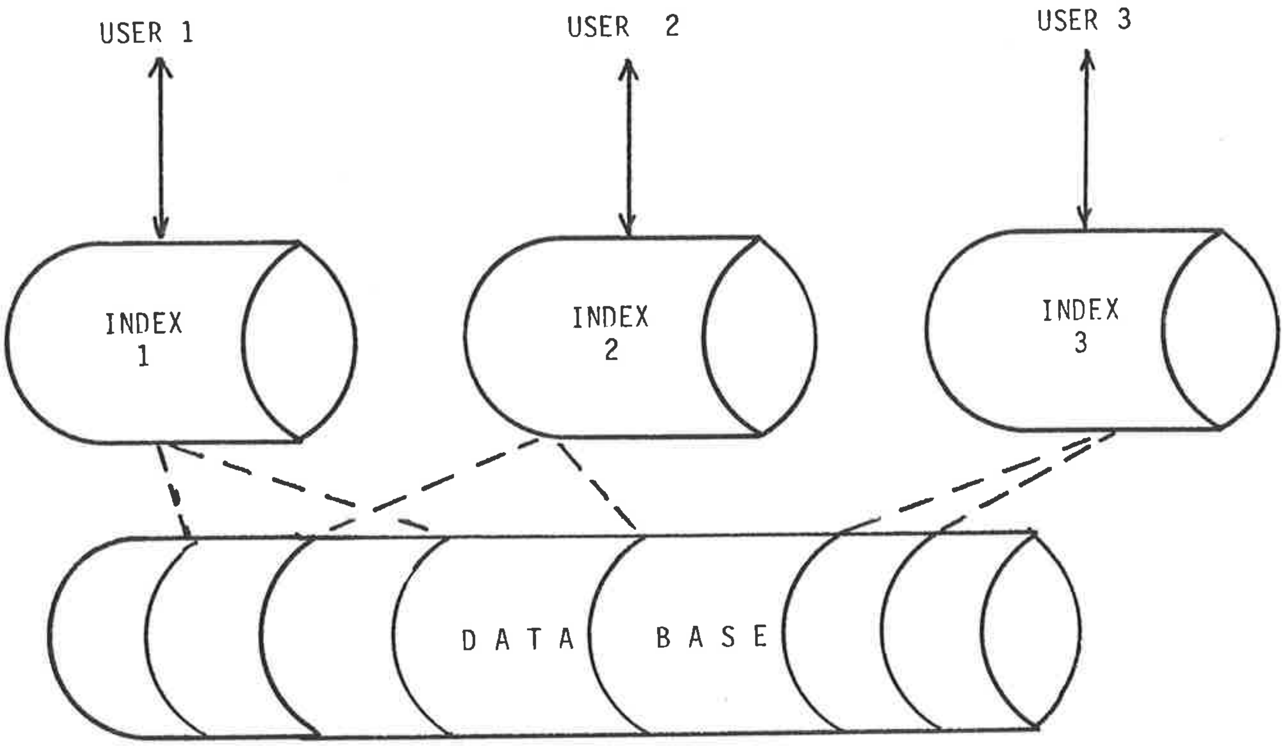


Figure 6.2: INVERSE User Views

the length of the field and for numeric fields this value may be followed by WITH 2 DECIMAL PLACES.

The language described above (and that given in Appendix 4) is somewhat verbose. This is satisfactory for use in examples but a shorthand form is available for experienced users which omits all optional and noise words and abbreviates certain key words.

```
INDEX  ZONING-CODE  205  A  3.
       FRONTAGE      50  N  5.
```

is all the DDL needed for the example given earlier in this section.

The PRINT SUMMARY statement, if specified, produces a concordance of values for each of the indexed fields. This concordance takes the form

FIELD NAME	FIELD VALUE	NUMBER OF OCCURRENCES
MARITAL-STATUS	D	27
MARITAL-STATUS	M	271
MARITAL-STATUS	S	83
MARITAL-STATUS	W	48
SEX	F	184
SEX	M	245

6.3 INVERSE Index Files

The index files have three levels of indexes leading to the data records (see Fig. 6.3). The top level is used to select a particular field (or attribute), the second level to select a particular attribute value, and finally the third level to select particular records.

Although there are three levels of index, there are only two different entry types in the index file (see Fig. 6.4).

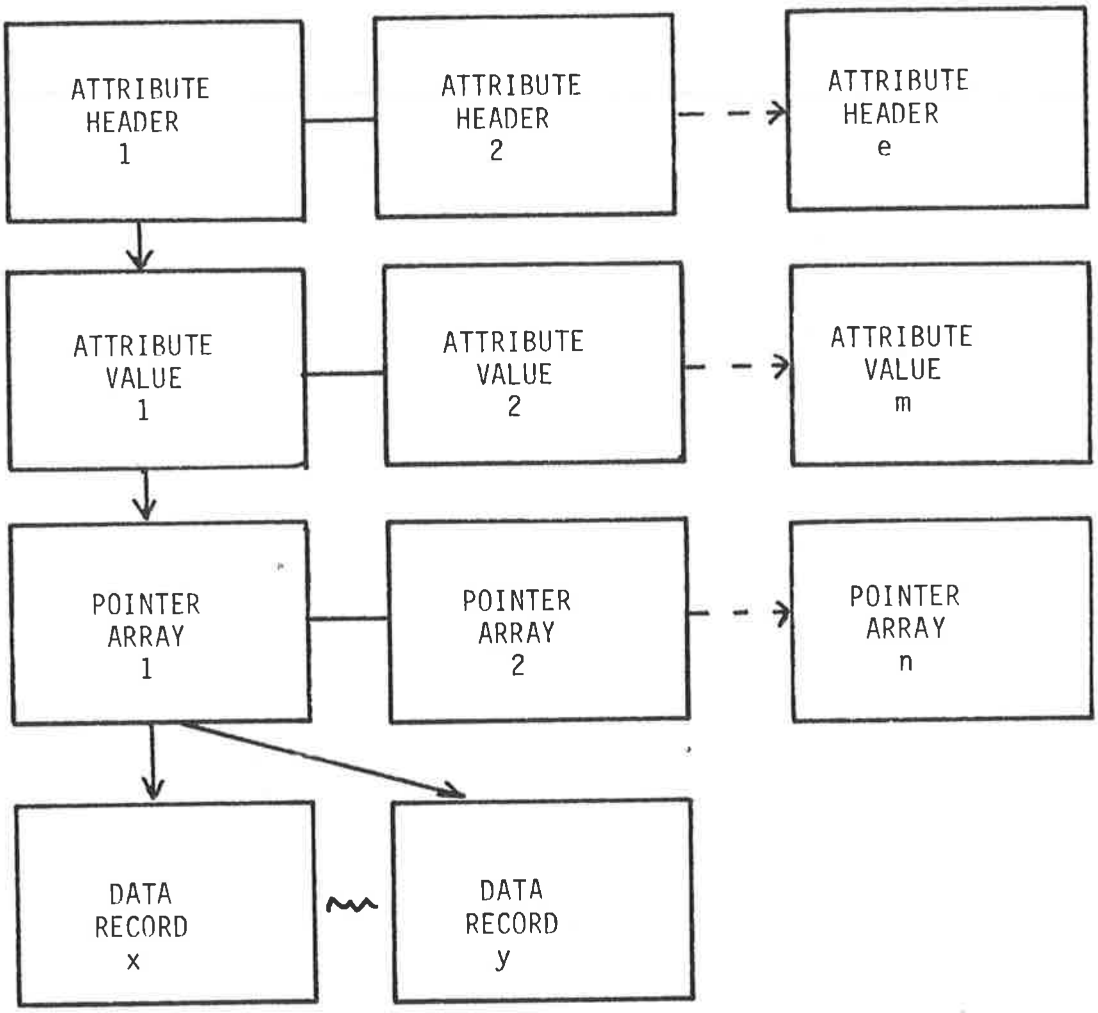


Figure 6.3: INVERSE Index Structure

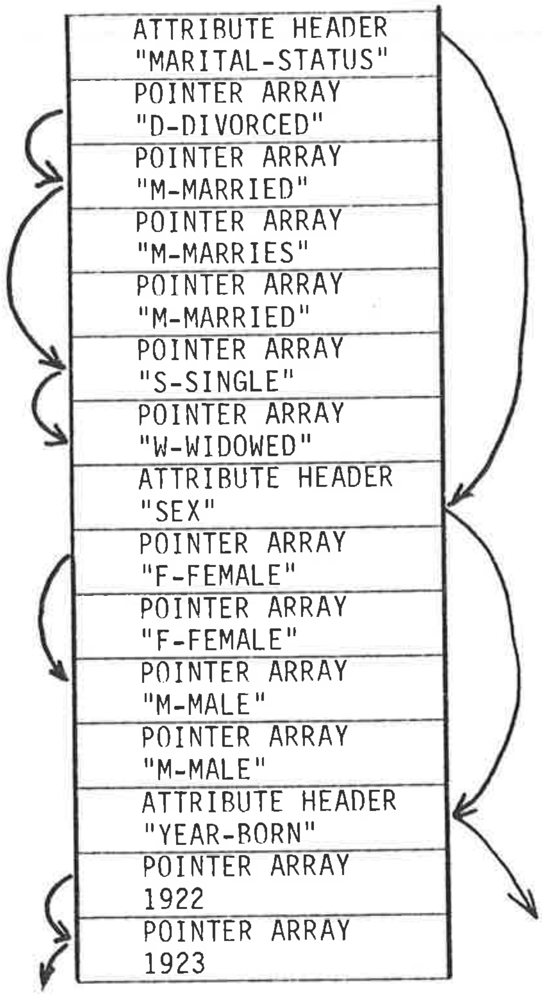


Figure 6.4: INVERSE Index File

Each pointer array consists of an attribute value and an array of pointers (record ordinals) to the data file. As implemented these arrays consist of 99 elements for a record size of 523 characters. By varying the record size the pointer arrays could be made shorter or longer and this can have a significant effect both on index file size and index search time. Consider the example in Fig. 6.4 where "marital-status" has four values of which one (married) requires three pointer array records, and the other three values require only one. There will thus be 4 "half-empty" index records. The attribute "sex" however will only have 2 incomplete records, while "year-born" could have 50-60. Thus if most of the attributes indexed have few values (like sex) a large index record size is desirable, whereas if most attributes indexed have many values (e.g. year-born) a small index record size is to be preferred. Thus the record size implemented is likely to be a compromise between these two extremes. The concordance listings can be used to monitor the index structure, and if desired the record size can be changed.

Within the index file the pointer arrays for the same value of the same attribute are grouped together in consecutive index records (see Fig. 6.4). For any attribute the groups of records for each value are also stored next to each other in the index file. Within the attribute the attribute values are stored in ascending sequence. The attributes themselves are also stored in ascending sequence. Within the index the attribute header records are stored immediately in front of the first pointer array record for the attribute. All the attribute headers are linked by a singly-linked list. Within each attribute the first pointer array record of each value is linked to the next highest value by other singly-linked lists.

Non-indexed attributes are written in the same format as indexed attributes. They are placed in front of the indexed attribute headers starting at record 1.

6.4 Building the Inverted Index

The inverted index is built by program IBUILD (see Figure 6.1). The program reads in the DDL describing the fields to be indexed, and secondly builds the index.

The index is built in three stages. In the first stage the data file is read from start to finish (or between the limits set by the INVERT...FROM...TO... statement). As each record is read all the fields to be indexed are extracted. Each of the extracted fields are written to a work file with the following information in each work record:

Data Record Ordinal
Attribute Name
Attribute Value.

When all the work records have been written, stage two sorts the work file on attribute value within attribute name. The work file can now be read sequentially by the third stage which loads the index attribute by attribute, value by value.

6.5 Inverted Database Query/Update

The Inverted Data base query program IQUERY is invoked by the CYBER CCL procedure call

```
      INVERSE,QUERY.
or    INVERSE, QUERY, I=TEXT (batch from file TEXT)
```

The module structure of IQUERY is shown in Fig. 6.5. Modules QLSCE and LEXAN are standard to all the query programs.

Module SCANIV is a special purpose module for evaluating query boolean expressions against an inverted database. SCANIV is called by QLSCE with the parameters described in Chapter 4, Section 3. Briefly recapping, these parameters include

- SEARCH-FUNCTION
- CONDITIONS
- TEST-VALUES
- RETRIEVE-FIELDS
- BUFFER
- SEARCH-STATUS

SEARCH-FUNCTION can take one of the values "OPEN", "CLOSE", "FIND", "GET" and "PUT".

The OPEN function opens the index, data and journal files and then locates all the attribute headers in the index file. This ensures that any reference to a particular attribute can go directly to the first value record for that attribute (see Fig. 6.4).

The CLOSE function closes the index data and journal files.

The bulk of the SCANIV program is concerned with the FIND function. The FIND function takes the CONDITIONS and TEST-VALUES and evaluates each condition by locating the pointer arrays associated with the appropriate values of the attribute named in the condition.

Thus the QUILL query

WHERE SEX = F DISPLAY NAME.

will retrieve the pointer array elements for the attribute "SEX" and the attribute value "F". This pointer array of data record ordinal

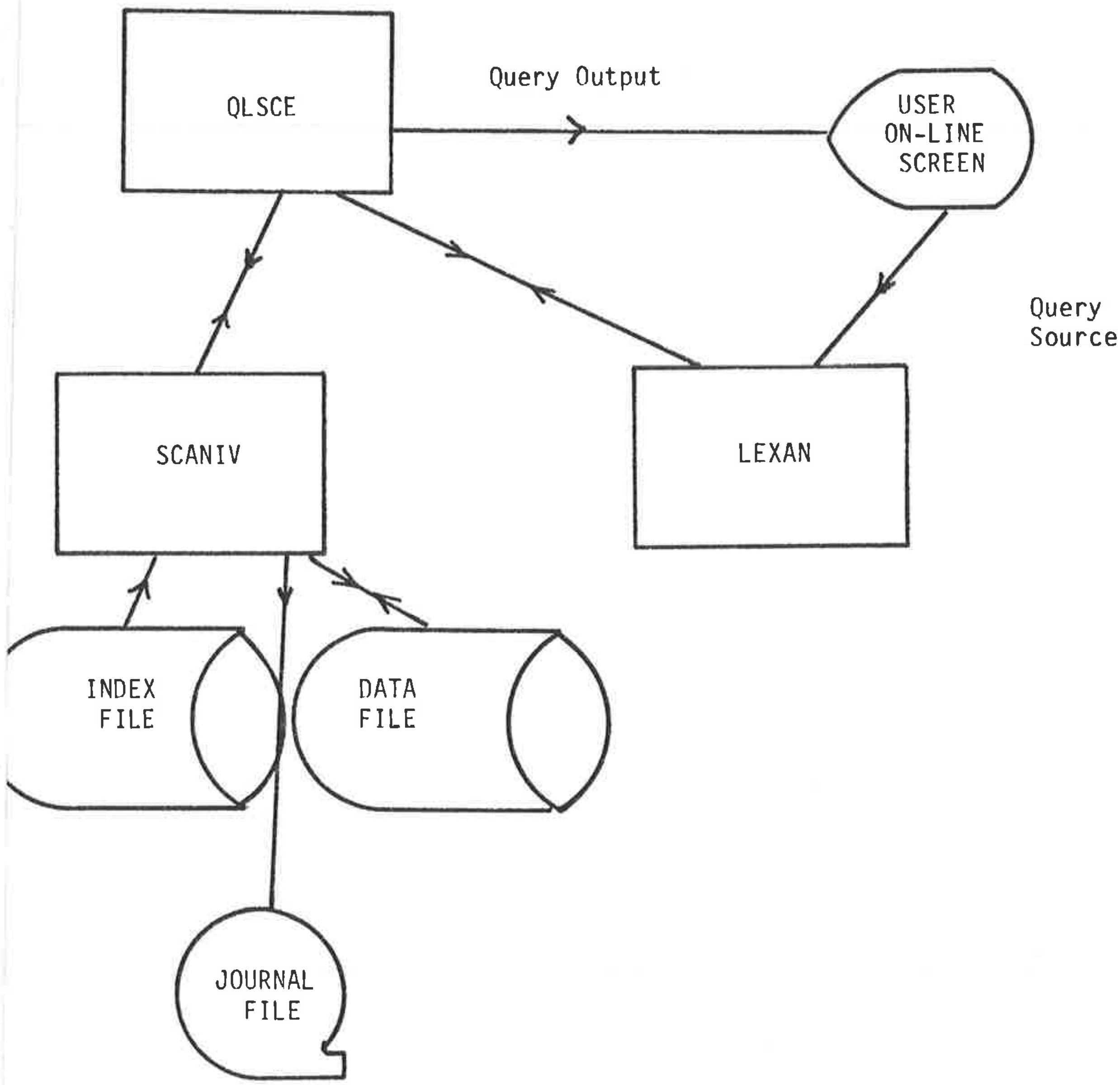


Figure 6.5: INVERSE Query/Update Program IQUERY Module Structure

numbers is then made available to the GET function described later in this chapter).

With the query

WHERE SEX = F AND MARITAL-STATUS = "S" DISPLAY NAME.

first the list of females is built and this is added to the top of a stack of such lists. Next the list of single people is built and this is added to the top of the stack. Finally the two lists are combined into a single list. In the above query the combination results in a new list containing only record ordinals common to both lists (see Fig. 6.6). However with the query

WHERE SEX = F OR MARITAL-STATUS = S display name.

the combination results in a list containing the records numbers found in either (or both) original lists as in Fig. 6.7.

The combination can only proceed if both lists relate to conditions at the same level (depth of parenthesis). If the second condition refers to a higher level (deeper parenthesis) then both lists are left on the stack (see Fig. 6.8) and are not "reduced" until either a lower level condition is encountered (equivalent to passing through a right parenthesis) or else the end of the boolean expression is reached. The reduction process continually reduces the level of the list at the top of the stack and combines it with the list immediately underneath it (if both lists are now at the same level) until the level of the topmost list is equal to the level of the condition about to be evaluated.

Thus consider the QUILL statement

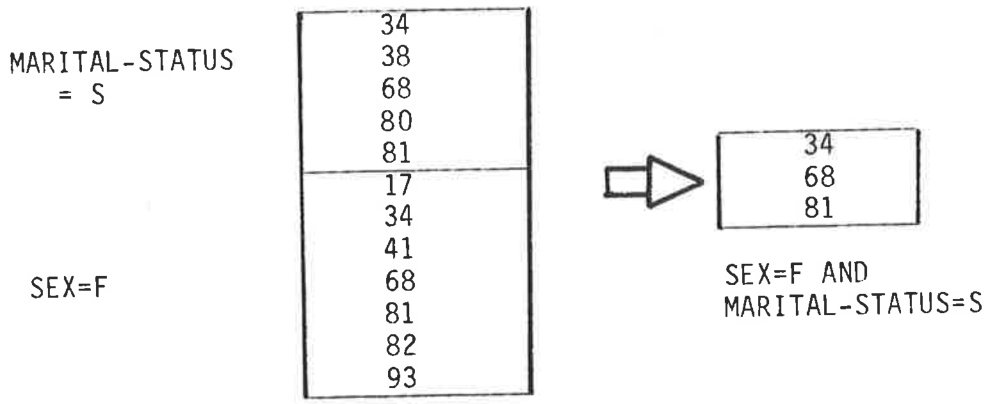


Figure 6.6: AND stack lists

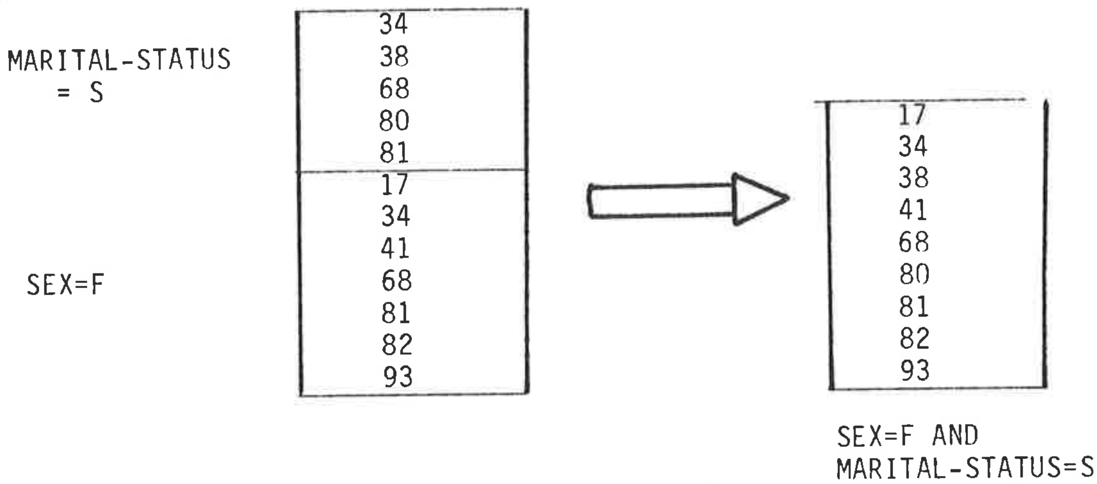


Figure 6.7: OR stack lists

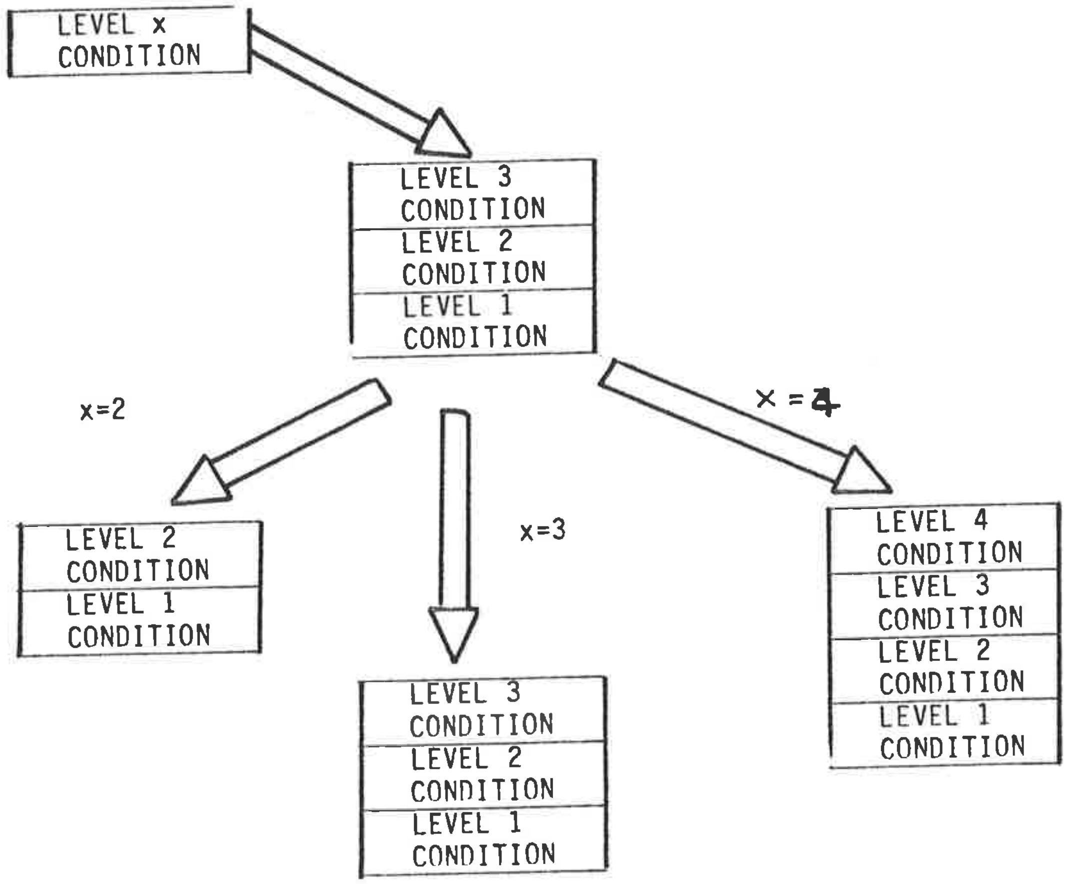


Figure 6.8: SCANIV Condition Evaluation Stack

```
WHERE SEX = F AND (YEAR-BORN = 1942 OR MARITAL-STATUS = S)
OR MAIDEN-NAME = JONES DISPLAY NAME.
```

The nested conditions are now reduced in a multi-stage combination process. First the SEX=F list is added to the stack and then the YEAR-BORN = 1942 list is placed on top. Next the MARITAL-STATUS = S list is put on the stack and then the top two lists are combined (see Fig. 6.9).

When MAIDEN-NAME = JONES is encountered, SCANIV recognises that this condition is at level 1 whereas the top of the stack has a level 2 condition (YEAR-BORN = 1942 OR MARITAL-STATUS = S). This level 2 condition is reduced by 1 level and combined with the level/condition underneath it (SEX = F). Only after this has been done is the new level 1 condition (MAIDEN-NAME = JONES) added to the stack (see Fig. 6.10). This reduction process ensures that where levels of parenthesis are equal then a left-to-right evaluation is performed.

After the FIND function has built its single list of record ordinals the GET function of SCANIV reads the list of data record pointers resulting from the invocation of the FIND statement. Each use of GET returns a single record to the calling routine (the QLSCE module, see Chapter 4). If a record is available the STATUS-FLAG of the calling parameters is set to spaces and the fields specified in the OUTPUT-FIELDS list are extracted from the data record and loaded into BUFFER. If GET is used and all records found by FIND have been returned then the STATUS-FLAG is set to "NO MORE".

The PUT function is used by the calling routine to indicate that some (or all) of the fields in the BUFFER have been changed. A before and after image is logged on the journal file and the data record is updated

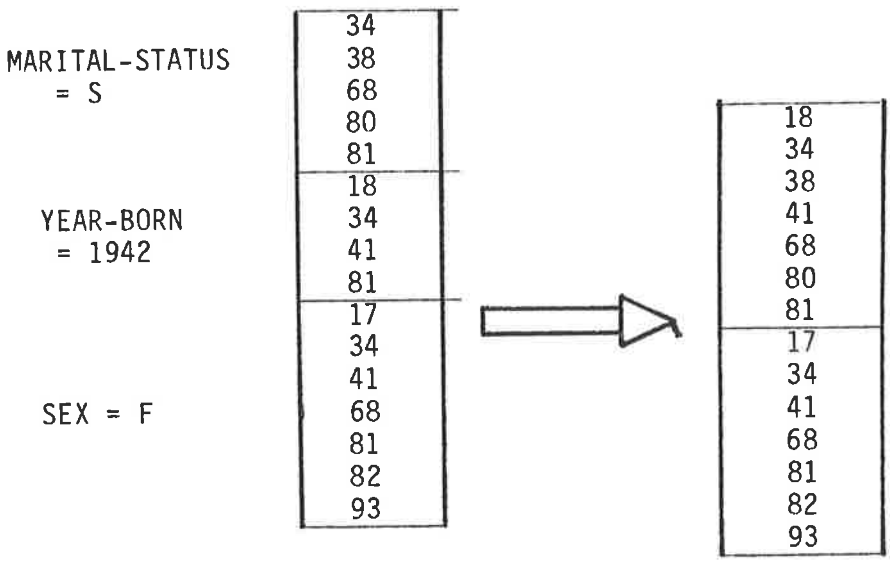


Figure 6.9: Evaluation of nested conditions (part 1)

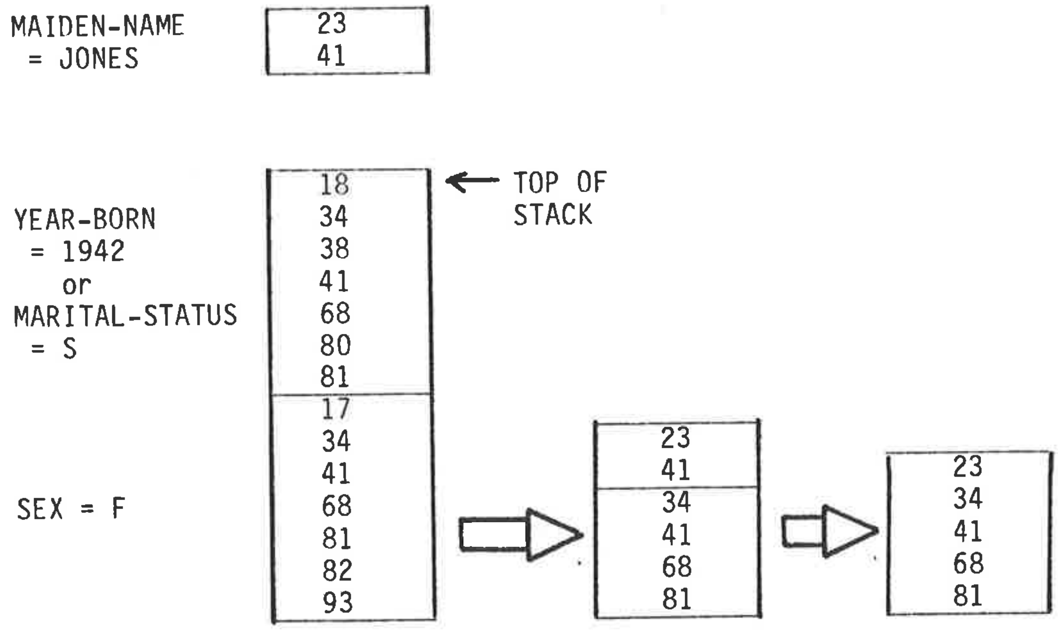


Figure 6.10: Evaluation of nested conditions (part 2)

using the fields in the buffer. The contents of the journal records are shown below

Query number (1 → n)

Data Record Ordinal number

Image Flag (A = After, B = Before)

Copy of data record

The structure diagrams for SCANIV are included in Appendix 2.

CHAPTER 7

HIERARCHIC DATABASE SYSTEM (PYRAMID)

7.1 Introduction

A PYRAMID database consists of a collection of entity types contained within a single indexed sequential file.

The entity types are organised in a hierarchy where, with the exception of the root type, each entity type is "owned" by another type of entity. Consider Figure 7.1 where a COMPANY database consists of zero or more DEPARTMENT's. Each department (the root entity type) owns zero or more instances of both EMPLOYEE and PROJECT entities. In turn the employees own zero or more ALLOWANCE's and the projects zero or more PURCHASES's. Each entity type (except the root) can only be identified with respect to its owning entity. This in Fig. 7.1 there may be several project entities with the same key (of project number) but there will not be duplication of project numbers within any department.

All five entity types described above are stored together in a single physical file. One or more physical files are described in the "Internal Schema" using an Internal Schema Data Description Language. For any given internal schema there may be several user views or "External Schemas". These are described in External Schema Data Description Language. Each external view is a subset of an internal schema in which certain attributes from certain entities are defined. Thus one user view of the internal schema of Fig. 7.1 is shown in Fig. 7.2.

The PYRAMID system has two user interfaces

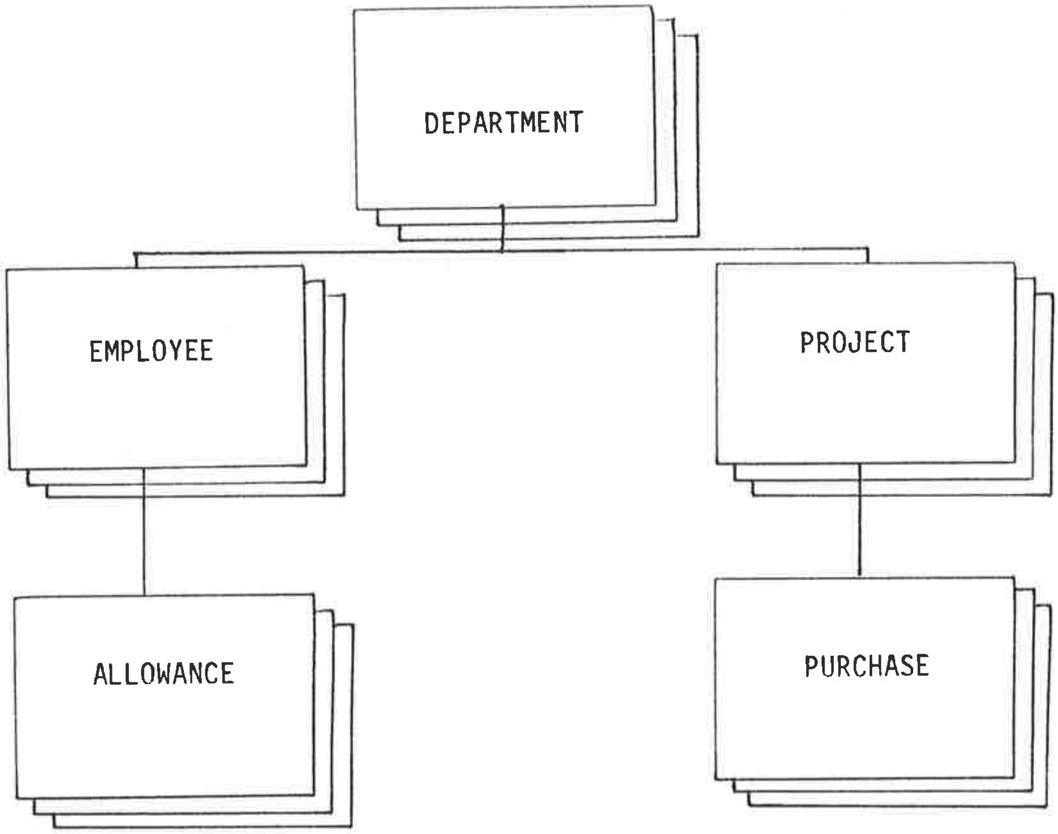


Figure 7.1: COMPANY internal schema

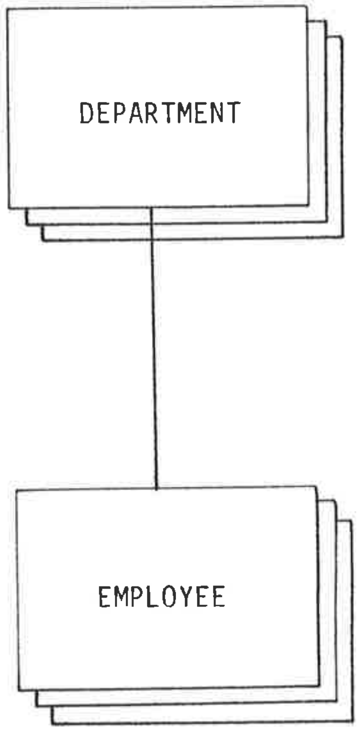


Figure 7.2: PAYROLL external schema

- host language interface
- query language interface.

The overall structure of the PYRAMID system is shown in Fig. 7.3. Programs INTDDL and EXTDDL handle the Internal and External Schema Maintenance activities. Program PBUILD generates a COBOL sub-program that maps user calls in terms of the external schema into file and record processes on the physical files of the internal schema. When compiled to form the "Mapping Object Code" this mapping can be combined either with a user program or with the QUILL query language module QLSCE to form a complete program. On the CYBER all except the last of these activities are controlled by the PYRAMID CCL procedure.

7.2 PYRAMID Databases

The entities of a PYRAMID database may be accessed randomly or sequentially. In both cases access to lower level entities is through the owning entity (and so on up through the tree to the root entity).

An efficient implementation of the above requirements demands that groups of owned entities can be accessed easily once the owning entity is located, and that any entity can be located directly using a key.

Figure 7.4 shows a typical implementation of the hierarchy

CUSTOMER
↓
INVOICE
↓
ITEM

where customers order a number of items to be billed on an invoice. At any one time several such invoices may be on order. In Fig. 7.4 the CUSTOMER entities might be accessed directly via an index or hashing algorithm (or more rarely chained together). The INVOICE entities owned

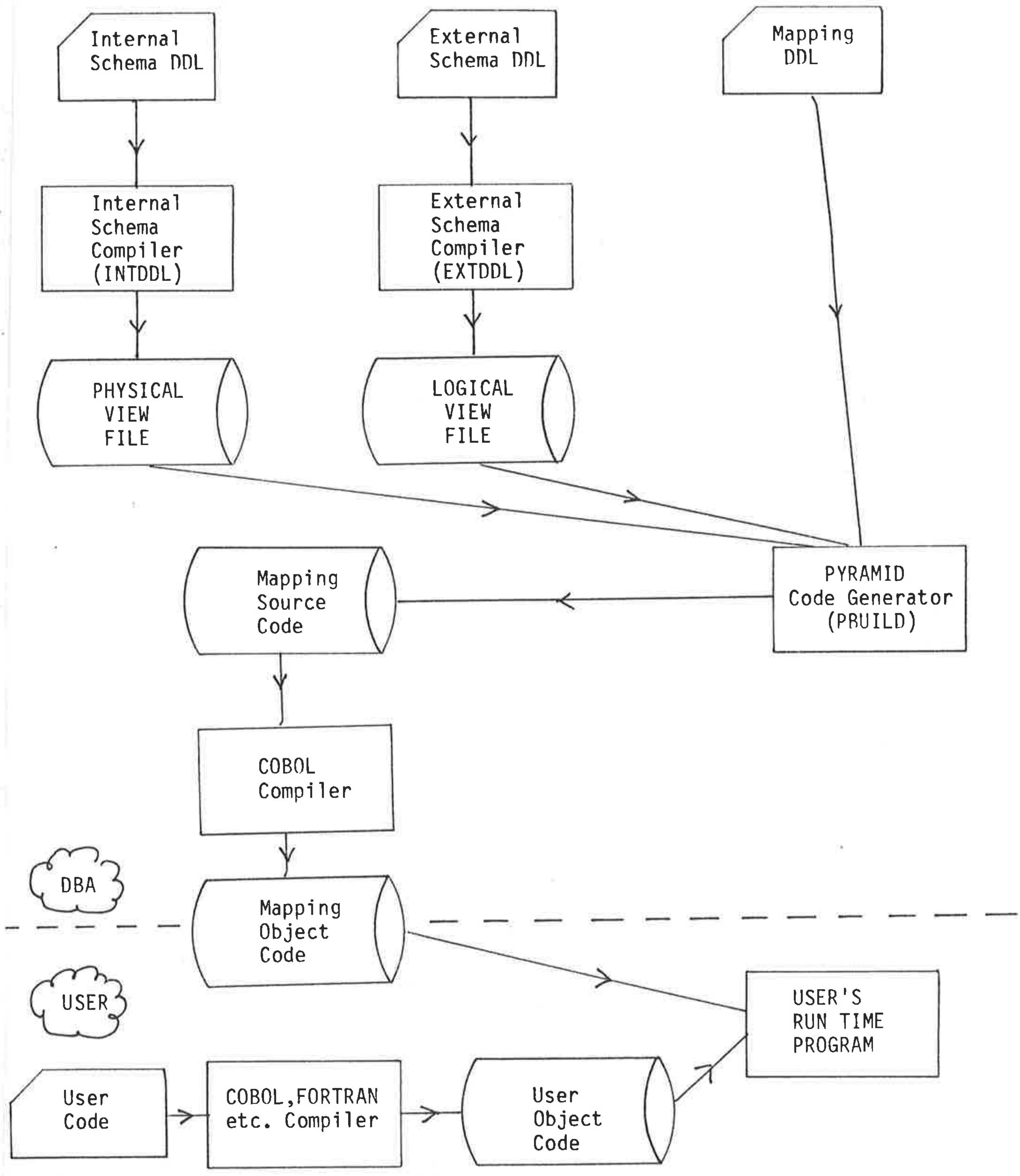


Figure 7.3: PYRAMID System Structure

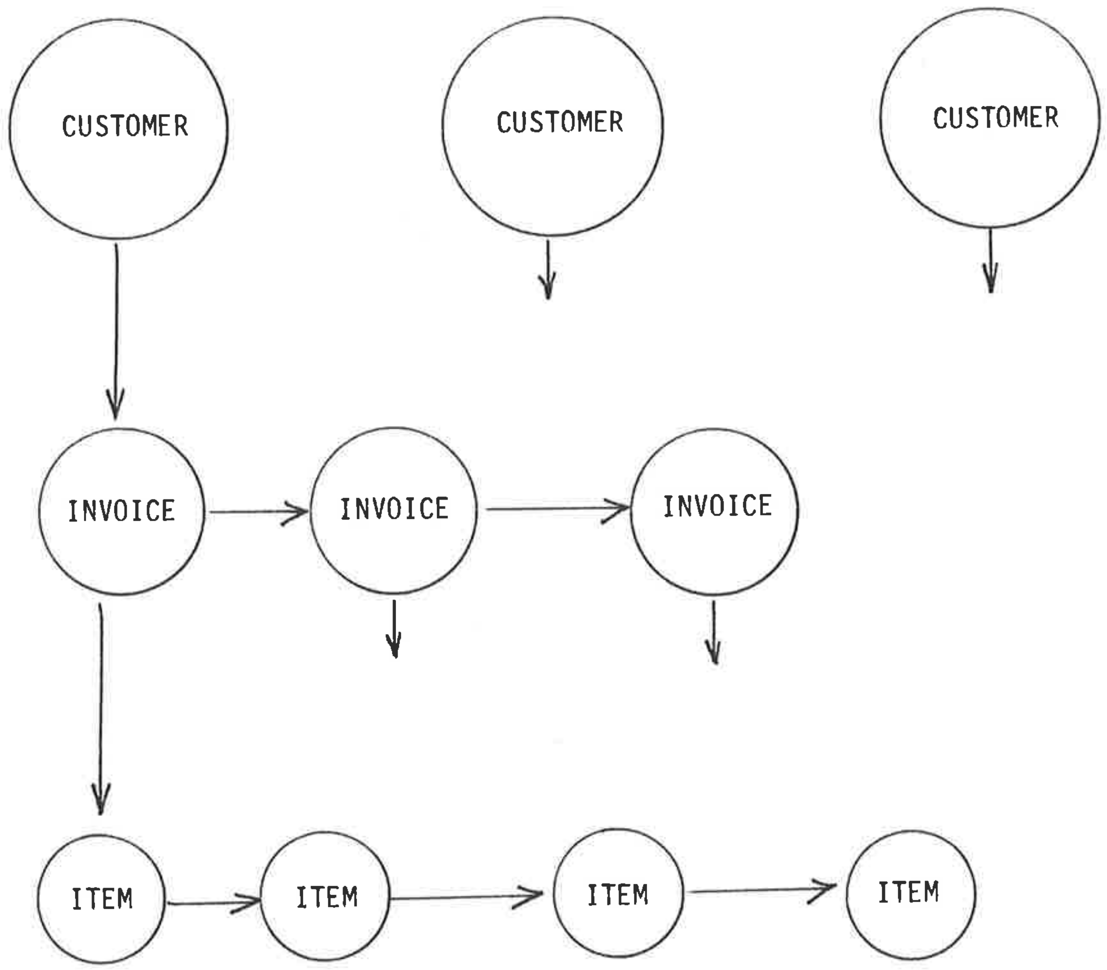


Figure 7.4: Chained Implementation of a Hierarchy

by any given CUSTOMER could be linked to each other to form a chain with the list head printer in the owner entity. In like manner the ITEM entities can be linked to an INVOICE entity. The major advantage of this approach is that by using (say) record ordinals to identify records little disc space overhead is taken up by the pointers.

A major disadvantage however is that access to specific owned entities requires the chain of owned entities to be traversed. This search can be speeded up by maintaining the owned records in some key order within the chain but this improvement in retrieval time is achieved at the expense of complicating the process of inserting new owned entities.

An alternative arrangement is to dispense with the owned entity chain and hold pointers to all owned entities in the owned record (a "pointer array"). This arrangement works quite well when each entity owns only a small number of owned entities (e.g. PERSONS owning CARS), but causes problems when in the 1:n relationship n is large (e.g. ELECTORAL-AREA owning VOTER).

Another approach entirely to the representation of hierarchies is suggested by a traditional magnetic tape method using header and detail records. Thus given the need to represent the hierarchy

DEPARTMENT
↓
EMPLOYEE

a magnetic tape could contain the following sequence of records

DEPT	EMP	EMP	EMP	DEPT	EMP	EMP	DEPT
A	3	5	6	8	4	7	C

with employees 3, 5 and 6 being in department A, employees 4 and 7 in department B, etc. Each of the department records would typically contain information common to all employees in the department (e.g. department name, location, pay rates etc.). In some implementations the different record types are identified by a type field similar to Djikstra's discriminated union (Dahl 1972). For example a "record type" field might have the value D or E for department and employee records respectively. This technique is satisfactory where records can be maintained in order, but another technique of even greater vintage (dating back to the punched card era) not only idetifies each record type but also allows the sequence of owning and owned records to be maintained. This is achieved by having a multi-level sequence key (in the example above DEPT-NO and EMP-NO). By assigning a low value (e.g. zero) to the EMP-NO field of a department record, and by ensuring that all employee records have an EMP-NO greater than this low value and also have the same DEPT-NO value as their owning department record, then by sorting the records on EMP-NO within DEPT-NO the records on the file fall naturally into their correct hierarchic relationship. Department and employee records can be distinguished by whether or not the EMP-NO field is zero.

This method can be extended to more levels. Thus in the hierarchy

DIVISION
↓
DEPARTMENT
↓
EMPLOYEE

a department record would have EMP-NO zero but DEPT-NO and DIV-NO non-zero.

The implementation of the hierarchy used for PYRAMID combines the "multi-level key" and the "record type" techniques described above. The entities are not stored on a sequential file however but in an indexed sequential file and by this means it is possible to read entities directly.

For example, returning to the

```
CUSTOMER
   ↓
INVOICE
   ↓
  ITEM
```

hierachy, the PYRAMID entity layouts are shown in Figure 7.5.

The 3 key fields and the entity code field appear in the same place in each of the three entities (usually but not necessarily at the front). The data content of the three different entities vary both in use and total size.

A customer entity has a non-blank CUSTOMER-NO field, with the other two key fields being spaces. The COBOL literal SPACES is used instead of the literal LOW-VALUES so that not only the software can be transported to other machines but possibly also some example databases.

The invoice entity has a non-blank INVOICE-NO as well as CUSTOMER-NO. Only the order line entity has the ORDER-ITEM field present.

The traditional method has to be varied when the hierarchy has multiple-legs as well as multiple-levels. Considering the hierarchy shown in Fig. 7.6 where PAYMENT entities have been added to the database to record the receipt of money from the customer to pay for the products ordered on the invoices. Following the style of Djikstra the key structure of Fig. 7.7 could be used with the field LEG-NO having the value "1" for invoices and order-lines and the value "2" for payments. This technique keeps key length to a minimum and also keeps the invoices separate from

Customers file general record layout

KEY			ENTITY CODE	ENTITY DATA
CUSTOMER -NO	INVOICE -NO	ORDER -ITEM		

Customer entity

KEY	ENTITY CODE	CUSTOMER -NAME	CREDIT -LIMIT	BALANCE	TOTAL -VALUE -ON-ORDER

Invoice entity

KEY	ENTITY CODE	INVOICE -DATE

Order-line entity

KEY	ENTITY CODE	ORDER -QTY	ORDER -PRICE

Figure 7.5: CUSTOMERS File Entity Layouts

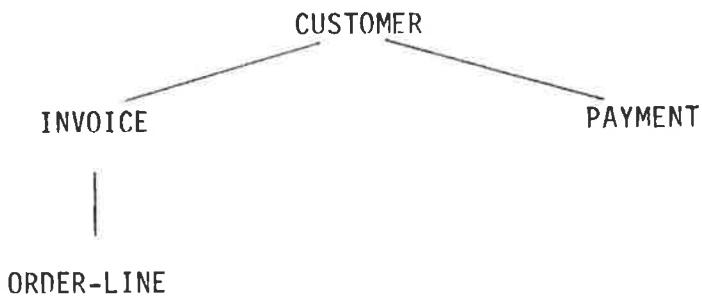


Figure 7.6: Multi-leg hierarchy

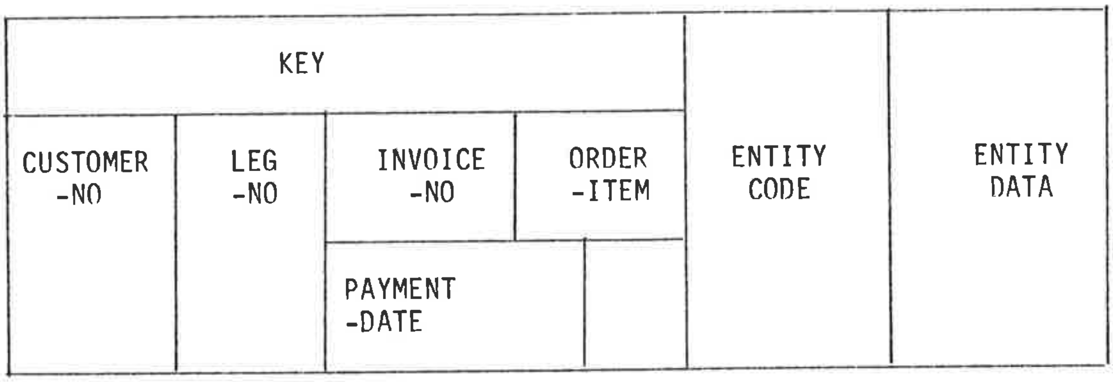


Figure 7.7: Possible key structure for a multi-leg hierarchy.

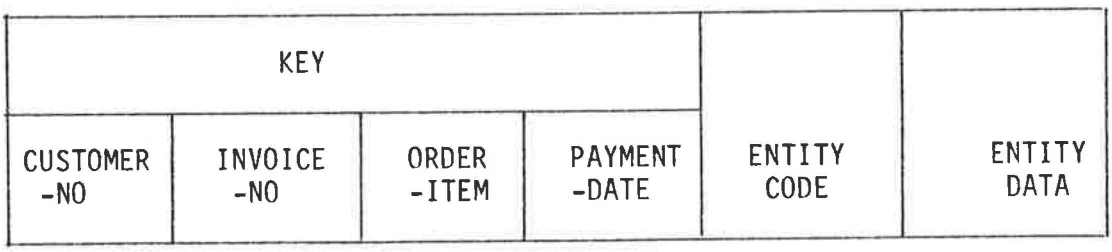


Figure 7.8: PYRAMID multi-leg hierarchy key structure.

the payments for any given customer. If the hierarchy branches into different legs at several points in the structure then LEG-A-NO, LEG-B-NO etc. can be used to control the structure.

The technique described above is fairly complicated for complex hierarchies, and so the PYRAMID databases are implemented using a conceptually simpler technique that does however make the key longer.

In the PYRAMID technique the key field for each type of entity has a unique place in the composite key area. In Fig. 7.8 the payment-date field is set to spaces for invoice and order-line entities. A payment entity has the payment-date field non-blank but has spaces in both the invoice-no and order-item fields.

In essence the key structure of PYRAMID linearizes the two-dimensional entity structure so that top-down in the hierarchy becomes left-right in the key order of the entities in the database. Provided that the owned entities of any given entity are located to the right of the owning entity, the placement of owned entities from different legs is immaterial. Thus Fig. 7.9, 7.10 and 7.11 are all permissible implementations of Fig. 7.6.

The three different database entity orders are achieved by specifying the entity descriptions in different orders.

File		Specification order		
Sequence	1st entity	2nd entity	3rd entity	4th entity
A	Customer	Invoice	Order-item	Payment
B	Customer	Invoice	Payment	Order-line
C	Customer	Payment	Invoice	Order-line

CUSTOMER -NO	INVOICE -NO	ORDER -ITEM	PAYMENT -DATE	ENTITY -CODE	DATA
1				1	CUST
1			1	4	PAY
1	1			2	I/V
1	1	1		3	O-I
1	1	2		3	O-I
1	2			2	I/V
1	2	1		3	O-I

Figure 7.9: CUSTOMER File Sequence A

CUSTOMER -NO	INVOICE -NO	PAYMENT -DATE	ORDER -ITEM	ENTITY -CODE	DATA
1				1	CUST
1		1		3	PAY
1	1			2	I/V
1	1		1	4	O-I
1	1		2	4	O-I
1	2			2	I/V
1	2		1	4	O-I

Figure 7.10: CUSTOMER File Sequence B

CUSTOMER -NO	PAYMENT -DATE	INVOICE -NO	ORDER -ITEM	ENTITY -CODE	DATA
1				1	CUST
1		1		3	I/V
1		1	1	4	O-I
1		1	2	4	O-I
1		2		3	I/V
1		2	1	4	O-I
1	1			2	PAY

Figure 7.11: CUSTOMER File Sequence C

The entity codes are assigned in specification order. It can be seen that the order in the database is the mirror-image of the standard post-order tree traversal algorithm (Knuth, 1973).

Thus the Database Administrator (DBA) can mould the file structure by writing the DDL in particular ways and hence the DBA can optimise particular sequential operations on the database. However, while sequences A, B and C may be more efficient for certain operations, it is clearly necessary that all 3 operate identically as far as the user is concerned and that the codes generated for the mapping should maintain the integrity of the external views.

7.3 PYRAMID Internal Scema DDL

Like INVERSE (see 6.2), the PYRAMID system uses a data description language (DDL) to describe both in Internal and External Schemas.

The "internal schema" is a description of the physical files on which the data is held.

Unlike SEQUENT and INVERSE, attributes within PYRAMID entities may not overlap but they may be sub-divided into further attributes. Thus the PYRAMID attributes have a hierarchic structure similar to the systems entity structure.

Because of the nested nature of the PYRAMID attributes, the COBOL-like DDL used for INVERSE is considered inappropriate for PYRAMID. Instead a more concise (but less easy to read) form of language is used.

The CYBER CCL procedure call

```
PYRAMID, INTDDL, I = data
```

invokes the Internal Schema Compiler to read the DDL and sets up the "Physical view dictionary".

A part of the DDL given in full in Appendix 5 is shown below.

```
NEW DICTIONARY.
INTERNAL SCHEMA NAME IS MANUFACTURING.
FILE NAME IS CUSTOMERS; ORGANISATION IS INDEXED;
ASSIGN TO ORDERS.
ENTITY NAME IS CUSTOMER; KEY IS CUSTOMER-NO
(CUSTOMER-NO/C 6, CUSTOMER-NAME/C 30, CREDIT-LIMIT/N 8.2,
BALANCE/N 10.2, TOTAL-VALUE-ON-ORDER/N 8.2).
ENTITY NAME IS INVOICE; OWNER IS CUSTOMER; KEY IS INVOICE-NO
(INVOICE-NO/C 6, INVOICE-DATE/C 6).
ENTITY NAME IS ORDER-LINE;  KEY IS ORDER-ITEM; OWNER IS INVOICE
(ORDER-ITEM/C 4, ORDER-QTY/N 6, ORDER-PRICE/N 5.2).
```

The formal syntax of the language is given in Appendix 1.

The NEW DICTIONARY statement appears if (and only if) a new dictionary file is to be created. (Several internal schemas may be held on the same dictionary file.) The statement INTERNAL SCHEMA NAME IS MANUFACTURING identifies the particular schema.

An internal schema can consist of one or more database files. In the example above there is only one file which has (but need not have) the same name and the schema. The ORGANIZATION clause is not used at present but allows for other implementations of PYRAMID data bases (e.g. DIRECT for a hashed file, SEQUENTIAL for a positional file, etc.). The ASSIGN clause identifies the physical file in the host operating systems filestore.

Each entity in the file is described in a single ENTITY statement.

The entity is named in the NAME clause. Except for the root-entity, the OWNER clause specifies the owning entity name. Thus in the example given CUSTOMER has no owner, whereas the INVOICE entity specifies the CUSTOMER entity as its owner.

The KEY clause names the attribute to be used to identify instances of the entity <u>within</u> a specific instance of the owning entity. The key may be an elementary attribute, or a composite attribute. For example the INVOICE entity is specified with key INVOICE-NO (an elementary attribute).

An example of a composite attribute being used as a key is the field NAME (consisting of the elementary attributes SURNAME and INITIALS from the following DDL.

```
ENTITY NAME IS EMPLOYEE;  KEY IS NAME (EMP-NO/C4,
NAME (SURNAME/C20, INITIALS/C4), SEX/C1,
SALARY/N5).
```

The attributes of the entity are described in sequence enclosed in parentheses. Each elementary attribute is followed by its format as in the examples below

CUSTOMER-NAME/C30	30 characters
ORDER-QTY/N6	6 digit integer
ORDER-PRICE/N5.2	5 digit number with 2 decimal places

Composite attributes are followed by their constituent elementary attributes enclosed in parentheses. For example

NAME(SURNAME/C20, INITIALS/C4)

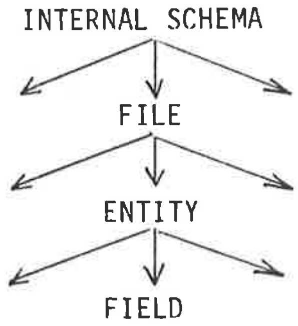
This nesting of attributes may be continued indefinitely. Thus the user can define

NAME(SURNAME/C20, FORENAMES(FIRST-NAME/C15, OTHER-INITIALS/C3))

The external (user) interface can refer to any of the names defined. Thus in the last example NAME is 30 characters long, FORENAMES is 18 characters, and FIRST-NAME is 15 characters.

The internal schema dictionary has a hierarchic structure with a key structure similar to a PYRAMID database. (Theoretically it is possible for the dictionary to be a PYRAMID database though this has not been implemented.)

The hierarchy is shown below



Each entry in the dictionary has a four-part key consisting of

INTERNAL-SCHEMA-NAME

FILE-NAME

ENTITY-NAME

FIELD-NAME

No entry exists at the internal-schema level as no information is required to be held for the schema as a whole. (Potential exists however for say privacy locks to be placed here if this is ever felt necessary.)

The file level entry contains the following information

File-organization

Access-Mode

Assign-name

From this entry the mapping generator (see 7.6) can generate the COBOL statement

SELECT filename ASSIGN TO assign-name

ORGANIZATION IS file-organization

ACCESS MODE IS access-mode.

All other clauses of the SELECT...ASSIGN statement are left to be installation defaults. As has been stated earlier, the file organisation must be specified as INDEXED and the system itself specifies access mode as DYNAMIC. If however the installation COBOL compiler requires different values then it would be relatively easy to change the file level entry to accommodate these differences.

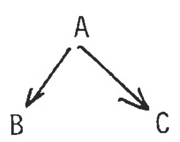
The entity level entry contains the following

Owner-name

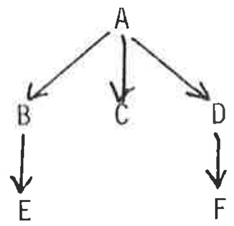
Entity-key

Entity-code

The owner-name identifies the opening entity (except for the root entity). The entity key identifies the field used to identify entity instances. The entity-code is a two-digit integer which identifies the entity type within the database. The code values are allocated in sequence from 1 as each entity is encountered in the DDL. It is thus possible to add new entities to a PYRAMID data base without changing the database other than extending the key field with spaces provided that the new entities can be and are added to the end of the DDL. This is always possible if (and this is usual) the new entity types are sub-ordinates to entities already in the data base. For example, the hierarchy of entities



defined in the order A B C can be extended to



without any problems by defining the entities in the order A B C E D F say. An example of this growth of a database definition is demonstrated in Appendix 5.

The field level entry contains the following information

Field-type

Field-length

Field-sequence

Field-level

Field-access

The first two contain the type and length of the field. Field-sequence is used to order the fields within an entity. The sequence numbers are allocated in the order the fields are defined in the DDL and they thus correspond to the order within the physical file record.

The field-level is 2 for an elementary field, with lower levels being used for parts of composite fields. The level numbers thus equate directly with COBOL level numbers and enable the mapping generator to generate the following COBOL Data Division code for the EMPLOYEE entity DDL given earlier.

```
01   EMPLOYEE
     02  FILLER PIC X(2).
     02  EMP-NO PIC X(4).
     02  FILLER PIC XX.
     02  NAME
         03  SURNAME  PIC X(20).
         03  INITIALS PIC X(4).
     02  SEX     PIC X.
     02  SALARY PIC 9(5).
```

The field-access is used to distinguish key fields from ordinary data fields.

7.4 PYRAMID External Schema DDL

The External Schema DDL of PYRAMID has a similar style to the internal schema DDL. The "external schema" is a description of the user view of one or more files and of the processing that the user is permitted to carry out through that view.

The external schema consists of one or more record descriptions with each record containing one or more items.

The record names must match some or all of the entities in the internal schema, and the item names for any record must match some or all of the field names of the matching entity.

The CYBER CCL procedure call

PYRAMID, EXTDDL, I = data

invokes the External Schema Compiler to read the DDL and sets up the "Logical view dictionary".

A part of the DDL given in full in Appendix 5 is shown below.

```
NEW DICTIONARY.
EXTERNAL SCHEMA NAME IS TROUBLE
PERMIT ACCESS FOR UPDATE, RETRIEVE, CREATE, FORMAT.
RECORD NAME IS CUSTOMER (CUSTOMER-NAME/C40, CUSTOMER-NO/C6,
CREDIT-LIMIT/N8.2, TOTAL-VALUE-ON-ORDER = TOTAL-VAL/N8.2).
RECORD-NAME IS INVOICE (INVOICE-NO/C6, INVOICE-DATE/N6).
RECORD ORDER-LINE = ORDER (ORDER-ITEM/C4, ORDER-PRICE/N5.2,
ORDER-QTY = QTY/N6).
RECORD NAME IS PART (DESCRIPTION/C40, PART-NO/C4,
UNIT-PRICE/N6.2, STOCK-IN-HAND/N6).
```

The NEW DICTIONARY statement is used to create a new dictionary and the EXTERNAL SCHEMA statement is used to identify the schema.

The PERMIT/DENY access statements control the range of options allowed to users of the view. If no such statement is present all facilities are available. If PERMIT is specified as in

PERMIT UPDATE, RETRIEVE.

then these two modes of access are permitted and all others (CREATE, FORMAT) are denied. The same effect can be obtained by writing

DENY CREATE, FORMAT.

RETRIEVE allows a user to retrieve records from the database, while UPDATE allows the contents of retrieved records to be changed and then replaced. CREATE allows the user to create new instances of records in the database. The FORMAT access allows the user to access a table of field formats for a given record. It is of use primarily to the query language QUILL through module SCANSQ.

The External Schema consists of a set of records and their constituent fields.

The records are defined as for example

RECORD NAME IS CUSTOMER

....

RECORD ORDER-LINE = ORDER

In the first extremal CUSTOMER records maps onto the internal CUSTOMER entity, whereas in the second the name ORDER is used externally to refer to the internal ORDER-LINE entity. The external user can choose their own record and field names using this technique, we are not bound to use the internal names.

The syntax

RECORD NAME IS INVOICE (INVOICE-NO/C6, INVOICE-DATE/N6)

defines that the users record INVOICE consists of a six character

INVOICE-NO field and a six digit INVOICE-DATE field. The fields can be described in any order and need not contain all the fields of the corresponding internal entity. In addition, as will be described later, the record may contain fields from owning entities higher up the hierarchy.

Notice further that the field CUSTOMER-NAME/C40 maps onto the internal attribute CUSTOMER-NAME/C30. Changing field sizes is permitted, but clearly the Database Administrator should exercise care in using this facility.

The description for the ORDER record (mapping onto the ORDER-LINE entity) includes a field defined as

ORDER-QTY = QTY/N6

which defines a six digit field QTY which maps onto the ORDER-QTY attribute in the internal schema. Thus both field and record names be changed at the external schema interface.

In the external schema DDL, any field from a high-level record can instead be included within any owned record. Thus the INVOICE record can be defined as

RECORD INVOICE (INVOICE-NO/C6, INVOICE-DATE/N6, CUSTOMER-NAME/C40)

to create an external schema record some of whose fields come from the internal schema entity INVOICE and some from its owner, the CUSTOMER entity. Two examples of the use of this feature are given in Appendix 5. One converts a three-level entity structure into a two-level record structure while the other merges all three levels into one mapping onto the lowest level. This latter form compresses the hierarchy into a single flat-file, and is used as the interface module when using the query language QUILL on PYRAMID databases. This is a powerful facility in read-only modes, but an update in the INVOICE example above while INVOICE-DATE may be changed, CUSTOMER-NAME obviously cannot as it is not uniquely identified by the key INVOICE-NO. The update operation only changes fields at the mapped level to preserve the integrity of the database.

7.5 PYRAMID Mapping Code

The interface between the PYRAMID database and the user is through a mapping code module generated to transform the internal view of the data into the user or external view.

In recent years much attention has been given to the so called "fourth generation languages" which often include facilities for generating user programs or program fragments from high level parametric descriptions of the problem.

Prywes (1979) describes the Model II language, a non-procedural language which is processed by a generator to produce a PL/I program. They give an example of the use of the language to generate a master file update program.

Horvath (1980) describes DESP (Database-Extract-Sort-Print) which generates a full ANS COBOL program for both IDMS databases and serial files.

Dwyer (1977) generates COBOL programs to implement decision tables using a pre-processor approach, while Baxter (1976) translates RPG and generates COBOL programs. The technique used by Baxter has been used in the Pyramid Mapping Code Generator, but many refinements have been made to the basic idea because the unstructured and long-winded code produced by Baxter is no longer acceptable today. Nevertheless, the idea of simultaneously generating code to many sections and then sorting the code into order later is based on Baxter's work.

Alternative approaches of skeleton programs or of interpretive approaches to program development were rejected as being too slow for Pyramid, but they have been used successfully elsewhere - Bertrand (1980), Butters (1980).

The essential core of the technique is to write a skeleton COBOL program and assign section identifiers to each distinct part. In PYRAMID the section identifier is a two-letter code from AA through to ZZ. For example AA was allocated the IDENTIFICATION DIVISION, BA and BB to the CONFIGURATION and INPUT-OUTPUT SECTIONS respectively of the IDENTIFICATION DIVISION and so on. As each line of code is generated it is allocated to a specific code section and is also given a four digit sequence number (generated from one onwards in chronological order). The code section identifier and the sequence number form the standard COBOL sequence number in columns 1 through 6. The final stage of the generation process sorts the code on these six characters and the code is thus grouped by purpose (code section) and within each section by order of generation.

The use of the above technique means that the generator can make effectively a single pass through the dictionary and for each dictionary entry simultaneously generate code in several places in the target program.

For example, at the start of the generation process, the following skeleton code is generated

```
Section        Code

BB             INPUT-OUTPUT SECTION.
BB             FILE-CONTROL.

EA             INITIAL-PARAGRAPH.
EA                 MOVE ZERO TO RESULT.
EA                 IF FUNCTION = "NEW"
EA                      PERFORM NEW-DATA-BASE
EA                 ELSE IF FUNCTION = "OLD"
EA                          PERFORM OLD-DATA-BASE
EA                      ELSE IF FUNCTION = "RELEASE"
EA                                PERFORM RELEASE-DATA-BASE
                            ELSE PERFORM BRANCH-ON-RECORD-NAME.

TA             NEW-DATA-BASE.
TA                 IF DATA-BASE-OPEN-FLAG = "YES"
TA                        MOVE 101 TO RESULT
TA                 ELSE
TA                        PERFORM CREATE-DATA-BASE
TA                        PERFORM CLOSE-DATA-BASE
TA                        PERFORM UPDATE-DATA-BASE
TA                        MOVE "YES" TO DATA-BASE OPEN-FLAG.

TB             CREATE-DATA-BASE.

TC             UPDATE-DATA-BASE.

TD             CLOSE-DATA-BASE.
```

When a dictionary entry for a physical file (say CUSTOMERS) is located in the dictionary, then the following code is generated.

```
Section        Code

BB             SELECT INTERNAL-CUSTOMERS
BB                  ASSIGN TO etc.

CB             FD INTERNAL-CUSTOMERS
                    etc.

TB             OPEN OUTPUT INTERNAL-CUSTOMERS.

TC             OPEN I-O INTERNAL-CUSTOMERS.

TD             CLOSE INTERNAL CUSTOMERS.
```

The PYRAMID mapping code generator is involved in two ways

```
PYRAMID, BUILD          (on-line)

PYRAMID, BUILD, I=data (batch from file "data").
```

7.6 Hierarchic database queries

The Scan Sequential module SCANSQ (see A2.3) can be used to call either module SCANSF (see Fig. 5.2) or it can be made to call the DBMS module of mapping code produced by PYRAMID. In the former case the SEQUENT query program SQUERY is produced, and in the latter the PYRAMID query program PQUERY is constructed.

PYRAMID queries can be involved in two ways

```
PYRAMID, QUERY          (on-line)
PYRAMID, QUERY, I=data (batch from file "data").
```

The query program uses the QUILL query language in exactly the same way as SEQUENT, details of which were included in Chapter 5, Section 3.

CHAPTER 8

CONCLUSIONS

8.1 The database software in retrospect

While the software described in this thesis was being developed it is clear that there has been a shift from navigational models (hierarchic and network) towards the relational model. The decision then to build the hierarchic PYRAMID system is perhaps with hindsight not the best model to have implemented.

Remmen (1979) however can be quoted in defence:

> "... the quality of data structures does not depend primarily on the model being used, but on the insight of the designer.
>
> Education should not aim at advocating certain models exclusively but at using models in the right way. The only thing to be advocated is insight, which is to be achieved by appropriate education."

Since the PYRAMID model offers a fair degree of physical and logical data independence, there is scope for using the system to illustrate the common advantages that exist for both PYRAMID and commercially available DBMS's.

The desire to carry to extremes the non-procedurality of the actions in a QUILL statement has led to a language that seems unduly restrictive to experienced programmers. Non-programmers have reported no such disquiet however and they are the target users, not experienced programmers.

The QUILL language as implemented has been most effective for INVERSE databases where efficient use can be made of the inverted indexes.

The inability to use PYRAMID prime keys for rapid access to lower levels of the hierarchy clearly limits the use of QUILL to small hierarchic databases, and similar restrictions apply to SEQUENT.

The major aims of the software were however met. A stand alone query facility has been provided in SEQUENT, and INVERSE has permitted students to significantly reduce retrieval times for accessing a large data base of some 400,000 records. PYRAMID has allowed students to manipulate logical structures that are relatively independent of the physical structures. Both INVERSE and PYRAMID have provided privacy features, while INVERSE can produce a recovery audit trail.

All the above has been achieved without the running programs requiring excessive main memory. In fact, of the programs that a student would normally use, none uses more memory than the CYBER Loader used to link/load the programs into memory and they can thus all run with minimal memory limits. The only program of significant main memory size is the PYRAMID Mapping Code Generator and this will be run by the Data Base Administrator and then but rarely. (This can be contrasted to the S.A.I.T. CYBER where student memory limits are insufficient to use multiple key indexed sequential files in an on-line COBOL program - they have to be run as off-peak batch programs.)

8.2 Potential Development of the Software

Perhaps the most useful development for the software would be the development of network and relational models under the QUILL umbrella. It is easy to envisage a PYRAMID-like generator producing code to manipulate a logical view from a network data base. If this logical view happened to be a hierarchy then the QUILL language could be used to access the database as for PYRAMID.

The relational model would not fit as well under the QUILL umbrella. While it would be possible to derive a logical view of a single global relation formed as a join of all physical relations, and then to use QUILL as on the single relation as for SEQUENT, this accessing technique would take away both the power and beauty of the Relational Calculus. It may be better to abandon QUILL altogether and opt for a multi-file rather than single-file conceptual view for the query language.

The development of alternative query languages would allow a study of end-user/machine interactions to be carried out (following Schneiderman, Miller, Welty & Stemple etc.). Such a development would allow the various psychological theories to be subjected to rigorous examinations without the issue being clouded by having different databases, operating systems, hardware etc.

The development of a pre-processor for the PYRAMID system would be a relatively simple task if the syntax and code layout of the data sub-language was controlled - e.g. by having unique verbs to introduce DML statements and perhaps requiring such statements to appear on separate code lines.

Apart from these free-standing developments, it is clearly possible to add extra features to the existing programs - to add back in, in fact, many of those features deliberately left out during the design stage. An example might be to add an audit trail capability to the PYRAMID system. Each such addition however is one more thing for the student to learn, and also increases the size constraint on the running programs.

8.3 Concluding Remarks

Remmen (1979) has stated

> "The aim of every education is that the persons involved (students, pupils) gain a personal insight into the relevant subject matter.
>
> ...
>
> The insight mentioned can only be developed by a personal learning process of the student himself.
>
> ...
>
> The (happy) end of such a personal struggle can easily be regarded as the spontaneous manifestation of an 'aha'-experience."

Elsewhere in the same paper Remmen says

> "Experience in different learning situations has clearly shown that manipulation of data structures is the best way to promote the understanding of these structures."

The author of this thesis strongly endorses Remmen's views. Database concepts cannot realistically be taught using the so-called "purple cow" approach (I don't have a purple cow but if I describe its characteristics carefully enough hopefully my students will recognise such a beast when they see one).

So to teach database concepts students must be able to lay their hands on some DBMS software. The software described in this thesis offers an alternative to more costly commercial DBMS's and being less general should be easier to learn while still enabling all the major features to be used.

ERRATA

The following should be added to the list of references in pages REF-1 through REF-7.

Brodie, M.L. and Schmidt, J.W., 1981, Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group, Doc. No. SPARC-81-690.

Caelli, W.J., 1979, The CODASYL 1978 Data Base Proposals : An Overview and Discussion, Australian Computer Journal, Vol. 11, No. 2, pp 48-59.

Chamberlin, D.D., 1976, Relational Data-Base Management Systems, Australian Computing Surveys, Vol. 8, No. 1, pp 43-66.

Dee, E., Johnson, E.M., and King, P.J.H., 1973, An Example of Programming using the CODASYL Data Base Task Group Proposal with COBOL as the Host Language, Data Base Management, Infotech International State of the Art report, 1973, pp 455-488.

Deen, S.M., 1977, Fundamentals of Data Base Systems, McMillan, London.

Fry, J.P. and Sibley, E.H., 1976, Evolution of Data-Base Management Systems, ACM Computing Surveys, Vol. 8, No. 1, pp 7-42.

Kim, W., 1979, Relational Database Systems, ACM Computing Surveys, Vol. 11, No. 3, pp 185-211.

REFERENCES

Ansi, 1974, American National Standard - Programming language COBOL, Report ANSI X3.23-1974, ANSI, New York.

Antonacci, F., Dell'Oras, P., Spadevecchia, V.N. and Turtur, A., 1978, AQL: A problem-solving Query Language for Relational Data Bases, IBM J. Res. & Dev., Vol. 22, No. 5, pp. 541-559.

Ashton, D. and Wade, K., A Fourth Generation/End-User Language in a College Environment, Proc. Australian College of Adv. Education Conf., Aug. 1982, Launceston.

Astrahan, M.M. et al., 1979, System R: A Relational Data Base Management System, Computer, 1979, pp. 42-48.

Astrahan, M.M., Schkolnick, M. and Kim, W., 1980, Performance of the System R Access Path Selection Mechanism, Information Processing 80, Proc. IFIC Congress 80, Melbourne, pp. 487-492.

Atre, S., 1980, Data Base: Structural Techniques for Design, Performance & Management, Wiley, New York.

Baxter, A.Q. and Johnson, R.R., 1978, A Block Structure Query Language for Accessing a Relational Data Base, ACM SIGIR, Vol. 13, No. 19,

Baxter, J.D. and Vincent, G.J., 1976, Report Program Generator (Single File Input System) Users Manaual, Univ. Melbourne.

BCS, 1971, Proc. October 1971 Conference on APRIL 71 REPORT, British Computer Society.

BCS, 1977, The British Computer Society Data Dictionary Systems Working Party Report, reprinted in ACM SIGMOD RECORD, Vol. 9, No. 2.

Benbasat, I. and Dexter, A.S., 1981, An Experimental Study of the Human/Computer Interface, Comm. ACM, Vol. 24, No. 11, pp. 752-762.

Bertrand, O.P. and Daudenarde, J.J., 1980, USAGE: Generating Interactive Application Programs from Grammatical Descriptions, ACM DATA BASE, Vol. 11, No. 3, pp. 76-83.

Bird, R.M., Newsbaum, J.B. and Trelftzs, J.L., 1978, Text File Inversion: An Evaluation, ACM SIGARCH, Vol. 7, No. 2, pp. 42-50.

Borman, L., Chalice, R.,, Dillamen, D., Dominick, W. and Kobbe, R., 1976, RIQS-Remote Information Query System Users Manual, Northwestern Univ. Report 74-003.

Bourne, T.J., 1979, The Data Dictionary System in Analysis and Design, ICL Technical Journal, Nov. 1979, pp. 292-298.

Boyce, R.F., Chamberlin, D.D., King, W.F. III and Hammer, M.M., 1975, Specifying Queries and Relational Expressions: The SQUARE Data Sublanguage, Comm ACM, Vol. 18, No. 11, pp. 621-628.

Bonczek, R.H., Cash, J.I. and Whinston, A.B., 1977, A Transformational Grammar-Based Query Processor for Access Control in a Planning System, ACM Transactions on Database Systems, Vol. 2, No. 4, pp. 326-338.

Bradley. J., 1982, File and Data Base Techniques, Holt, Rinehart and Winston, New York.

Butters, E.H. and Seymour, C.M., 1980, Generalized Systems: Reducing High Cost of Application Development, ACM DATABASE, Vol. 11, No. 3, pp. 99-105.

Burns, D., 1975, Data Handling Techniques in ROBOT, Data Base Systems, INFOTECA, pp. 239-263.

Cagan, C., 1973, Data Management Systems, Melville, Los Angeles.

Canning, R.G., ed., 1981, A New View of Data Dictionaries, EDP Analyzer, Vol. 19, No. 7, July 1981.

Canning, R.G., ed., 1982, Relational Database Systems are Here, EDP Analyzer, Vol. 20, No. 10, Oct. 1982.

Cardenas, A.F., 1975, Analysis and Performance of Inverted Data Base Structures, Comm ACM, Vol. 18, No. 5, pp. 253-263.

Carter, R.J. and McVitie, D.G., 1969, PEARL Preliminary System Description, ICL Report K/AD n37.

Chamberlin, D.D. et al., 1981, A History and Evaluation of System R, Comm ACM, Vol. 24, No. 10, pp. 632-646.

Champine, G.A., 1979, Current Trends in Data Base Systems, COMPUTER, May 1979, pp. 27-40.

Clemons, E.K., 1981, Design of an External Schema Facility to Define and Process Recursive Structures, ACM Trans. on Database Systems, Vol. 6, No. 2, pp. 295-311.

CODASYL, 1969, CODASYL Data Base Task Group, October 1969 Report, CODASYL, 1969.

CODASYL, 1971, CODASYL Data Base Task Group, April 1971 Report, ACM, New York, 1971.

CODASYL, 1979, A Status Report on the Activities of the CODASYL End User Facilities Committee, ed. H.C. Lefkovits, ACM SIGMOD Record, Vol. 10, Nos. 2 and 3.

Codd, E.F., 1970, A Relational Model of Data for Large Shared Data Banks, Comm ACM, Vol. 13, No. 6, pp. 377-387.

Codd, E.F., 1971a, Further Normalization of the Data Base Relational Model, Data Base Systems, ed. Rustin, Prentice-Hall Inc., New Jersey, pp. 33-64.

Codd, E.F., 1971b, Relational Completeness of Data Base Sublanguages, Data Base Systems, ed. Rustin, Prentice-Hall Inc., New Jersey, pp. 65-98.

Codd, E.F., 1971c, Normalized Data Base Structure: A Brief Tutorial, 1971 ACM SIGFIDET Workshop, ed. Codd & Dean, San Diego, Calif. pp. 1-17.

Codd, E.F., 1974, Seven Steps to Rendezvous with the Casual User, Data Base Management, ed. Klimbie and Kofferman, North-Holland, Amsterdam, pp. 179-199.

Codd, E.F., 1979, Extending the Data Base Relational Model to Capture More Meaning, Australian Computer Science Comm., Vol. 1, No. 1, March 1979, pp. 5-48.

Codd, E.F., 1980, Data Models in Database Management, ACM SIGMOD Record, Vol. 11, No. 2, Feb. 1981, pp. 112-114.

Cohen, L.J., ed., 1978, Data Base Management Systems, Q.E.D. Information Sciences, Mass., U.S.A.

Combes, D., 1980, REMOTE-QBE: An Overview, Australian Computer Science Communications, Vol. 2, No. 5, pp. 459-469.

Dadam, P. and Schlageter, G., 1980, Recovery in Distributed Databases Based on Non-synchronized Local Checkpoints, Information Processing 80, Proc. of IFIP Congress 80, Melbourne, pp. 457-462.

Dahl, O-J., Dijkstra, E.W. and Hoare, C.A.R., 1972, Structured Programming, Academic Press, Letchworth, England.

Date, C.J., 1977, An Introduction to Database Systems, 2nd ed., Addison-Wesley, Mass., U.S.A.

Davenport, R.A., 1980, Data Administration - the Needs for a New Function, Information Processing 80, Proc. of IFIP Congress 80, Melbourne, pp. 505-510.

DEC., 1982, Introduction to VAX-11 DATATRIEVE, Digital Equipment Corporation Manual AA-K082A-TE, Mass., U.S.A.

Deen, S.M., 1980, A canonical schema for a generalised data model with local interfaces, The Computer Journal, Vol. 23, No. 3, pp. 201-206.

Deen, S.M., Nikodem, D. and Vashishta, A., 1981, The design of a canonical database system (PRECI), The Computer Journal, Vol. 24, No. 3, pp. 200-209.

Drake, R.W. and Smith, J.L., 1971, Some Techniques for File Recovery, Australian Computer Journal, Vol. 3, No. 4, pp. 162-170.

Dwyer, B., 1977, How to Write Decision Tables for Use with COPE, Techsearch Inc., S.A. Institute of Technology, Adelaide.

Evans, M., 1982, Software Engineering for the COBOL Environment, Communications of the ACM, 25, pp. 874-882.

Fagin, R., 1977, Multivalued Dependencies and a New Normal Form for Relational Databases, ACM Transactions on Database Systems, Vol. 2., No. 3, pp. 262-278.

Fagin, R., 1979, Normal Forms and Relational Data Base Operators, ACM SIGMOD International Conf. on Management of Data, Boston, Mass.

Fossom, B.M., 1974, Data Base Integrity as Provided for by a Particular Data Base Management System, Data Base Management, North-Holland, Amsterdam, pp. 271-287.

Gudes, E., 1977, Teaching data base systens using Date and Computing Surveys, ACM SIGMOD Record, Vol. 9, No. 1, pp. 47-49.

Harder, T. and Reuter, A., 1979, Optimization of Logging and Recovery in a Database System, Data Base Architecture, North-Holland, Amsterdam, pp. 151-168.

Harris, L.R., 1978, The ROBOT System: Natural Language Processing Applied to Data Base Query, Proc. ACM 1978 Annual Conf., Dec. 4-6, Washington, D-C, U.S.A., pp. 165-172.

Haskell, R. and Harrison, P.G., 1980, System Conventions for non procedural languages, The Computer Journal, Vol. 23, No. 2, pp. 132-141.

Hawryskiewycz, I.T., 1980, Multi-model Data Base Architecture, Australian Computer Science Communications, Vol. 25, No. 4, pp. 400-416.

Hawryskiewycz, I.T., 1979, The Evolution of Data Base Technology and its Effect on the Teaching of Data Base, Proc. Colleges of Adv. Education Conf., Bendigo, 1979, pp. 3.11-3.19.

Hendrix, G.G., Sacerdoti, E.D., Sagolowicz, D. and Slocum, J., 1978, Developing a Natural Language Interface to Complex Data, ACM Transactions on Database Systems, Vol. 3, No. 2, pp. 105-147.

Hill, E., Jr., 1978a, Analysis of an Inverted Data Base Structure, ACM SIGIR, Vol. 13 No. 1, pp. 37-64.

Hill, E., Jr., 1978b, A Comparative Study of Very Large Data Bases, Springer-Verlag, Berlin.

Hill, I.D., 1972, Wouldn't it be nice if we could write computer programmers in ordinary English - or would it?, Computer Bulletin, Vol. 16, No. 6, pp. 306-312.

Honkanen, P.A., 1983, Installation of a Commercial Database Management System in a University Environment, ACM SIGCSE Bulletin, Vol. 15, No. 1, pp. 211-219.

Horvath, P.J., 1980, DESP - A COBOL Program Generator for IDMS Databases and Serial Files, ACM DATA BASE, Vol. 11, No. 3, pp.46-55.

ICL, 1968, Specification of PRIOR, ICL Business Information Systems Dept. report.

ICL, 1969, PLUTO System, ICL Tech. Pub. 4150, London, England.

Jackson, M., 1983, System Development, Prentice-Hall International, New Jersey, U.S.A.

Johnson, J..S and Webster, D.B., 1982, Updating an Inverted Index - a Performance Comparison of Two Techniques, The Computer Journal, Vol. 25, No. 2, pp. 169-175.

Kaplan, S.J. and Ferris, D., 1982, Natural Language in the DP World, Datamation, Vol. 2, No. 9, pp. 114-120.

Kaunitz, J. and Van Ekert, L., 1981, Data Base Backup - The Problem of Very Large Data Bases, Australian Computer Journal, Vol. 13, No. 4, pp. 136-142.

Kent, W., 1978, Data and Reality - basic assumptions in data processing revisited, North-Holland, Amsterdam.

Kent, W., 1983, A Simple Guide to Five Normal Forms in Relational Database Theory, Comm. ACM, Vol. 26, No. 2, pp. 120-125.

Knuth, D.E., 1973, The Art of Computer Programming - (Vol. 1) - Fundamental Algorithms, Addison-Wesley, Mass.

Kroenke, D., 1983, Database Processing, 2nd ed., SRA, Chicago.

Lawrence, M.J., 1979, The Computer Data Base Decision, Australian Computer Journal, Vol. 11, No. 1, pp. 13-20.

Lien, Y.L., 1981, Hierarchical Schemata for Relational Databases, ACM Trans. on Database Systems, Vol. 6, No. 1, pp. 48-69.

Liu, J.W.S., 1976, Algorithms for Parsing Search Queries in Systems with Inverted File Organization, ACM Transactions on Database Systems, Vol. 1, No. 4, pp. 299-316.

Ling, T-W, Tompa, F.W. and Kameda, T., 1981, An Improved Third Normal Form for Relational Databases, ACM Transactions on Database Systems, Vol. 6, No. 2, pp. 329-346.

LOGICA, 1982, RAPPORT-3, Designing and Using a Database, Logica Ltd. U.K.

Lyon, J.K., 1976, The Database Administrator, Wiley, New York.

Mayne, A., 1981, Database Management Systems: A technical review, NCC Publications, Manchester, England.

McDonnell, K.J., 1976, The Design of Associative Keylists (Secondary Indexes), Australian Computer Journal, Vol. 8, No. 1, pp. 13-18.

McDonnell, K.J., 1977, An Inverted Index Implementation, The Computer Journal, Vol. 20, No. 2, pp. 116-123.

McDonnell, K.J., 1979, Systems for Teaching Database Concepts, Proc. Austtralian Colleges of Adv. Education Conf., Aug. 1981, Perth, pp. 179-193.

McLeod, D., 1978, A Semantic Data Base Model and its Associated Structured User Interface, Report MIT/LCS/TR-214, MIT, Cambridge, Mass., U.S.A.

Mercz, L.I., 1979, Issues in Building a Relational Interface on a CODASYL DBMS, Proc. IFIP Working Conf. on Data Base Architecture, Data Base Architecture, ed., G. Bracchi & G.M. Nijssen, North-Holland, Amsterdam.

Michaels, A.S., Mittman, B. and Carlson, C.R., 1976, A Comparison of Relational and CODASYL Approaches to Data-Base Management, ACM Computing Surveys, Vol. 8, No. 1, pp. 125-151.

Miller, L.A., 1981, Natural-Language Programming: Styles, Strategies and Contrasts, IBM Perspectives in Computing, Vol. 1, No. 4, pp. 22-33.

Montgomery, A.Y., 1980, A Short Essay on Some Problems with Teaching of Computing in Australian Government Institutions Today, Australian Computer Science Communications, Vol. 2, No. 4, pp. 396-399.

Nijssen, G.M., 1983, What can CAI learn from the DATABASE world?, Proc. Conf. on Computer-Aided Learning in Tertiary Education. Brisbane, Qld., Sept. 1983.

Olle, T.W., 1973, Data Base Management Systems Software: The CODASYL DBTG Proposals Data Base Management, Infotech International State of the Art Report, 1973, pp. 407-428.

Peat, L.R., 1982, Practical Guide to DBMS Selection, Walter de Gruyter, Berlin.

Prywes, N.S., Pnueli, A. and Shastry, S., 1979, Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development, ACM Transactions on Programming Languages and Systems, Vol. 1, No. 2, pp.196-217.

Reisner, P., 1981, Human Factors Studies of Database Query Languages: A Survey and Assessment, ACM Computing Survey, Vol. 13, No. 1, pp. 13-31.

Relational Software Inc., 1983, ORACLE Terminal User Guide, Version 3.1, U.S.A.

Remmen, F., 1979, Education in Databases - the Coaching of a Learning Process, Post Secondary and Vocational Education in Data Processing, ed. Jackson, H.L.W. and Wiechers, G., North-Holland, Amsterdam, pp. 137-150.

Robinson, H., 1981, Database Analysis and Design. Chartwell-Bratt, England.

Robinson, M.A., 1981, A Review of Data Base Query Languages, Australian Computer Journal, Vol. 13, No. 4, pp. 143-159.

Rowe, L.A. and Stonebraker, M., 1981, Architecture of Future Data Base Systems, ACM SIGMOD Record, Vol. 11, No. 1, pp. 30-44.

Samet, P.A., ed., 1981, Query Languages - A unified approach, Heyden & Son/BCS Monographs in Information.

Schneiderman, B., 1978, Improving the Human Factors Aspect of Database Interactions, ACM Transactions on Database Systems, Vol. 3, No. 4, pp. 417-439.

Simsion, G.C. and Symington, J.A., 1981, A Comparison of Network and Relational Data Base Architectures in a Commercial Environment, Australian Computer Journal, Vol. 13, No. 4, pp. 122-126.

Sockut, G.H., 1981, Comparison and Mapping of the Relational and Codasyl Data Models -- An Annotated Bibliography, ACM SIGMOD Record, Vol. 11, No. 3, pp. 55-68.

Software AG, 1980, ADABAS-M Application Programmers Manual, Software AG Manual ADM-110-030, Darmstadt, W. Germany.

Stamen, J. and Costello, W., Evaluating Database Languages, Datamation, Vol. 27, No. 5, pp. 116-122.

Stonebraker, M., Wong, E., Kreps, P. and Held, G., 1976, The Design and Implementation of INGRES, ACM Transactions on Database Systems, Vol. 1, No. 3, pp. 189-222.

Su, S.Y.W. and Emam, A., 1978, CASDAL: CASSM's DAta Language, ACM Transactions on Database Systems, Vol. 3, No. 1, March 1978, pp. 57-91.

Tagg, M.R., 1981, Query Languages for some current DBMS, Proc. First British National Conference on Databases, Cambridge, England, July 1981, pp. 99-118.

Tagg, R.M., 1983, Interfacing a Query Language to a CODASYL DBMS, ACM SIGMOD Record, Vol. 13, No. 3, pp. 46-64.

Triance, J.M., 1978, Discussion and correspondence: A study of COBOL portability, The Computer Journal, Vol. 21, No. 3, pp. 278-281.

Tsichritzis, D.C. and Lochovsky, F.H., 1976, Hierarchical Data-Base Management: A Survey, ACM Computing Surveys, Vol. 8, No. 1, pp. 105-123.

Tsichritzis, D.C., 1977a, Research Directions in Database Management Systems, ACM SIGMOD Record, Vol. 9, No. 3, pp. 26-41.

Tsichritzis, D.C. and Lockovsky, F.H., 1977b, Data Base Management Systems, Academic Press, New York.

Verhofstad, J.S.M., 1978, Recovery Techniques for Database Systems, Computing Surveys, Vol. 10, No. 2, pp. 167-195.

Verhofstad, J.S.M., 1979, Recovery Based on Types, Data Base Architecture, North-Holland, Amsterdam, pp. 125-139.

Vetter, M. and Maddison, R.N., 1981, Database Design Methodology, Prentice-Hall International, London.

Welty, C. and Stemple, D.W., 1981, Human Factors Comparison of a Procedural and a Non-Procedural Query Language, ACM Transactions on Database Systems, Vol. 6, No. 3, pp. 464-485.

Yu, C.T., Luk, W.S. and Siu, M.K., 1978, On the Estimation of the Number of Desired Records with Respect to a Given Query, ACM Transactions on Database Systems, Vol. 3, No. 1, March 1978, pp.41-56.

Zaniolo, C., 1979, Multimodel External Schemas for CODASYL Data Base Management Systems, Proc. IFIP Working Conf. on Data Base Architecture, Data Base Architecture, ed. G. Bracchi and G.M. Nijssen, North-Holland, Amsterdam.

APPENDIX 1 - SYNTAX DESCRIPTIONS

This appendix contains a summary of the syntax description notation and then a formal definition of the syntax of the following languages:

(a) The QUILL Query/Update Language.

(b) The PYRAMID External Schema DDL.

(c) The PYRAMID Internal Schema DDL.

A1.1 Syntax Description Notation

In these syntax descriptions the following notation is adopted:

UPPER CASE	Special words of the various languages. They must be written exactly as specified. In general they should not be used except in their specified content (e.g. do not use as field/record names).
UNDERLINED UPPER CASE	These special words are mandatory whenever the format in which they occur is used. Special words that are not underlined are optional "noise" words.
lower case words	Generic terms which must be replaced by words, names or values supplied by the user. Within any given form if a generic term is repeated, each occurrence is identified by an appended integer (e.g. entity-name-1, entity-name-2).
Brackets [ ]	These surround an optional portion of a format. The entire contents of the brackets can be included or omitted as desired. If the brackets contain vertically stacked descriptions then only one of these descriptions can be used,

$$\left(\text{e.g.} \quad \left| \begin{matrix} a \\ b \\ c \end{matrix} \right| \equiv \begin{matrix} \text{at least no occurrences} \\ \text{at most one occurrence} \end{matrix} \right)$$

Braces { }	Only one of the vertically stacked descriptions can be used, $\left(\text{e.g.} \left\{\begin{matrix} a \\ b \\ c \end{matrix}\right\} \equiv \begin{matrix}\text{at least one occurrence} \\ \text{at most one occurrence}\end{matrix}\right)$ Braces are also used to enclose mandatory constructs which may be repeated.
Bars || ||	Each of the vertically stacked descriptions may occur in any order. Each description can occur only once, $\left(\text{e.g.} \left\Vert\begin{matrix} a \\ b \\ c \end{matrix}\right\Vert \equiv \begin{matrix}\text{at least one occurrence} \\ \text{at most one occurrence of each}\end{matrix}\right)$
Elipses ...	Indicates that the description immediately preceeding the ellipses and enclosed in brackets or braces can be repeated if desired.
Punctuation symbols	Generally required unless enclosed in brackets or specifically noted as optional. In general, commas (,) and semicolons (;) are optional and can in fact be used wherever a space can appear. Periods/fullstops (.) are mandatory at the ends of sentences.
Angle brackets < >	These surround parts of the description (gnerally clauses) which are defined later in the syntax descripti
::=	The construct to the left of the ::= symbol is defined by the description to the right of the symbol.

A1.2 The QUILL Query/Update Language Syntax

```
query ::=

        || <action> ...  ||
        || <qualifier>   ||


action ::=

        { <print-action>      }
        { <display-action>    }
        { <sum-action>        }
        { <average-action>    }
        { <add-action>        }
        { <subtract-action>   }
        { <multiply-action>   }
        { <divide-action>     }
        { <increase-action>   }
        { <decrease-action>   }
        { <set-action>        }
        { <generate-action>   }


sum-action ::=

        SUM <field-list>


average-action ::=

        AVERAGE <field-list>


add-action ::=

        ADD number TO field-name
```

subtract-action ::=

SUBTRACT number FROM field-name

multiply-action ::=

MULTIPLY field-name BY number

divide-action ::=

DIVIDE field-name BY number

increase action ::=

INCREASE field-name BY number [%]

decrease action ::=

DECREASE field-name BY number [%]

set-action ::=

SET field-name TO <literal>

generate action ::=

GENERATE <field-list>

field-list ::=

field-name

(<field-name> ...)

literal ::=

{ number
alphanumeric-literal
string }

A1.3 The PYRAMID External Schema DDL Syntax

external schema description ::=

[<mode-statement>]

[<create-statement>]

<external-schema-statement>

{<record-description>} ...

mode-statement ::=

MODE IS { BATCH | INTERACTIVE } .

create-statement ::=

NEW

external-schema-statement ::=

EXTERNAL SCHEMA NAME IS external-schema-name.

record-description ::=

RECORD NAME IS record-name

[= equivalent-record-name]

[({<item-description>}...)].

item-description ::=

item-name [= equivalent-item-name]

<item-format>

item-format ::=

/ <item-type> item-length

item-type ::=

$$\left\{ \begin{array}{c} \underline{C} \\ \underline{N} \end{array} \right\}$$

A1.4 The PYRAMID Internal Schema DDL Syntax

internal schema description ::=

[<mode-statement>]

[<create-statement>]

<internal-schema-statement>

{<file-description>} ...

mode-statement ::=

MODE IS { BATCH | INTERACTIVE } .

create-statement ::=

NEW DICTIONARY

internal-schema-statement ::=

INTERNAL SCHEMA NAME is internal-schema-name.

file-description ::=

<file-statement>

{<entity-description>} ...

file-statement ::=

FILE NAME IS file-name

|| <organisation-clause>
<access-clause>
<assign-clause> || .

organization-clause ::=

ORGANIZATION IS file-organization

access-clause ::=

ACCESS MODE IS access-mode

assign-clause ::=

ASSIGN to assignment-name

entity-description ::=

<entity-clause>

|| <owner clause> ||
|| <key clause> ||

{<field-description>} ...

entity-clause ::=

ENTITY NAME IS entity-name-1

owner-clause ::=

OWNER NAME IS entity-name-2

key-clause ::=

KEY NAME IS field-name-1

field-description ::=

{ field-name-2
  (<field-description> ...)
  <field-format> }

```
field-format ::=

       / <field-type> field-length


field-type ::=

      { C }
      { N }
```

APPENDIX 2

STRUCTURE DIAGRAMS

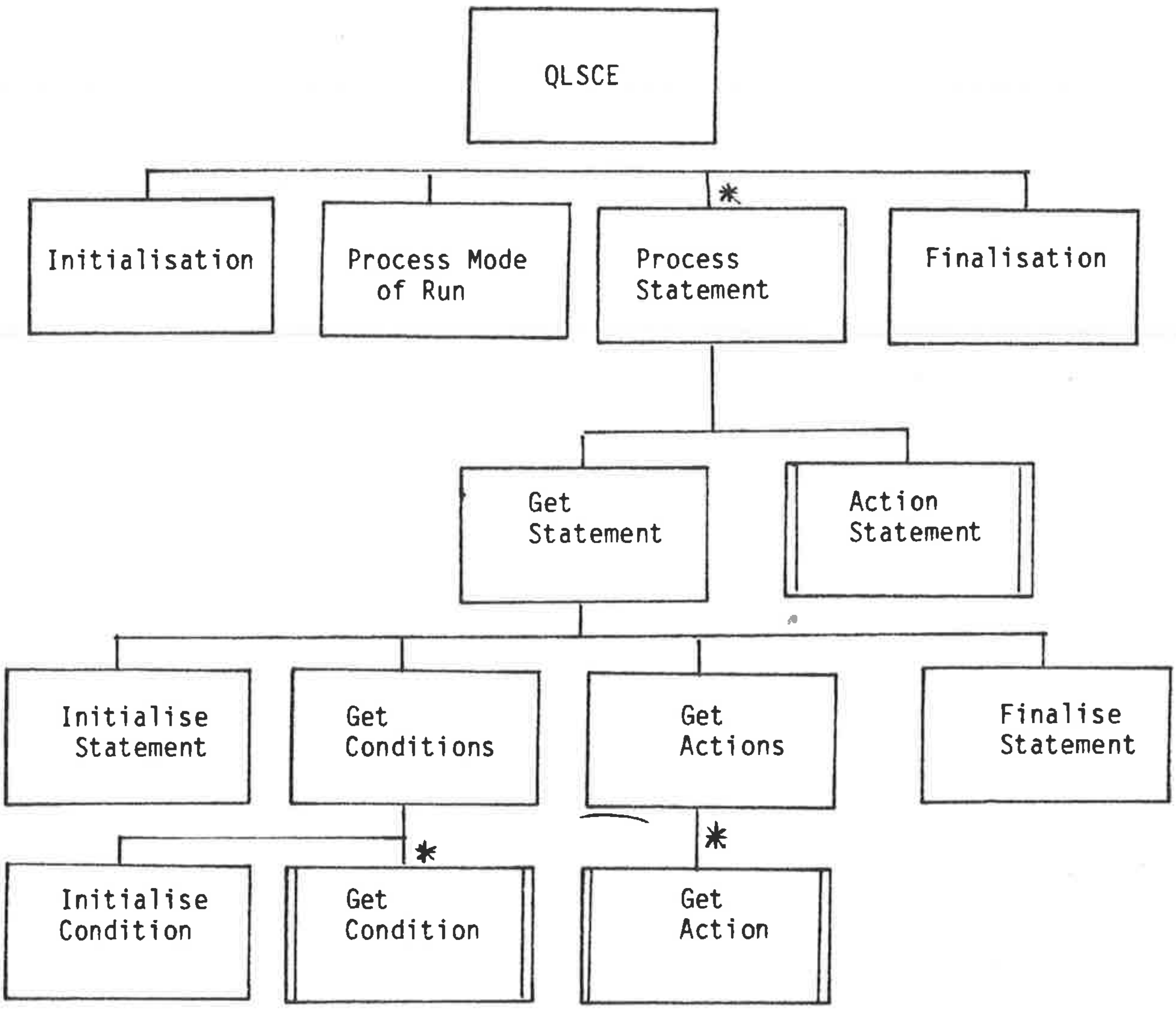
This appendix contains structure diagrams for the various programs and sub-programs of the database system.

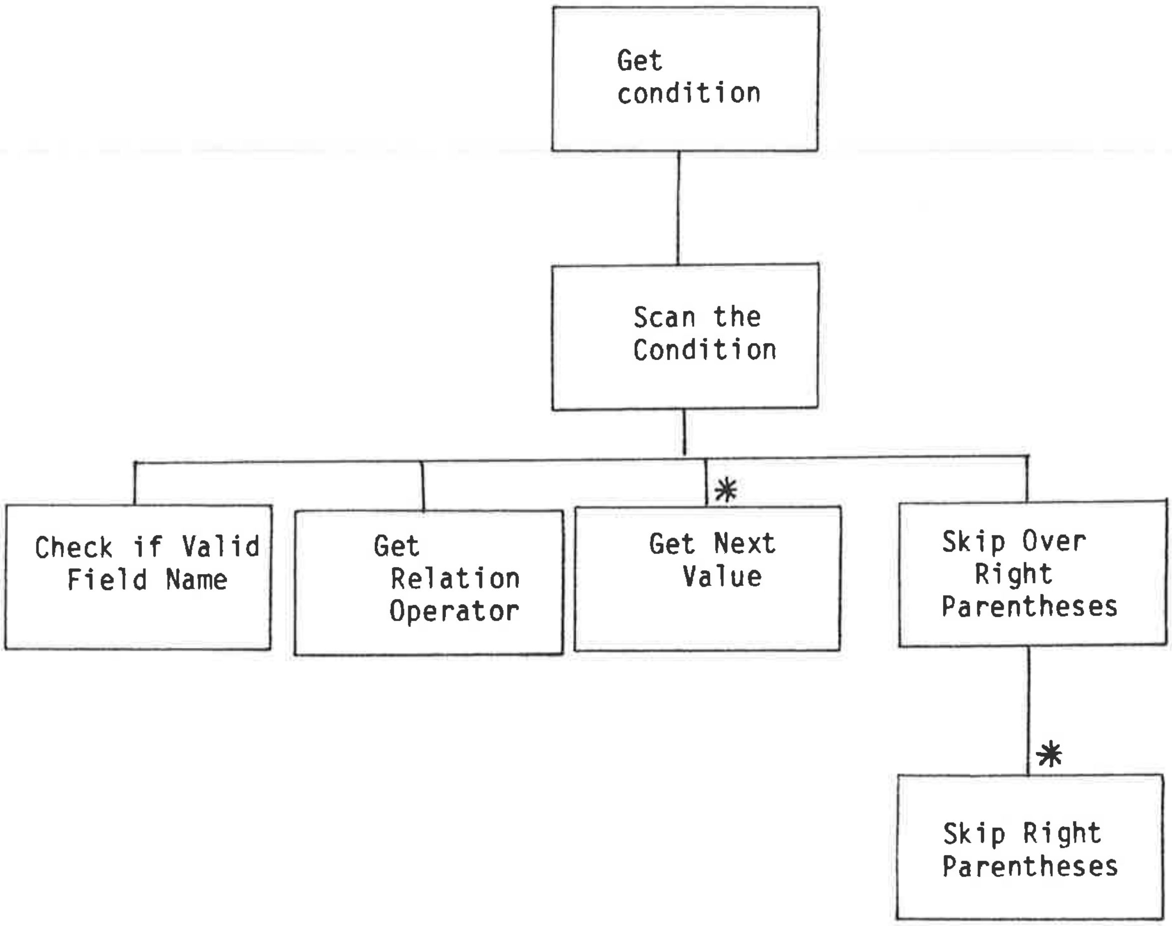
The conventions used for the charts are basically those of Jackson (1983).

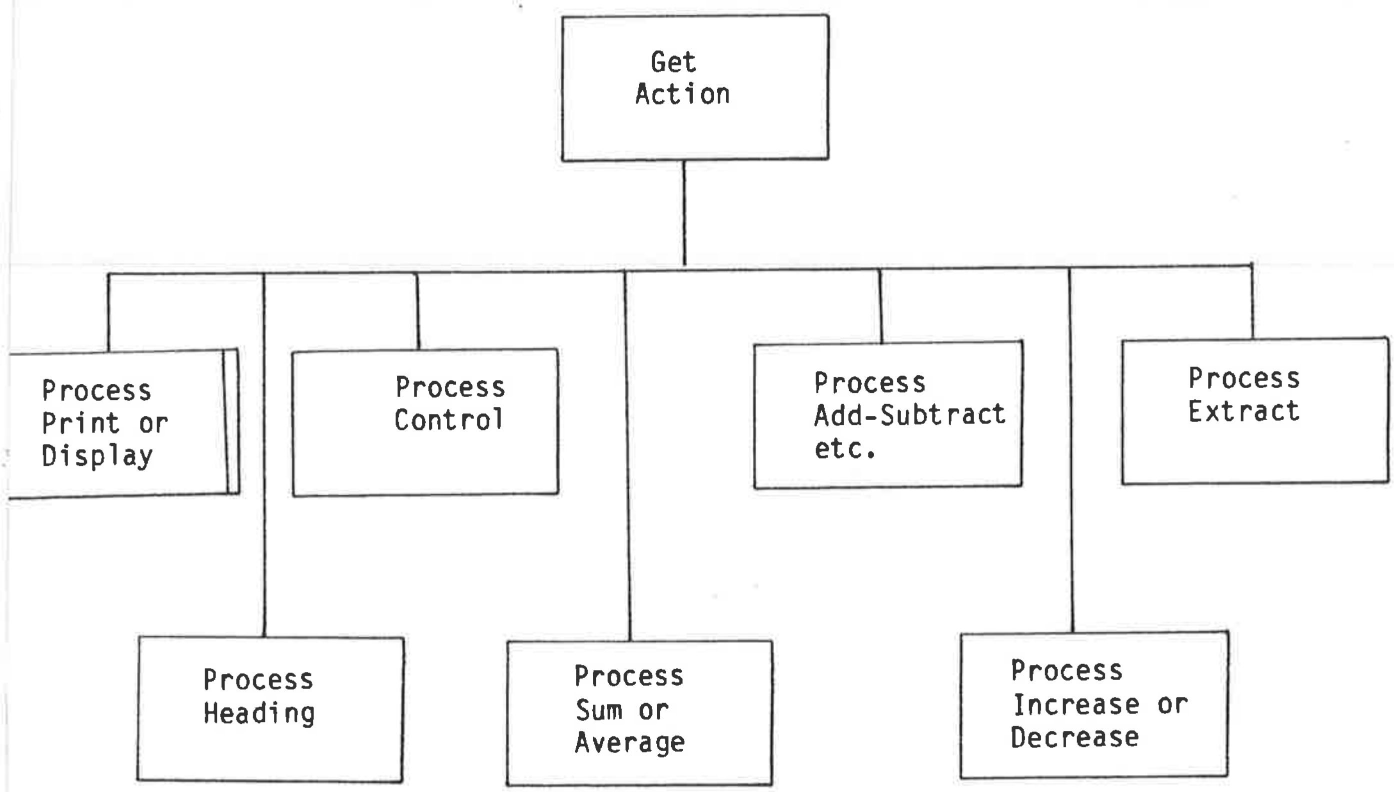
- rectangles indicate processes to be performed;
- processes are activated in a top-down, left-to-right order;
- an * indicates the process is activated repetitively (zero or more times);
- an 0 indicates that one of the sub-processes is selected;
- a double vertical border to a rectangle indicates that the process is further sub-divided on a subsequent chart.

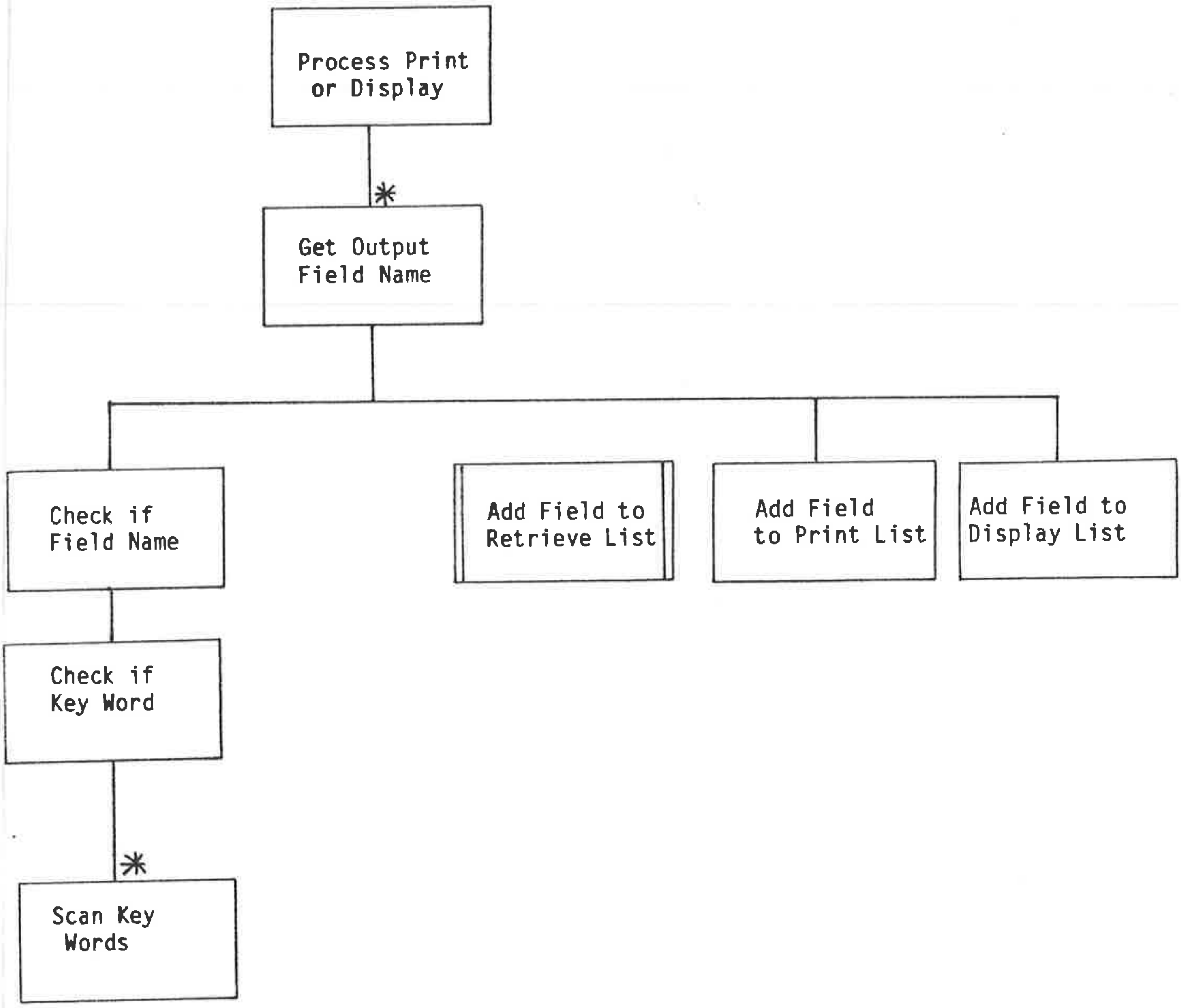
The following charts appear:

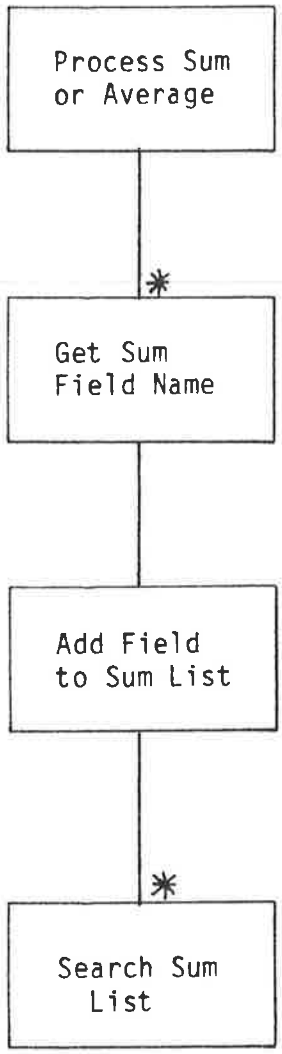
A2.1 QUILL Query/Update Language (QLSCE)

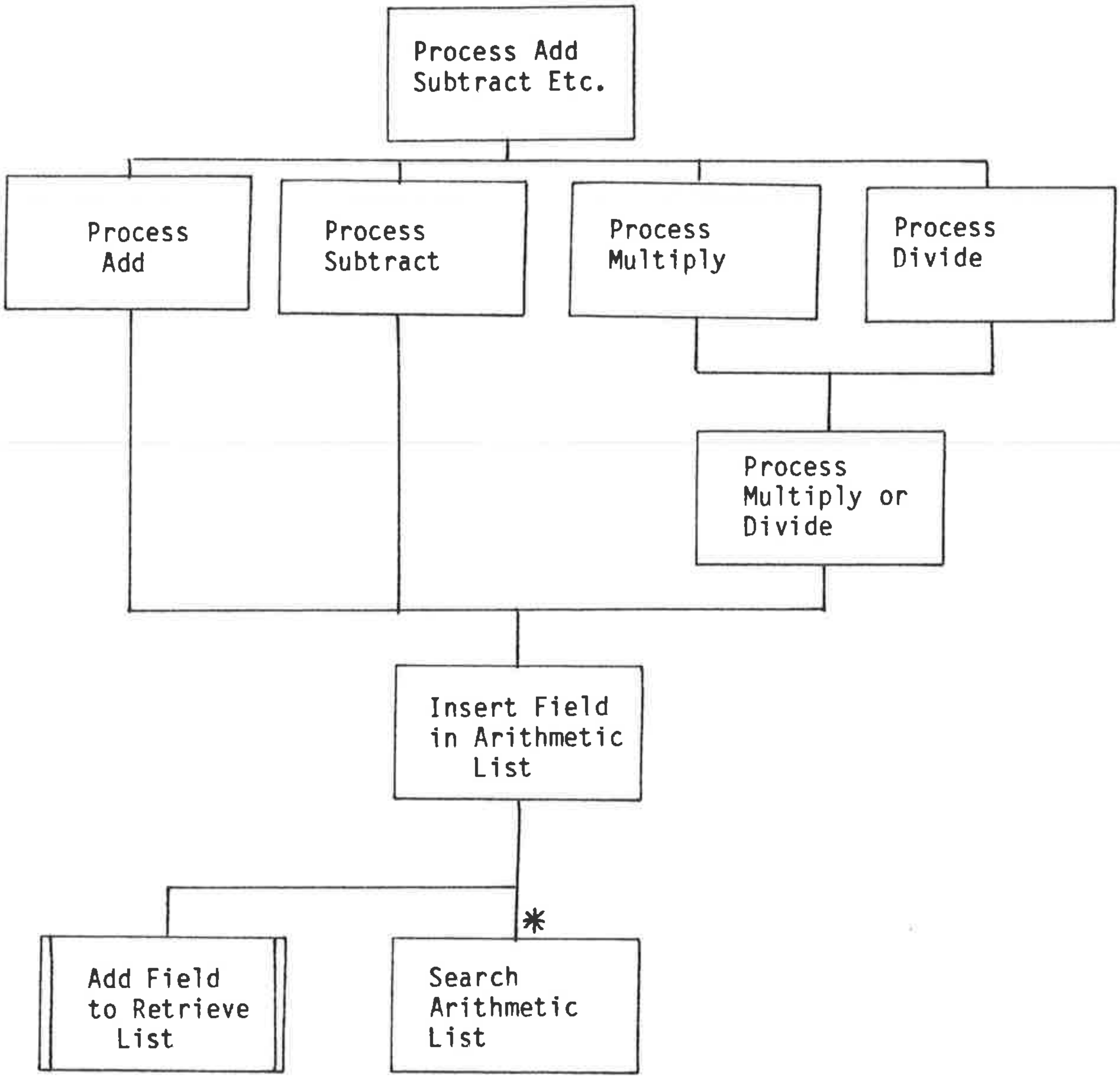


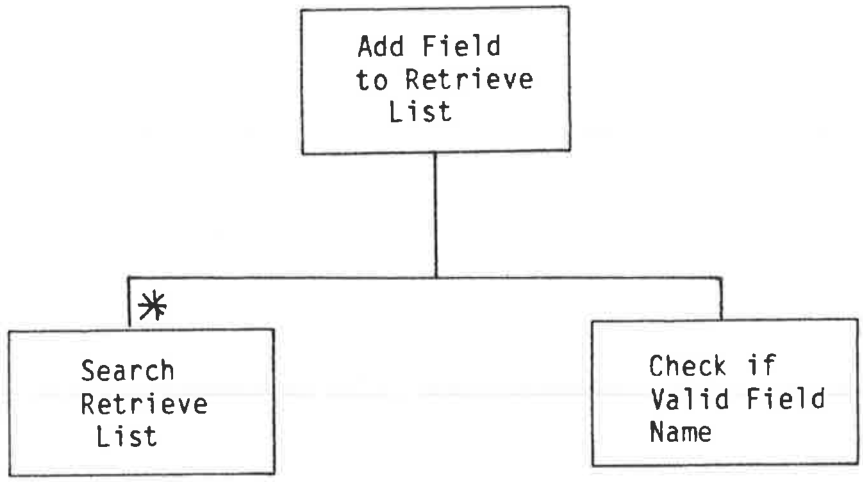

Get condition
Scan the Condition
Check if Valid Field Name
Get Relation Operator
*
Get Next Value
Skip Over Right Parentheses
*
Skip Right Parentheses

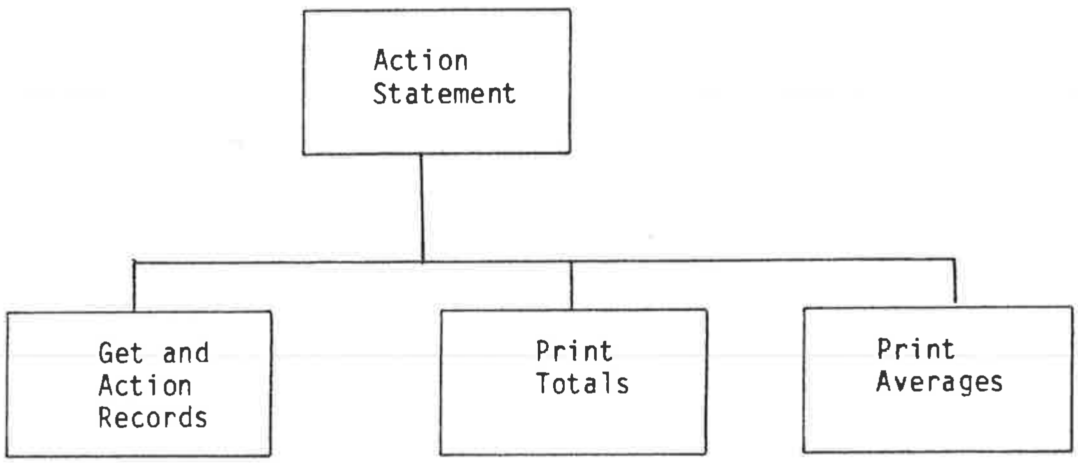

Get Action
Process Print or Display
Process Control
Process Add-Subtract etc.
Process Extract
Process Heading
Process Sum or Average
Process Increase or Decrease

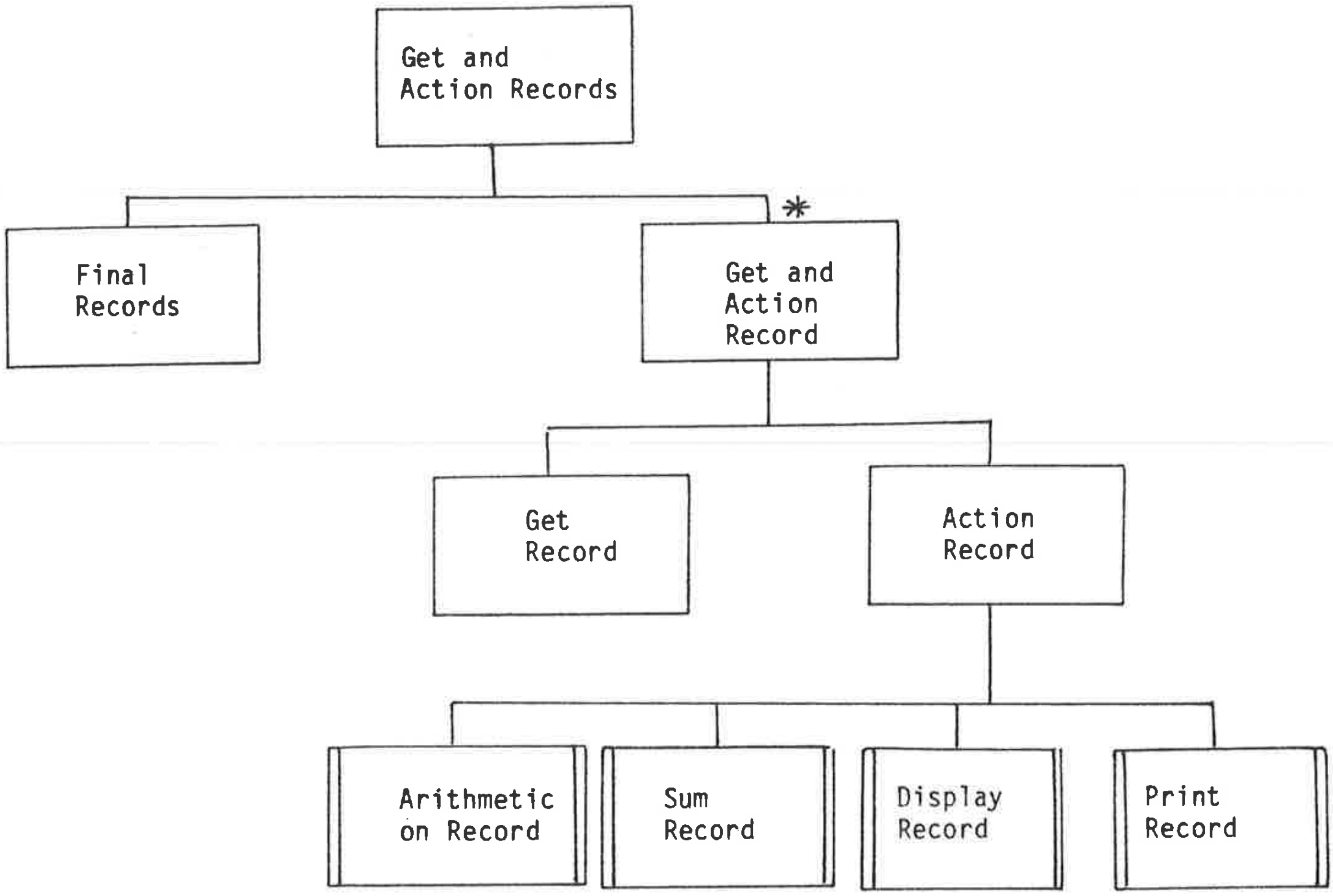

Process Print or Display
*
Get Output Field Name
Check if Field Name
Add Field to Retrieve List
Add Field to Print List
Add Field to Display List
Check if Key Word
*
Scan Key Words


Process Sum or Average
*
Get Sum Field Name
Add Field to Sum List
*
Search Sum List

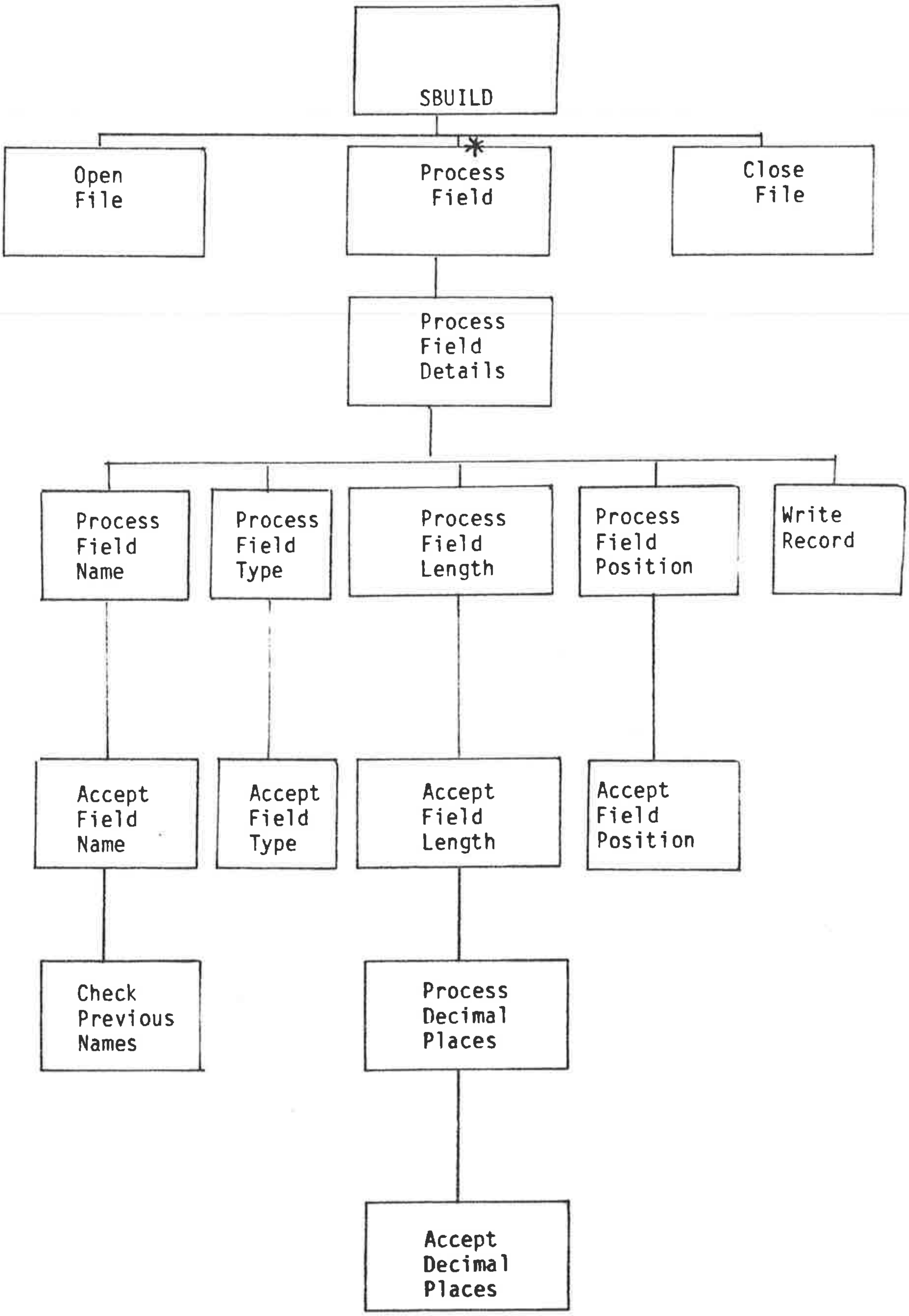

Process Add Subtract Etc.
Process Add
Process Subtract
Process Multiply
Process Divide
Process Multiply or Divide
Insert Field in Arithmetic List
*
Add Field to Retrieve List
Search Arithmetic List


Add Field
to Retrieve
List
*
Search
Retrieve
List
Check if
Valid Field
Name

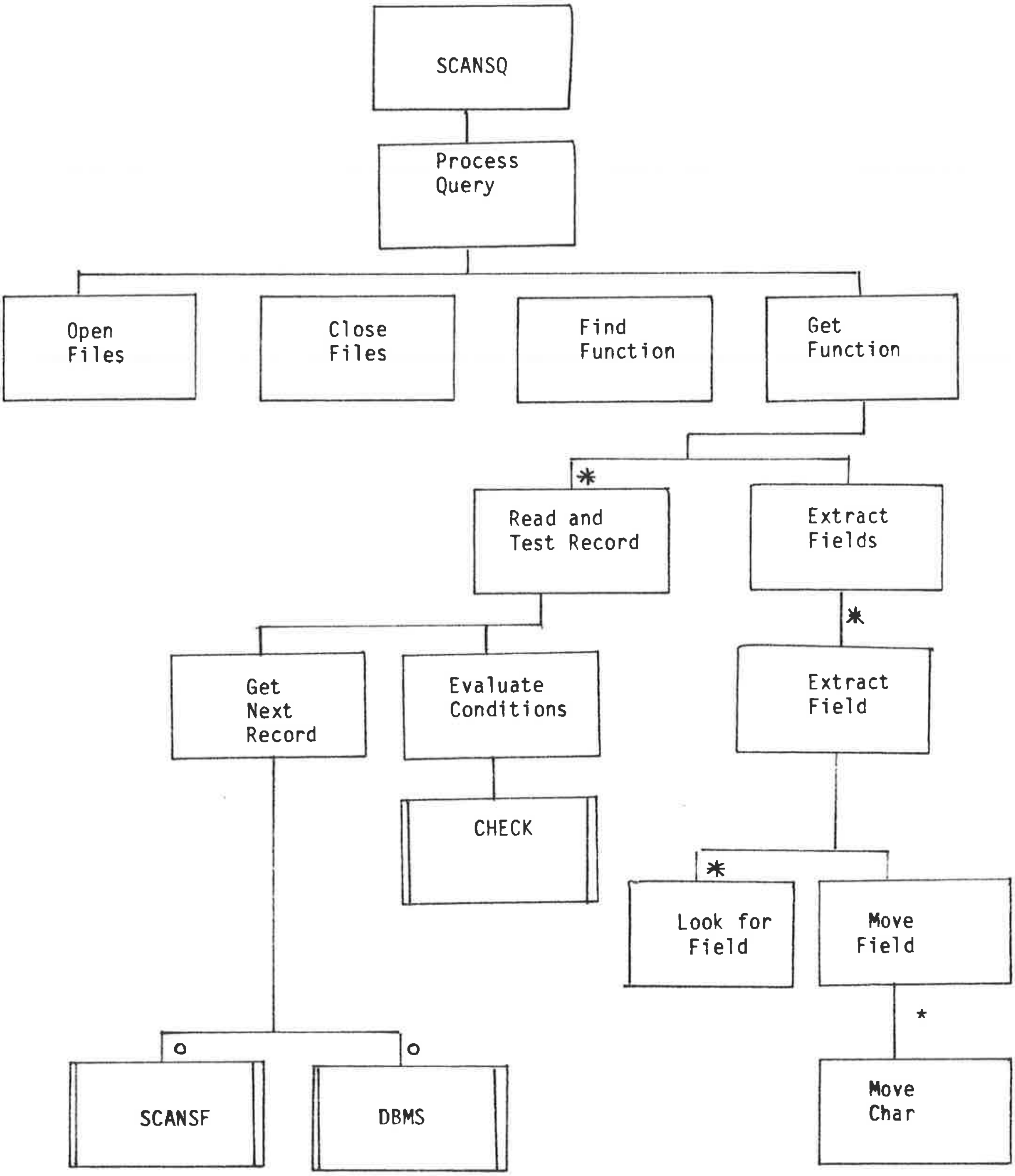

Action
Statement
Get and
Action
Records
Print
Totals
Print
Averages


Get and Action Records
Final Records
*
Get and Action Record
Get Record
Action Record
Arithmetic on Record
Sum Record
Display Record
Print Record

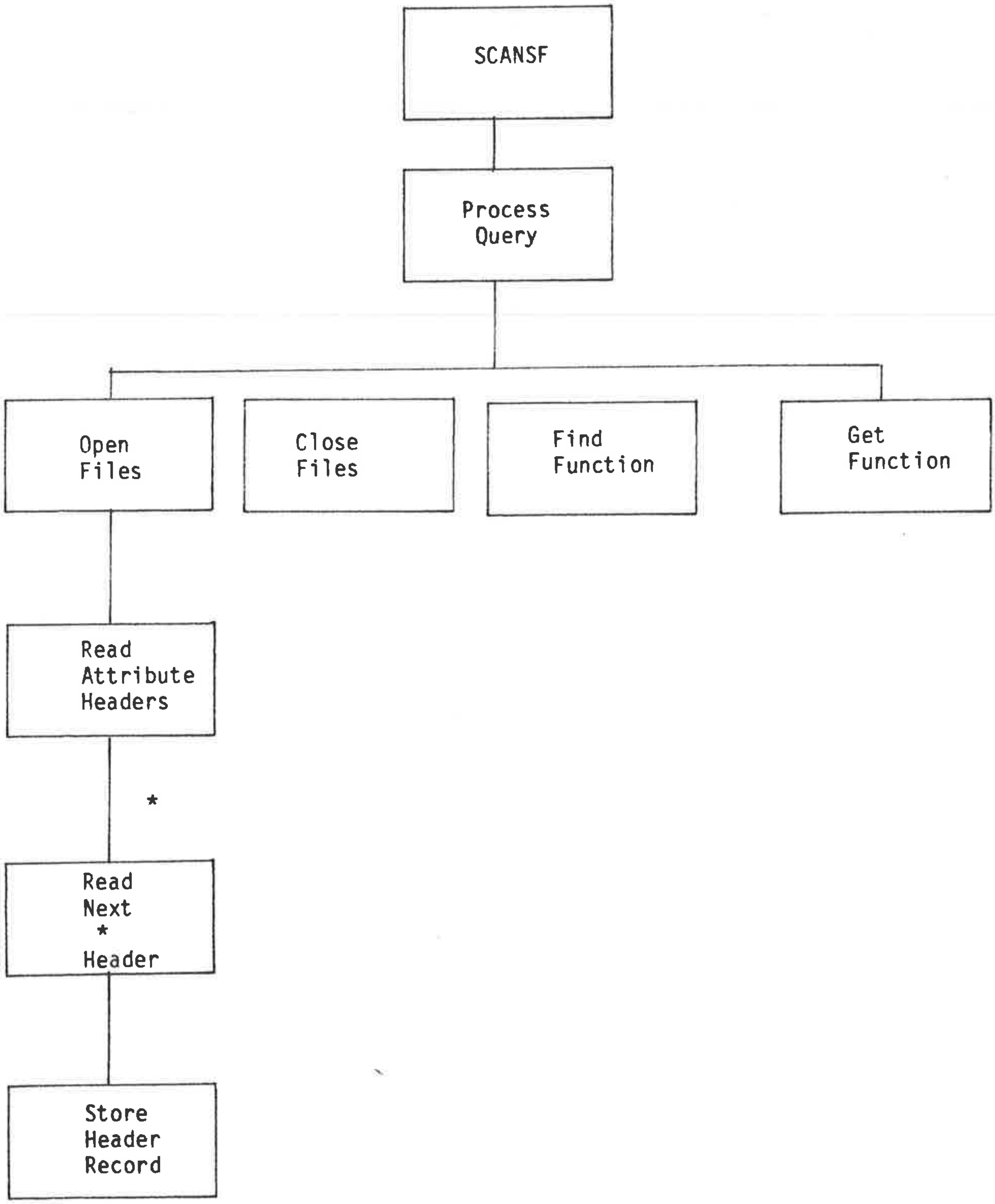
A2.2 Build Sequential File Dictionay (SBUILD)



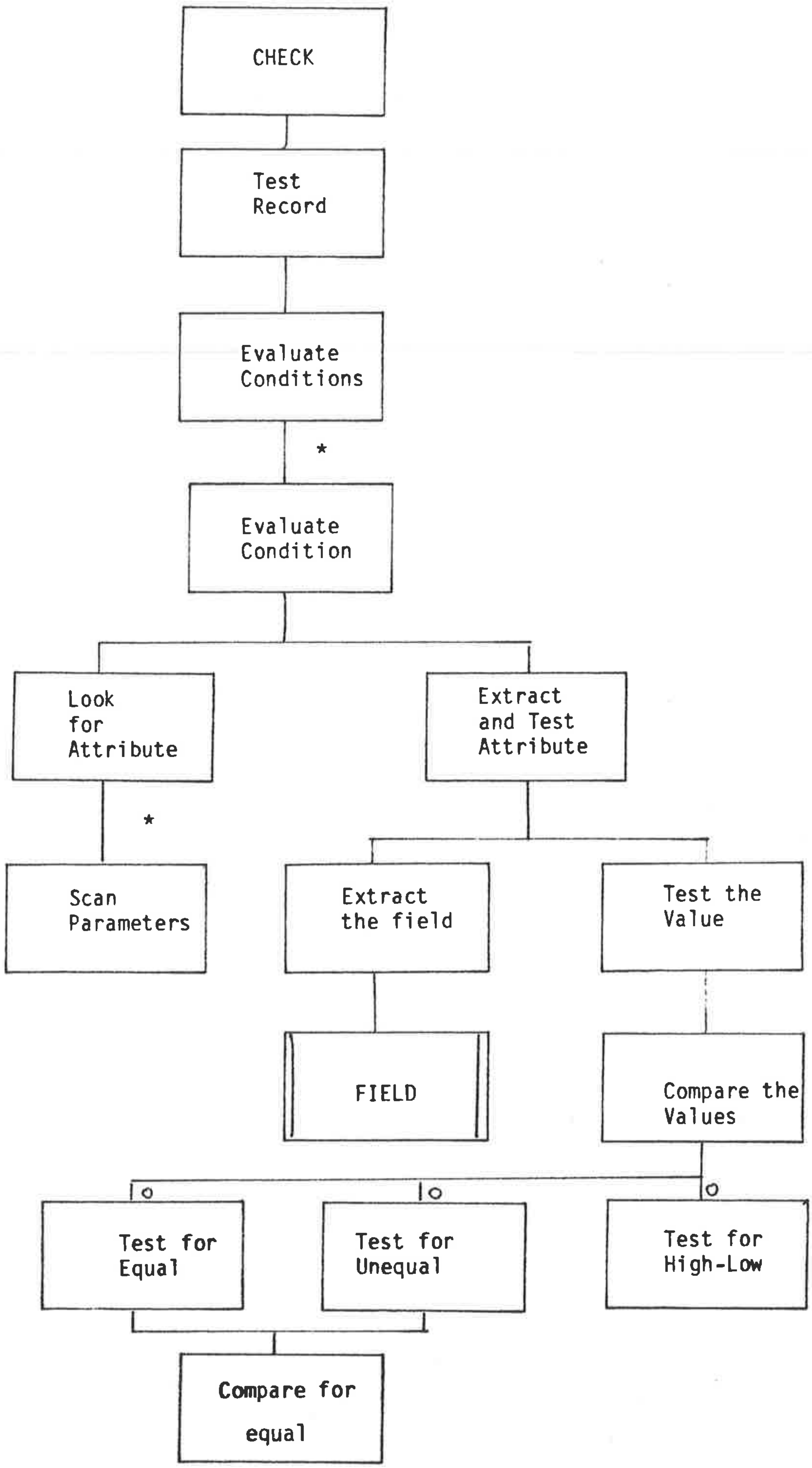
A2.3 Scan Sequential (SCANSQ)



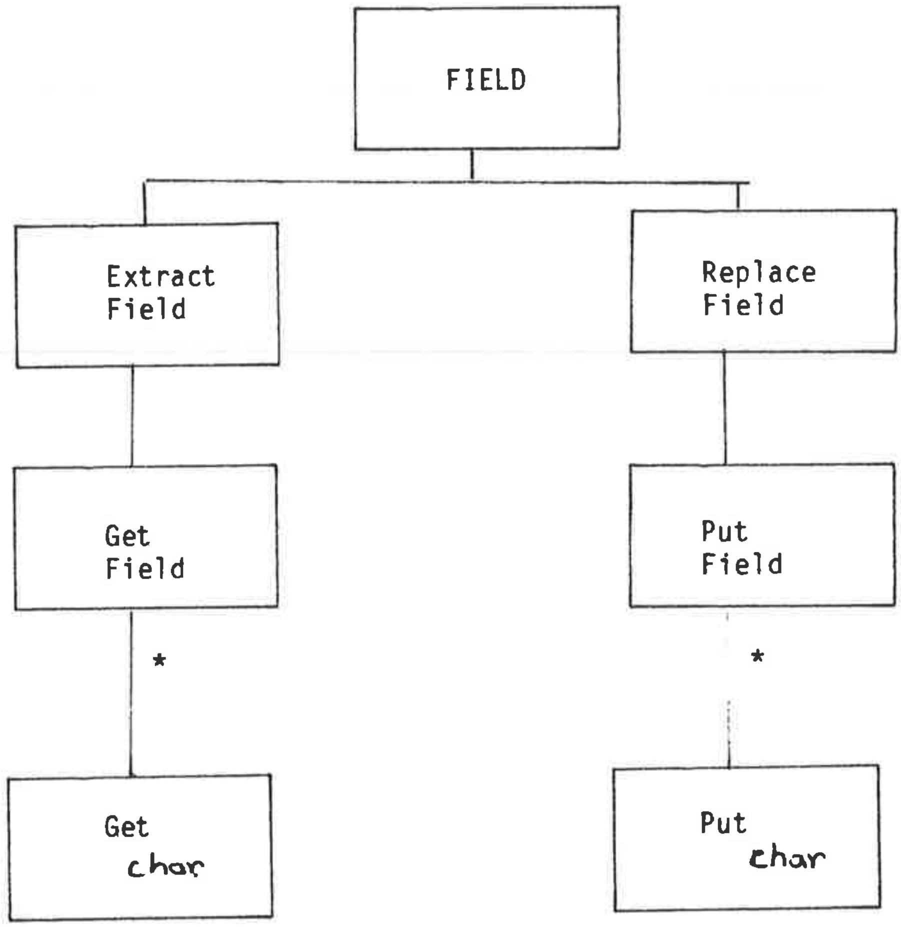
A2.4 Scan Sequential File (SCANSF)



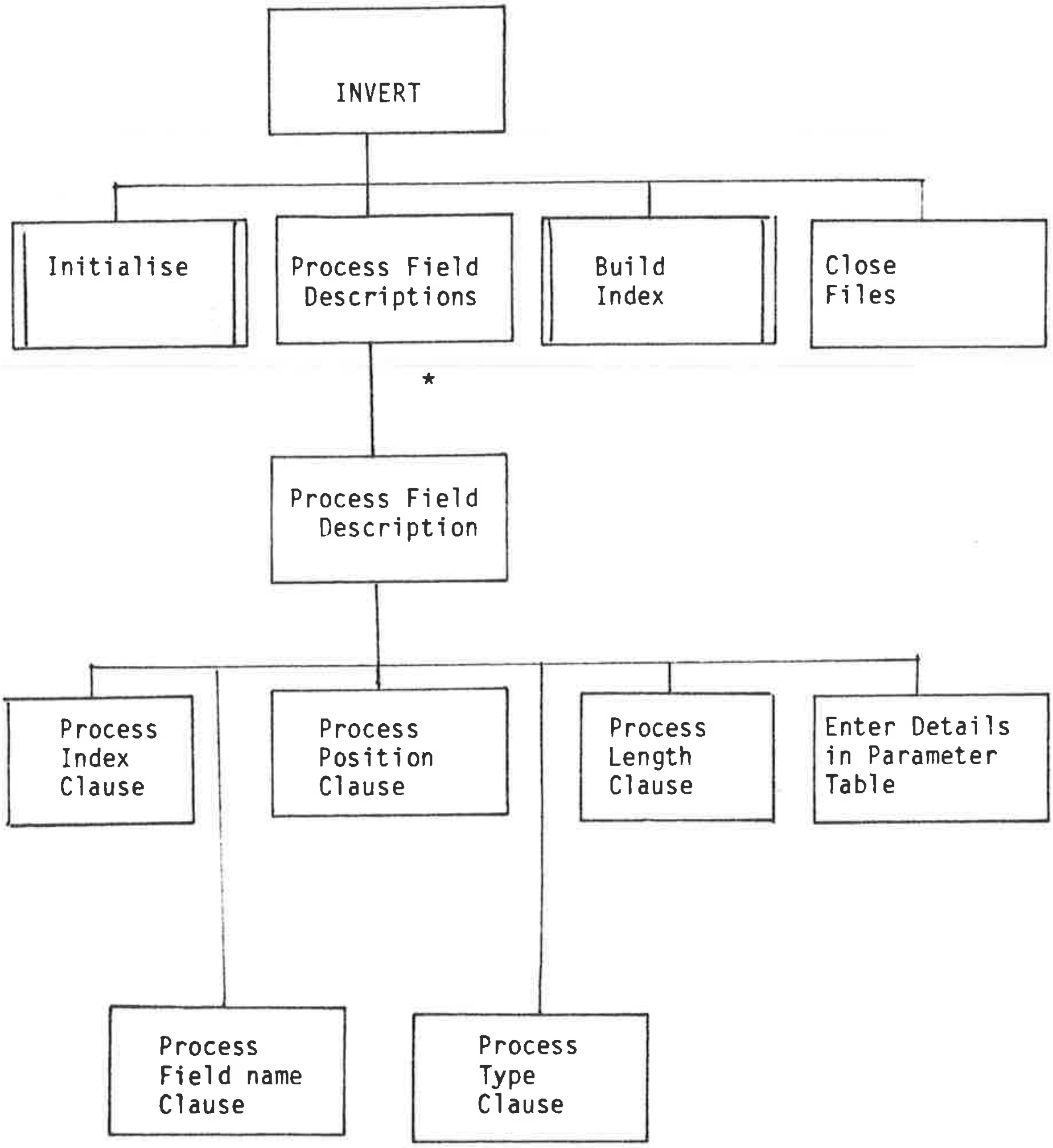
A2.5 Check Conditions (CHECK)

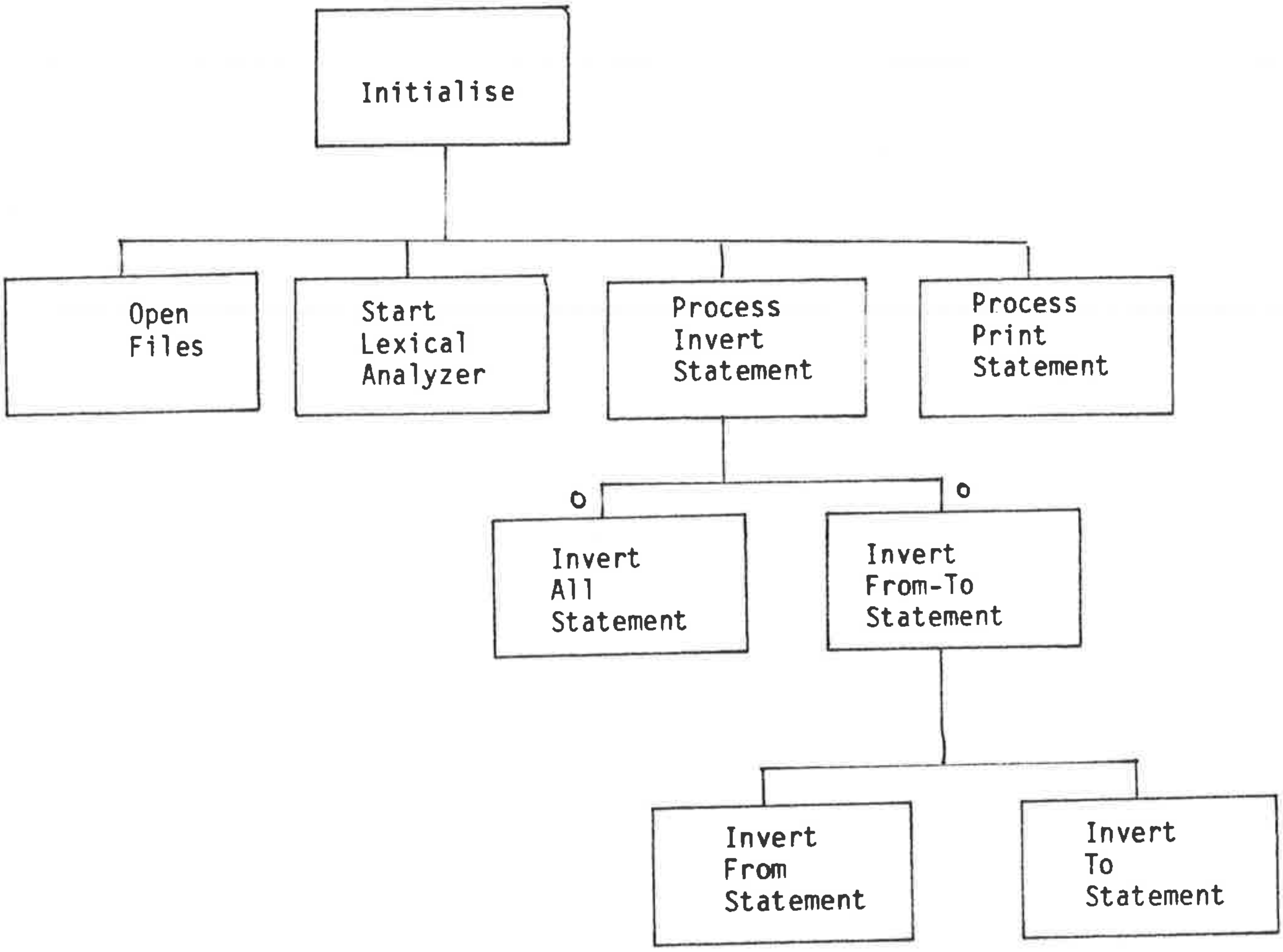


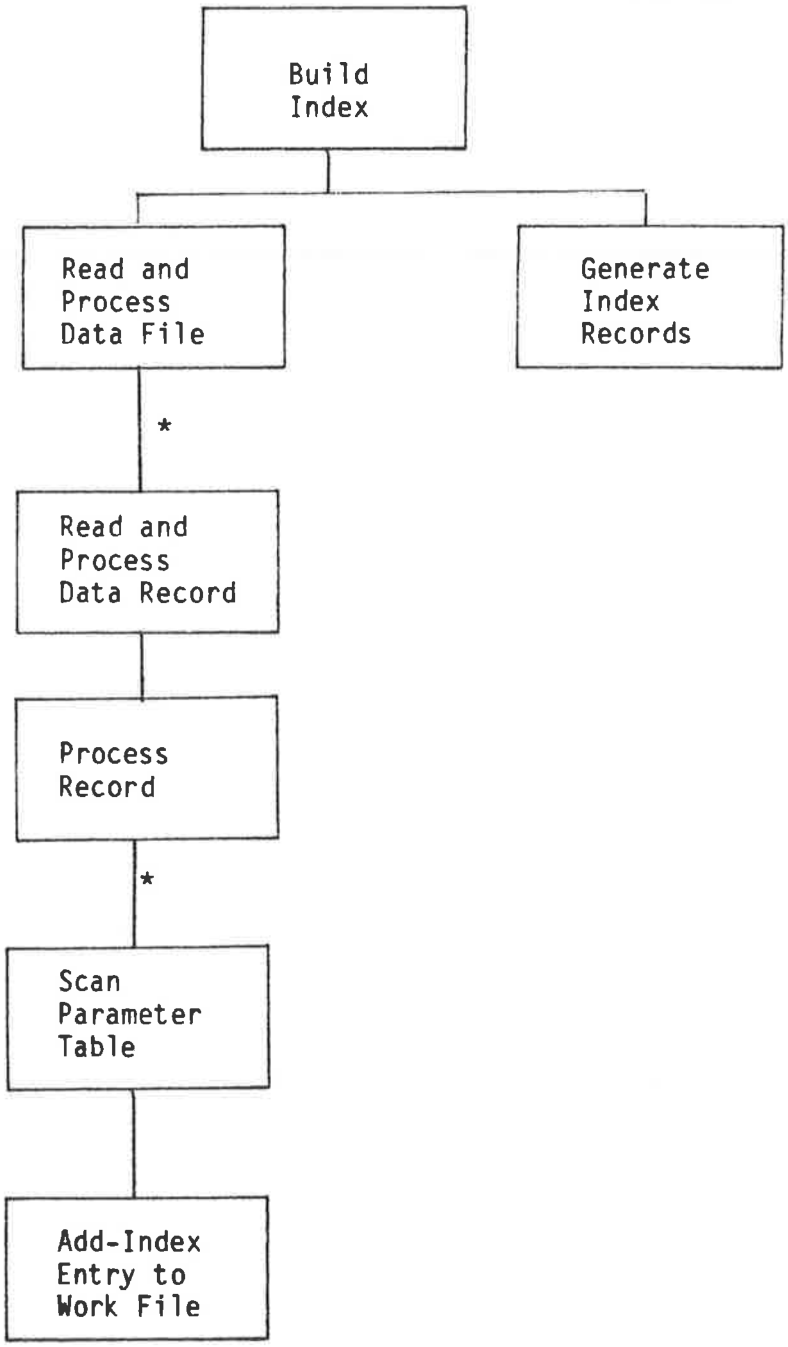
A2.6 Extract/Replace Field (FIELD)

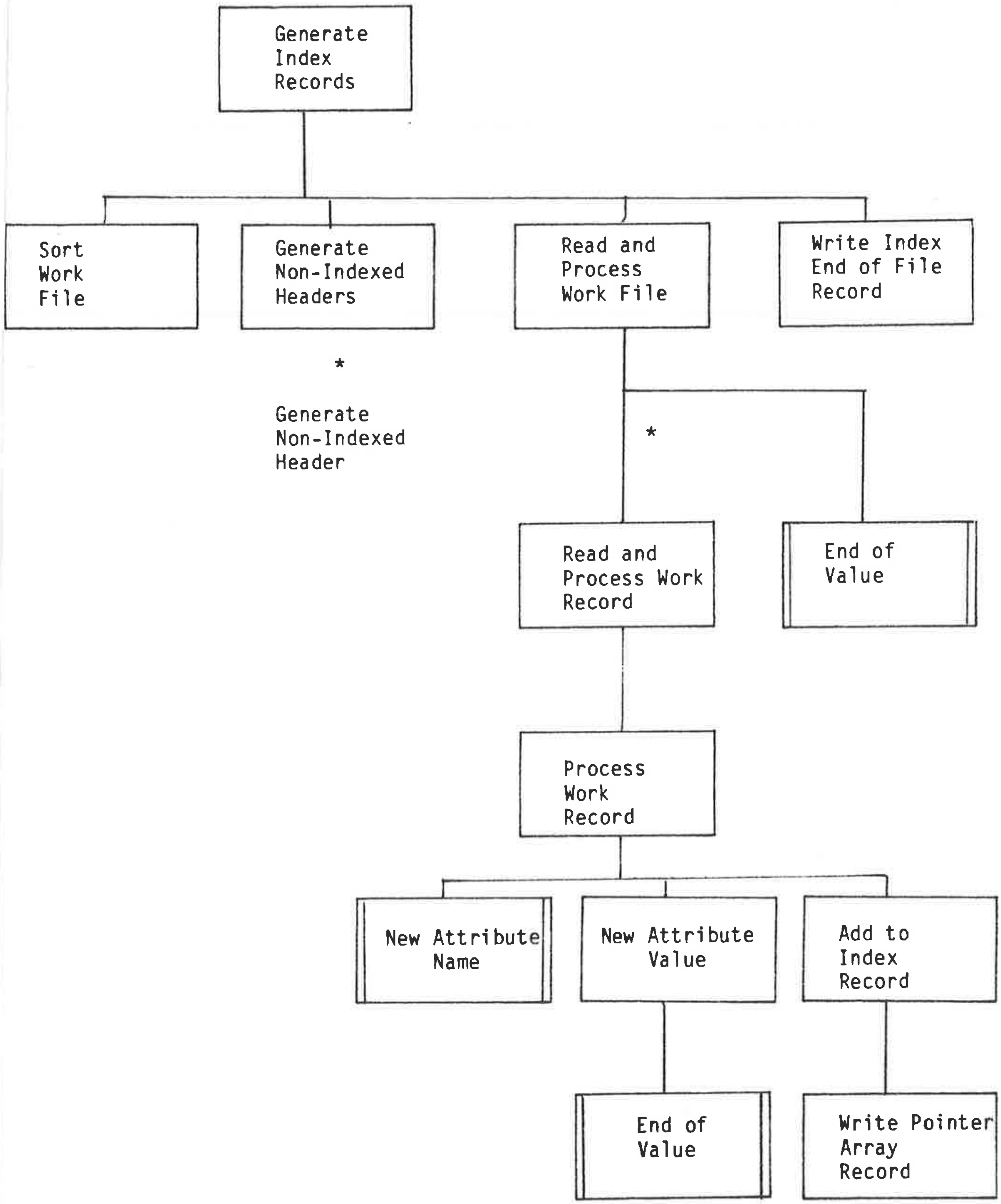


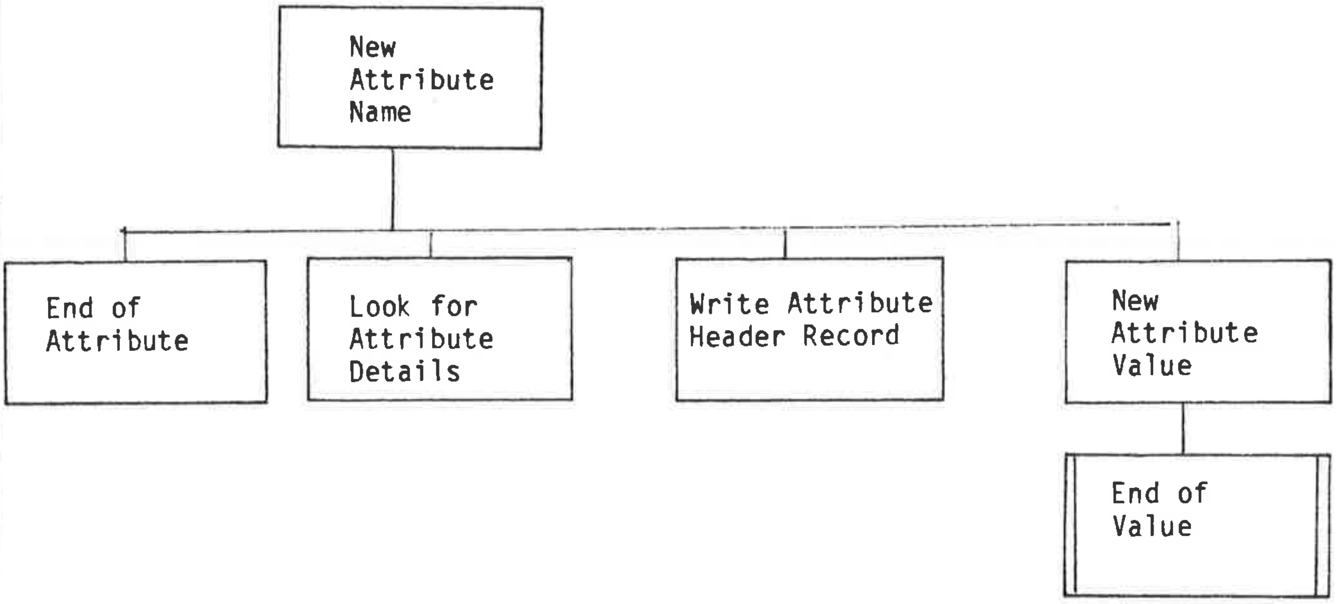
A2.7 Invert File (INVERT)

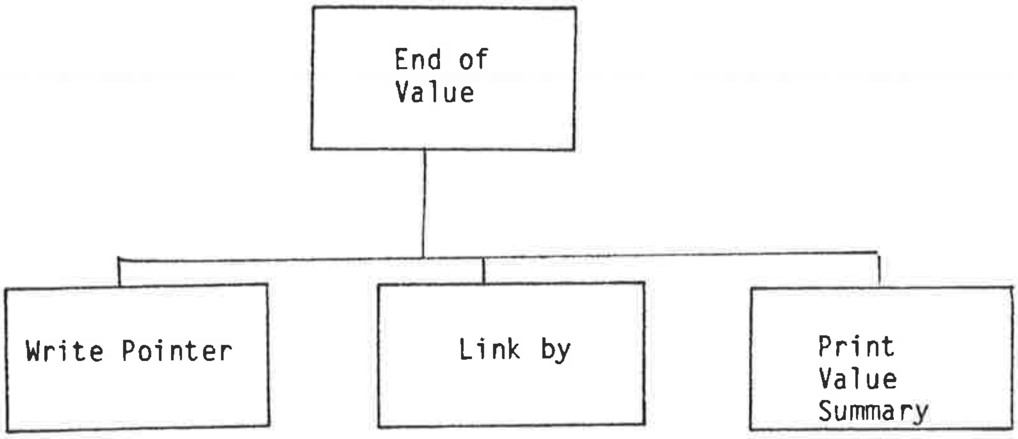



Initialise
Open Files
Start Lexical Analyzer
Process Invert Statement
Process Print Statement
o
o
Invert All Statement
Invert From-To Statement
Invert From Statement
Invert To Statement

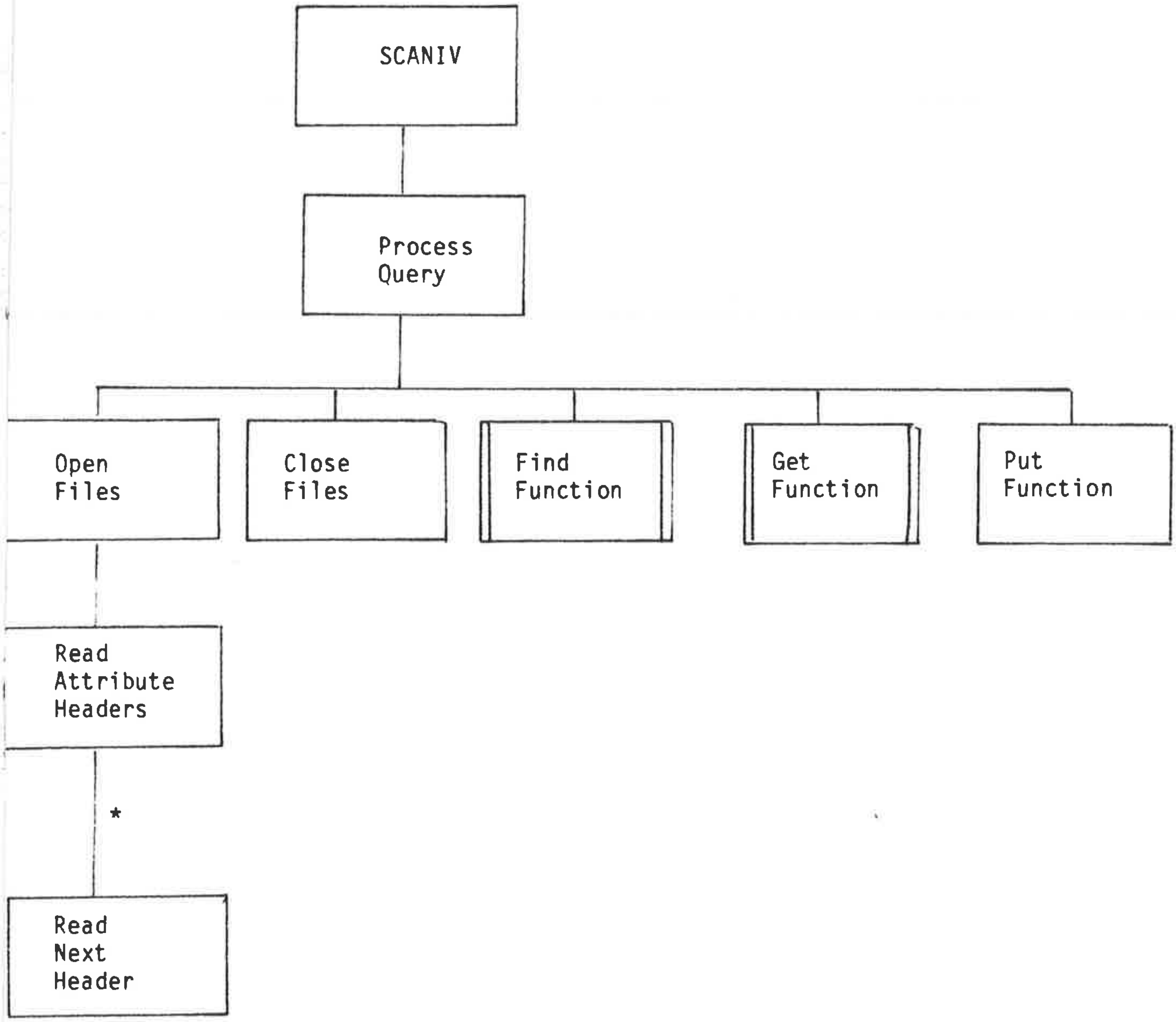

Build Index
Read and Process Data File
Generate Index Records
*
Read and Process Data Record
Process Record
*
Scan Parameter Table
Add-Index Entry to Work File

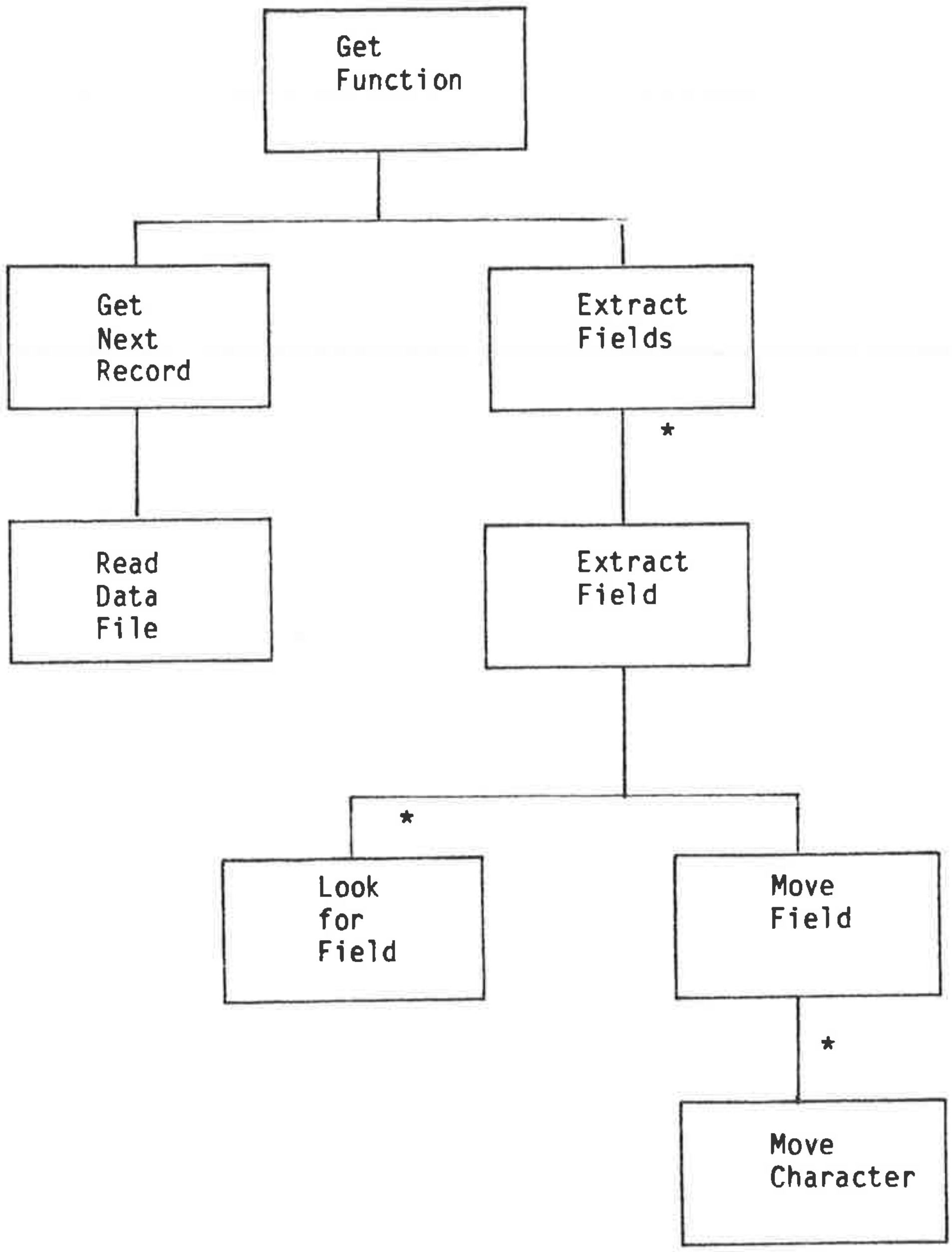

Generate Index Records
Sort Work File
Generate Non-Indexed Headers
Read and Process Work File
Write Index End of File Record
*
Generate Non-Indexed Header
*
Read and Process Work Record
End of Value
Process Work Record
New Attribute Name
New Attribute Value
Add to Index Record
End of Value
Write Pointer Array Record

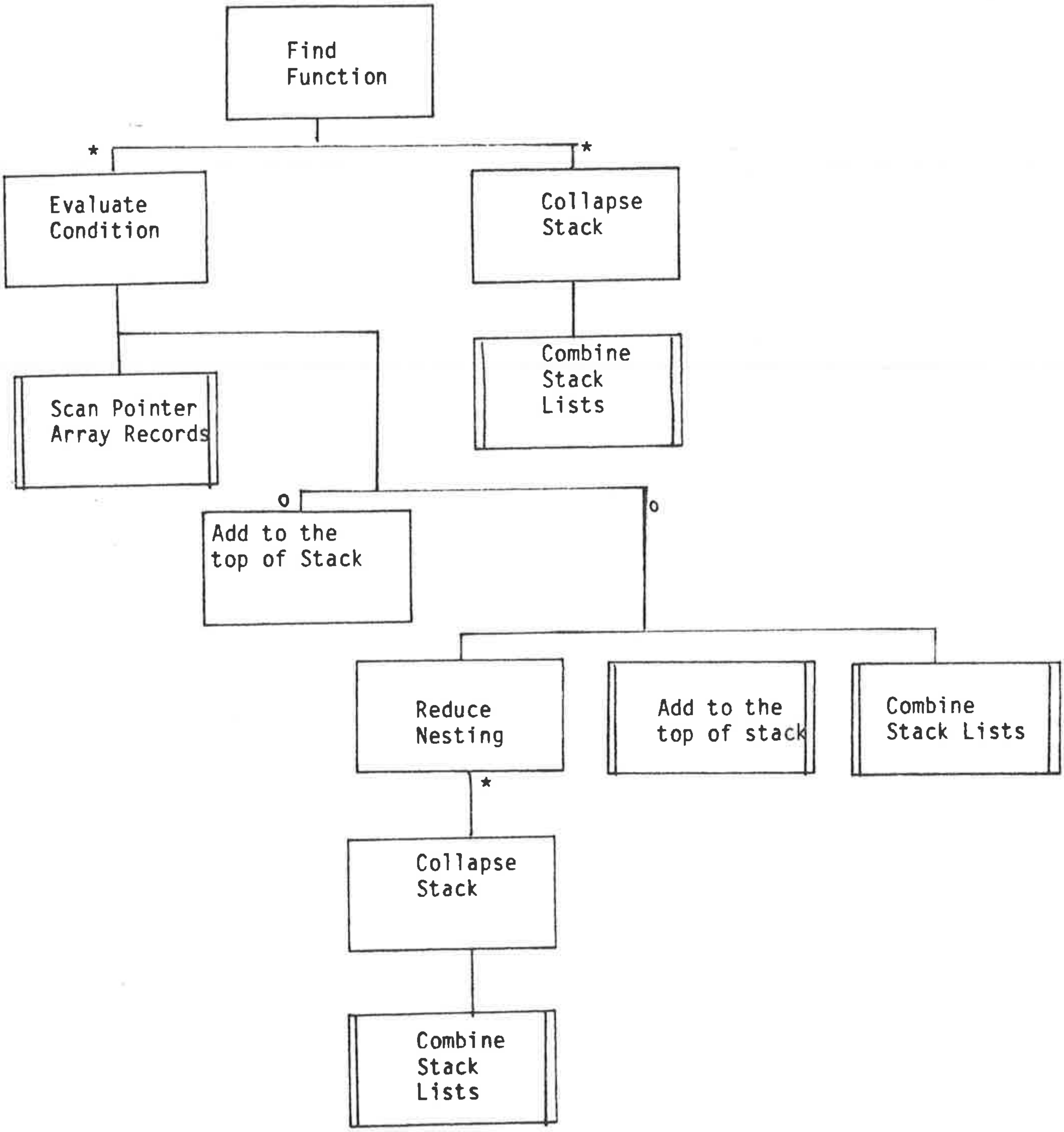

New Attribute Name
End of Attribute
Look for Attribute Details
Write Attribute Header Record
New Attribute Value
End of Value

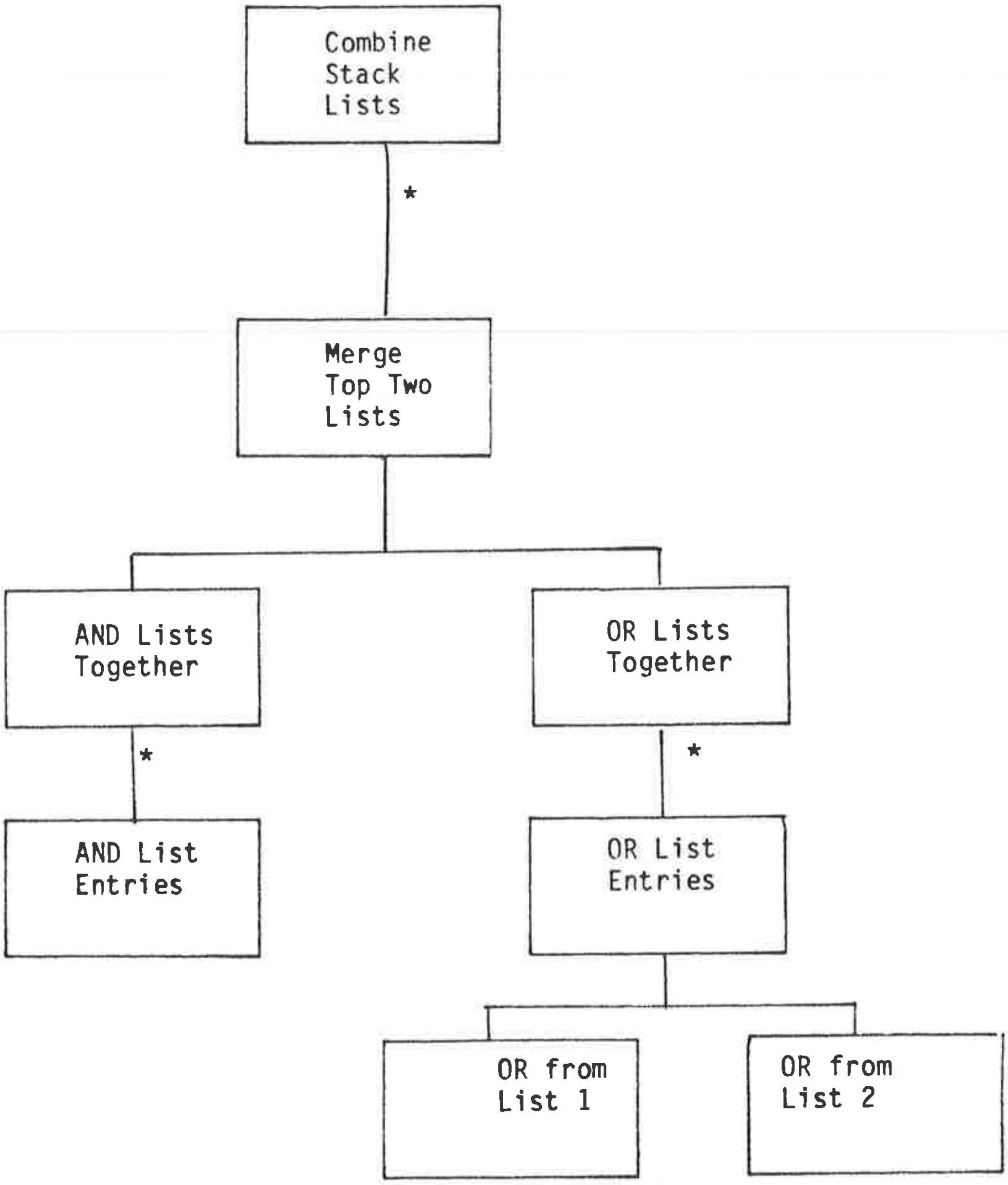

End of
Value
Write Pointer
Link by
Print
Value
Summary

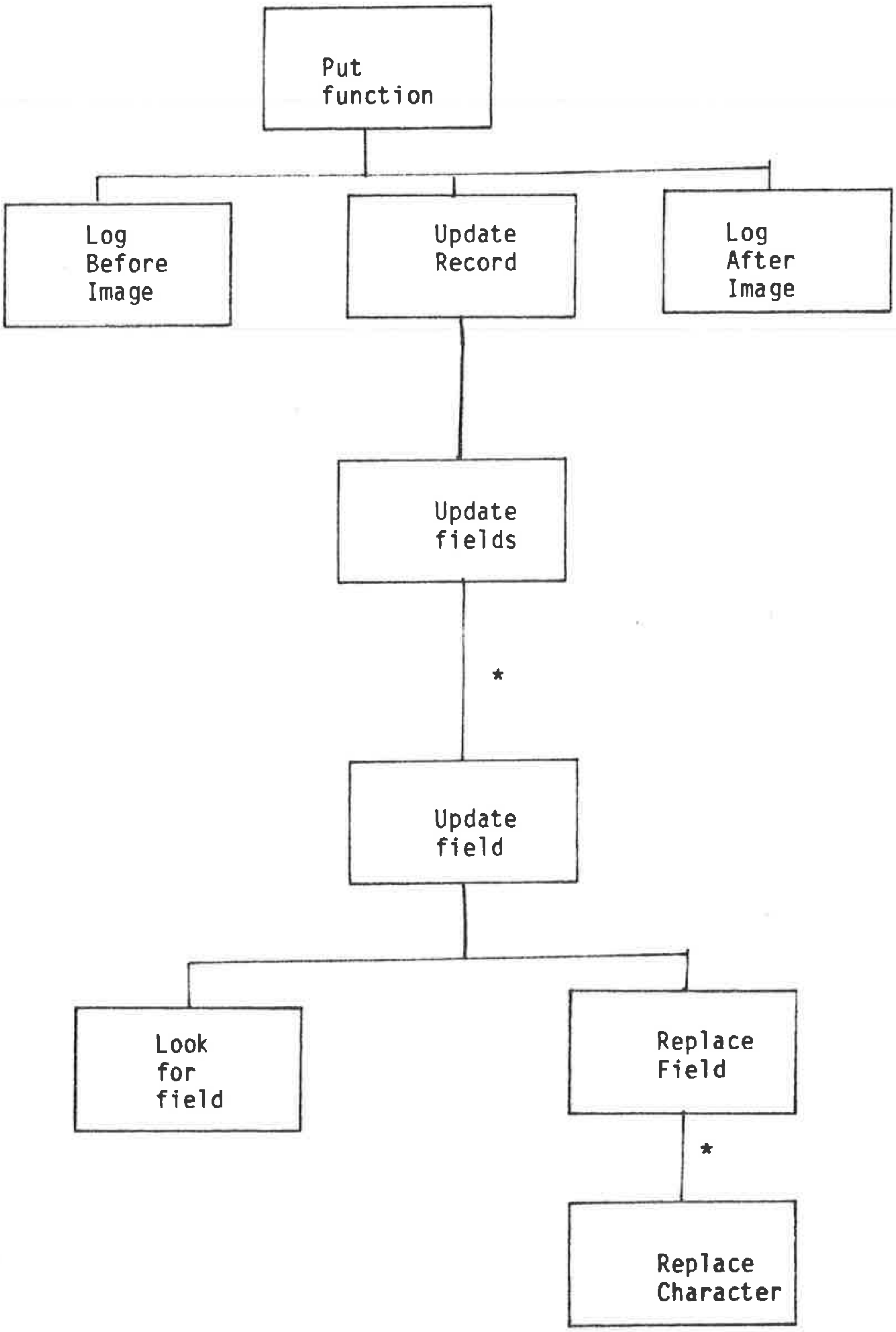
A2.8 Scan Inverted Database (SCANIV)



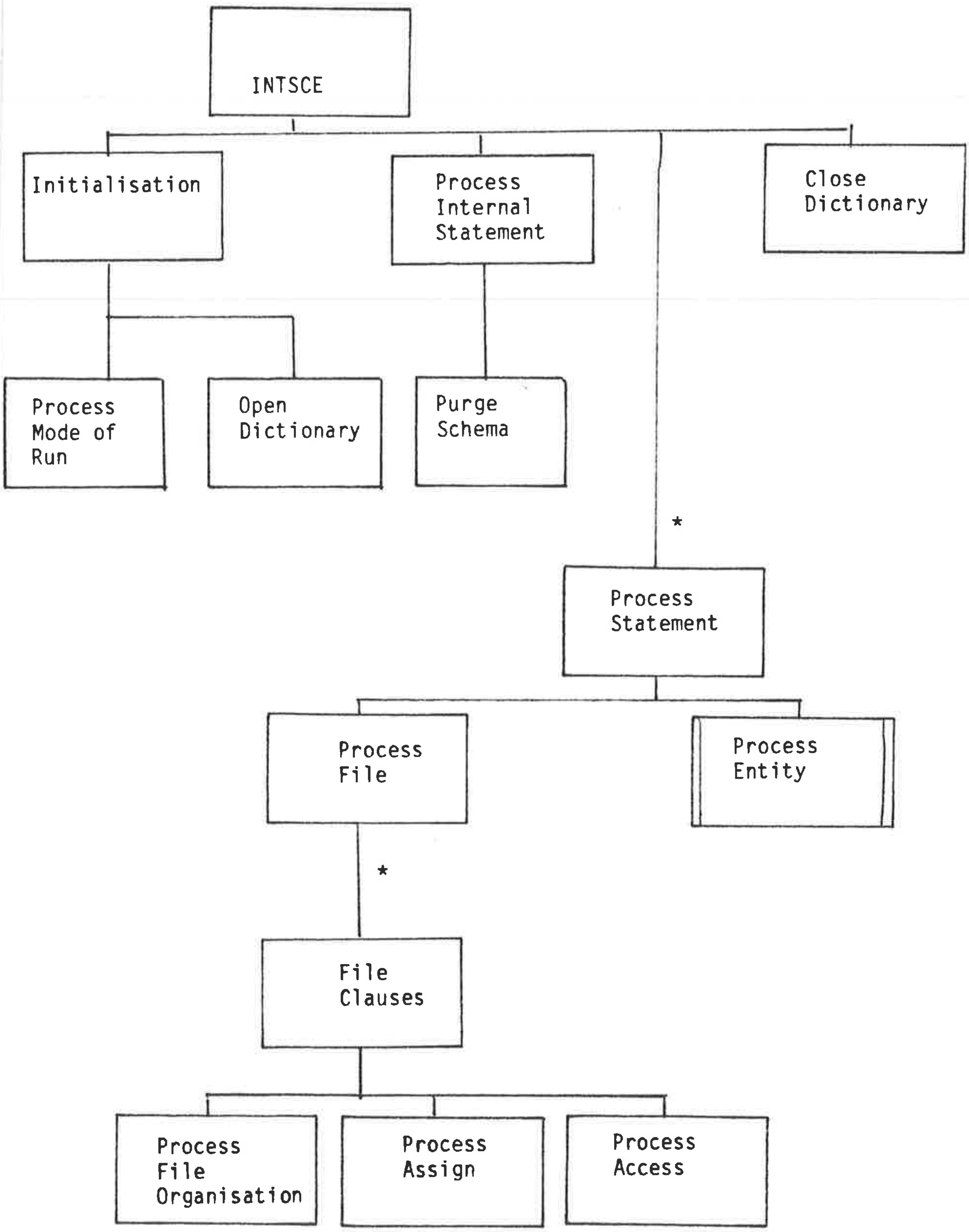

Get Function
Get Next Record
Extract Fields
*
Read Data File
Extract Field
*
Look for Field
Move Field
*
Move Character

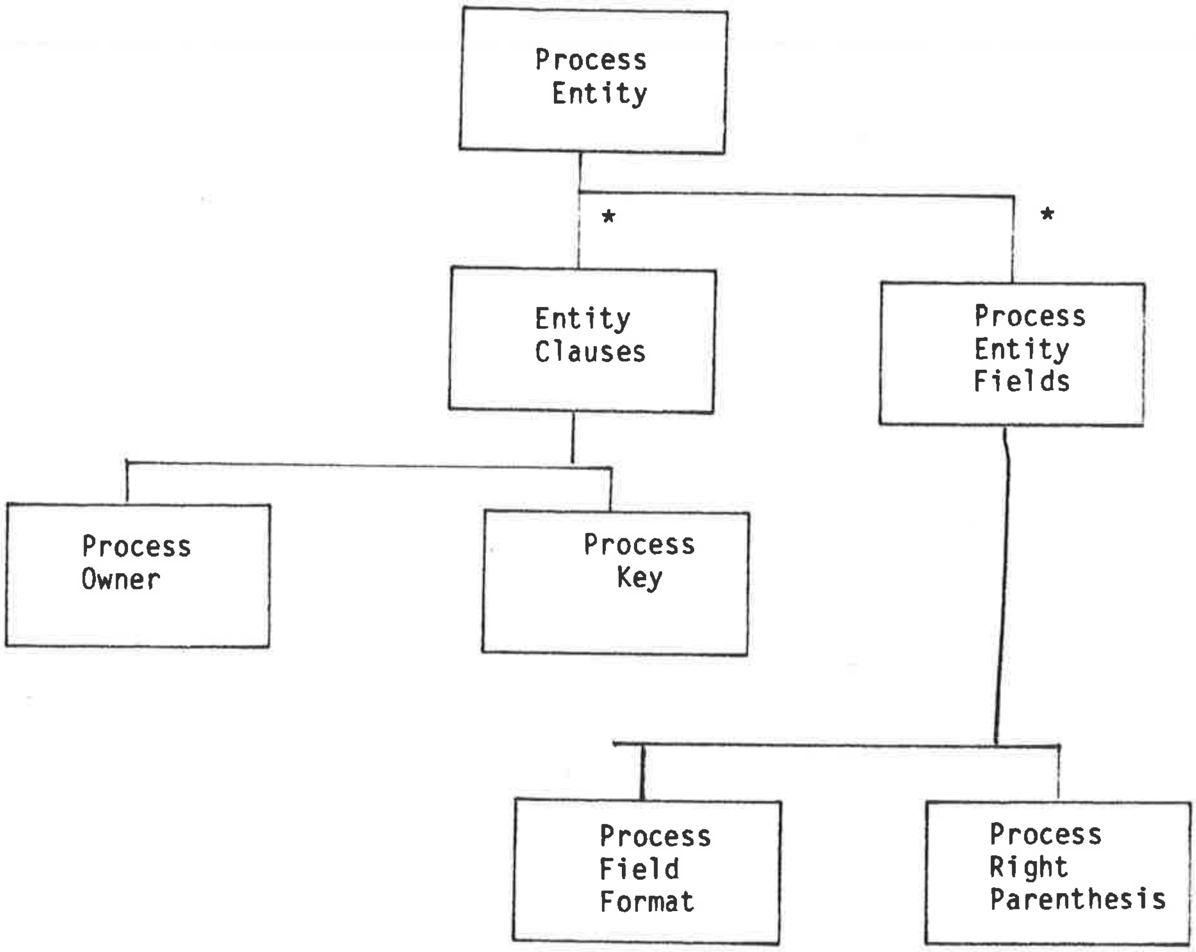

Find Function
*
*
Evaluate Condition
Collapse Stack
Combine Stack Lists
Scan Pointer Array Records
o
o
Add to the top of Stack
Reduce Nesting
Add to the top of stack
Combine Stack Lists
*
Collapse Stack
Combine Stack Lists


Combine
Stack
Lists
*
Merge
Top Two
Lists
AND Lists
Together
*
OR Lists
Together
*
AND List
Entries
OR List
Entries
OR from
List 1
OR from
List 2

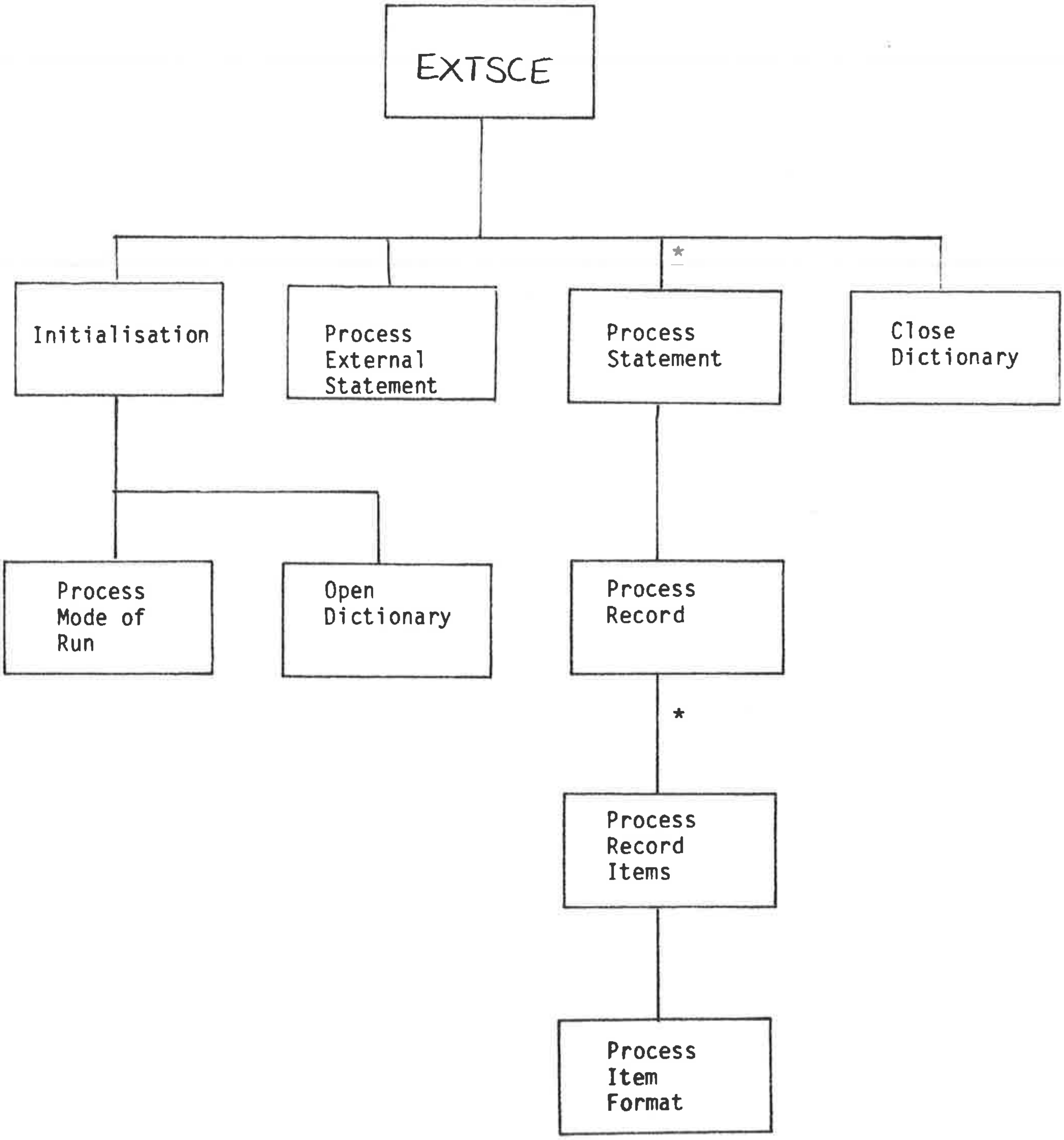

Put
function
Log
Before
Image
Update
Record
Log
After
Image
Update
fields
*
Update
field
Look
for
field
Replace
Field
*
Replace
Character

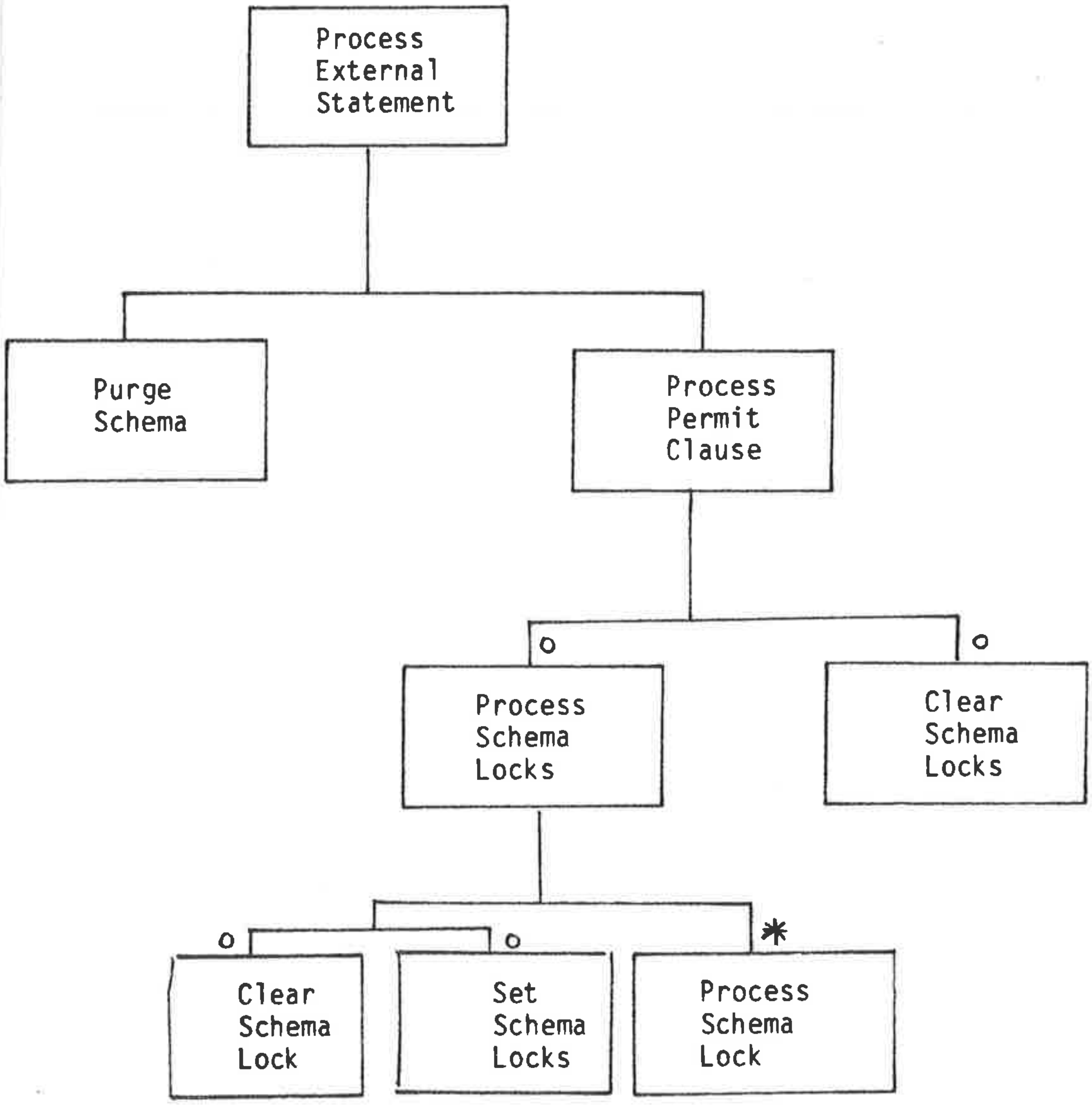
A2.9 Internal Schema DDL Compiler (INTSCE)




Process
Entity
*
*
Entity
Clauses
Process
Entity
Fields
Process
Owner
Process
Key
Process
Field
Format
Process
Right
Parenthesis

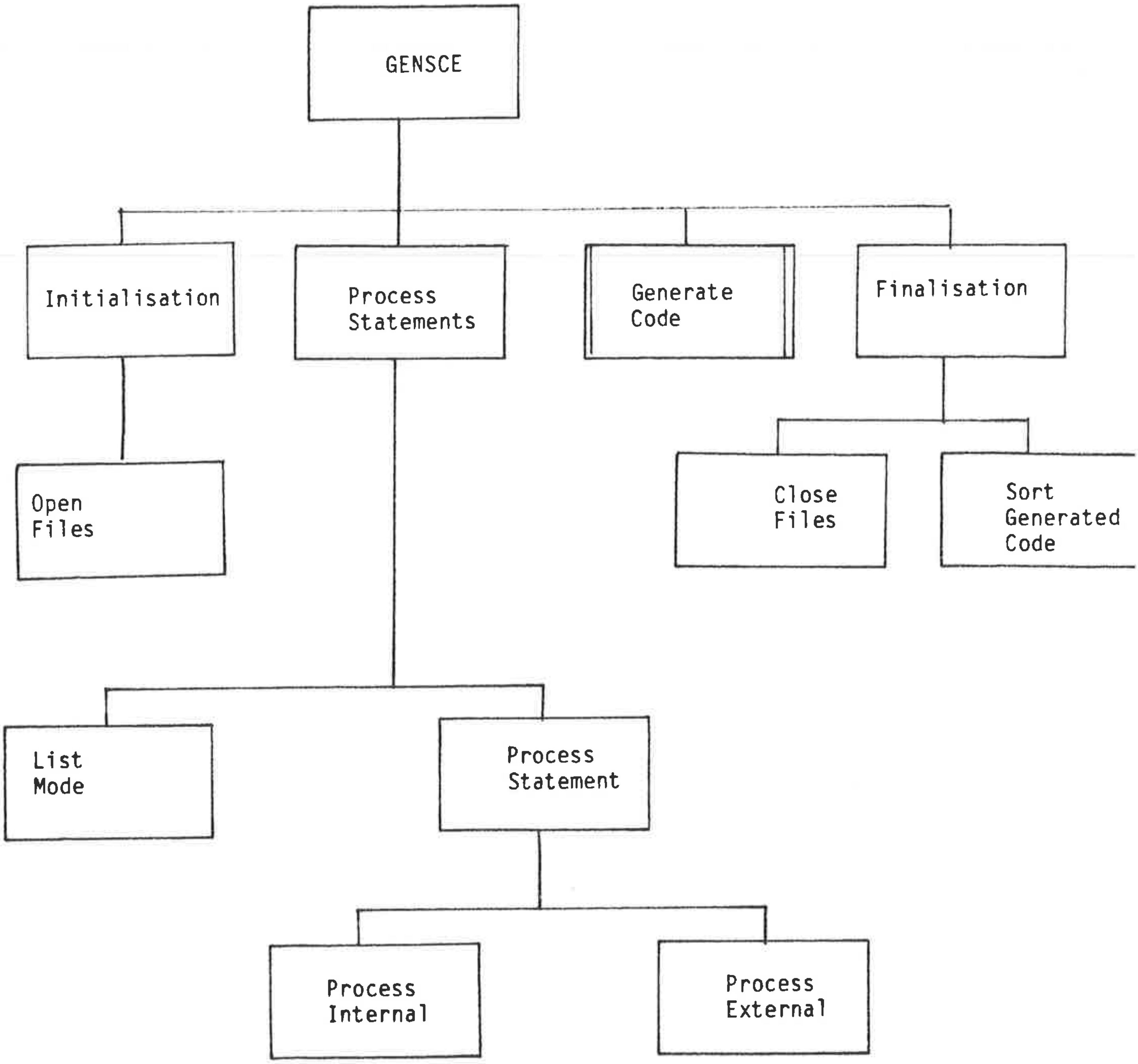
A2.10 External Schema DDL Compiler (EXTSCE)

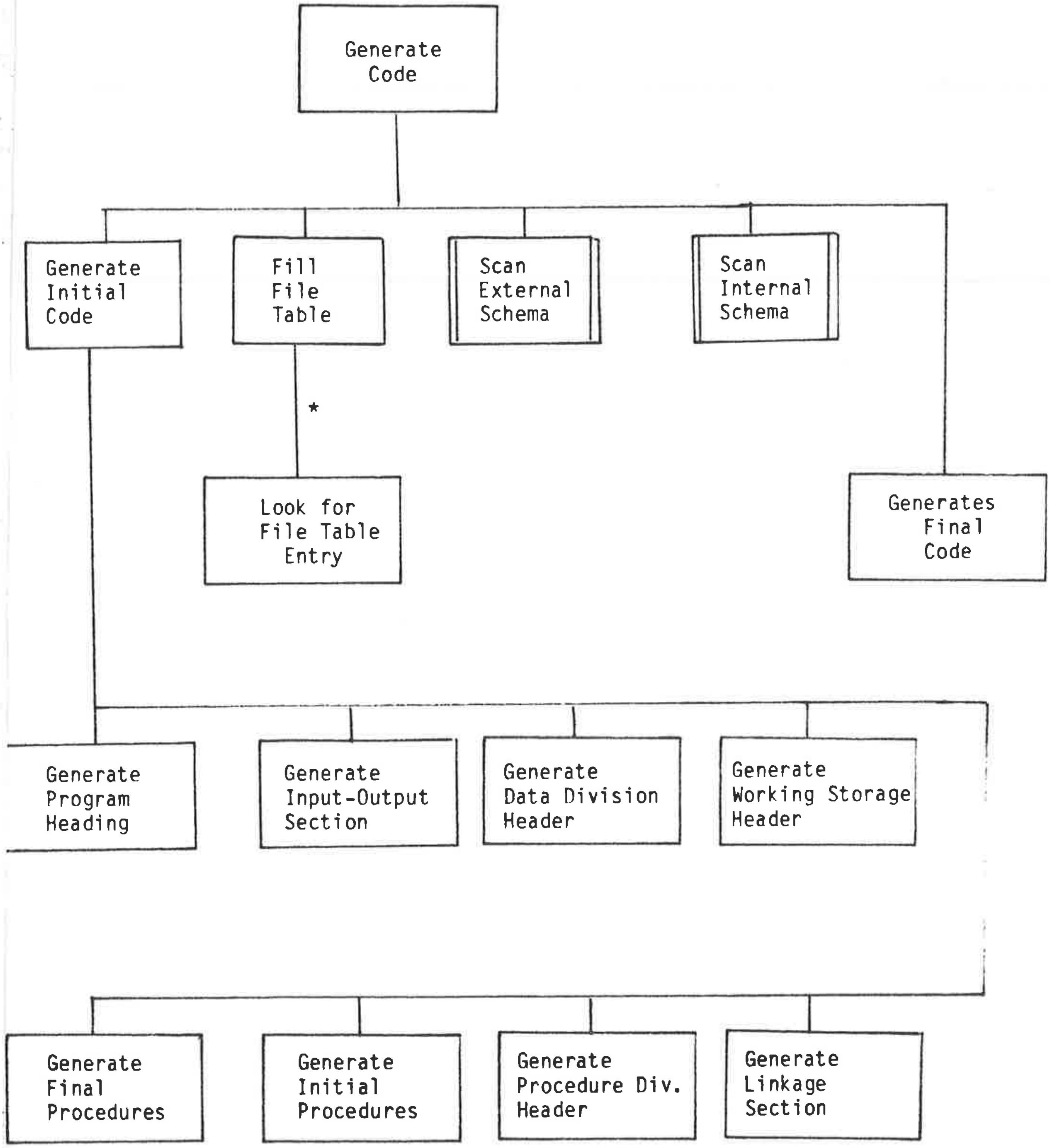


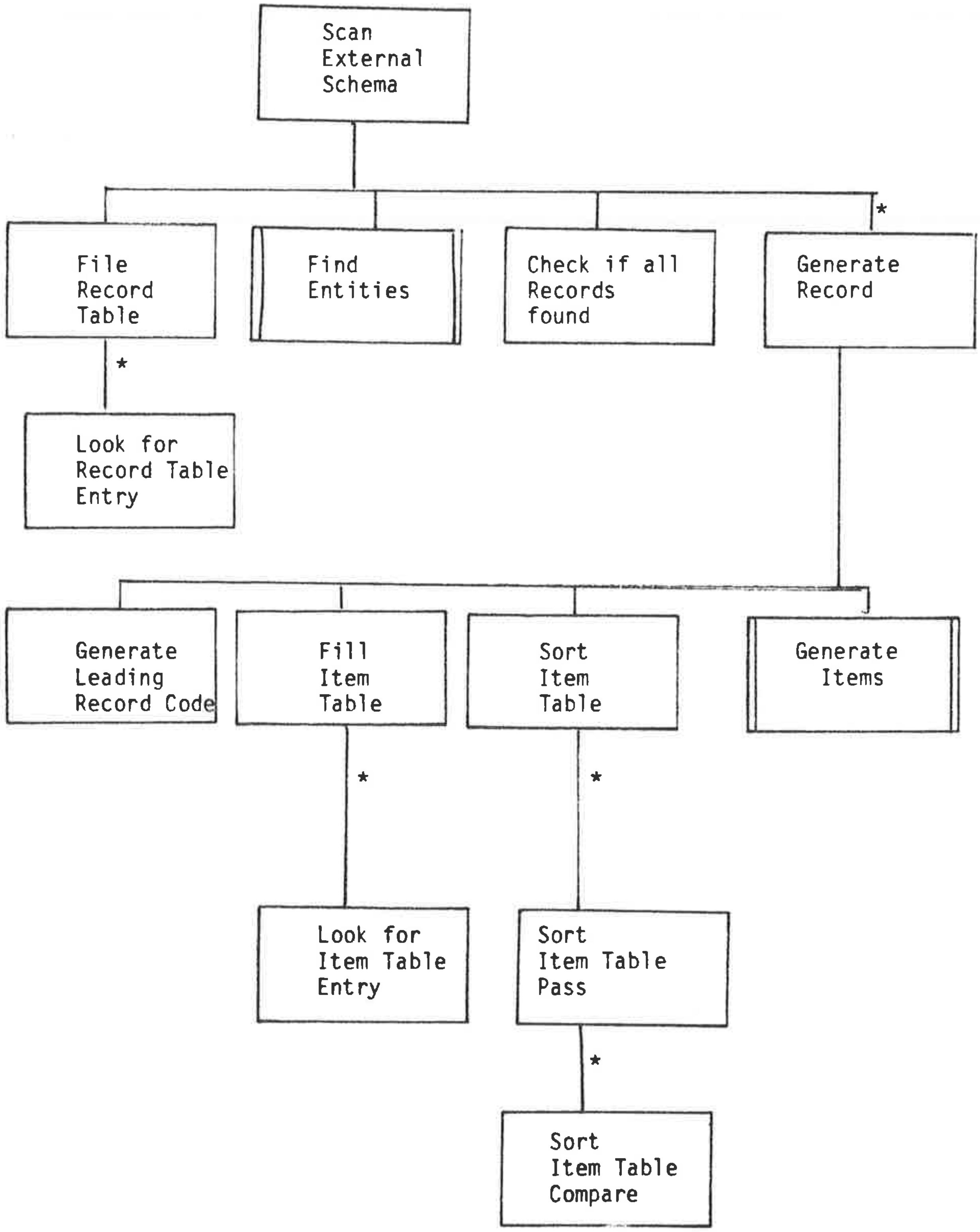


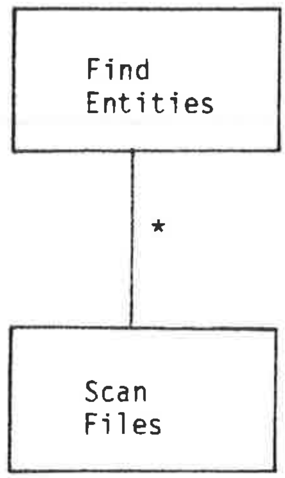
Process External Statement
Purge Schema
Process Permit Clause
o
o
Process Schema Locks
Clear Schema Locks
o
o
*
Clear Schema Lock
Set Schema Locks
Process Schema Lock

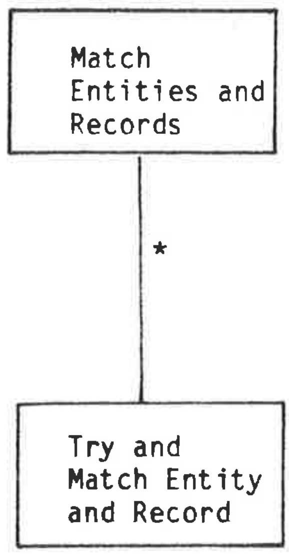
A2.11 Generate Mapping Code (GENSCE)

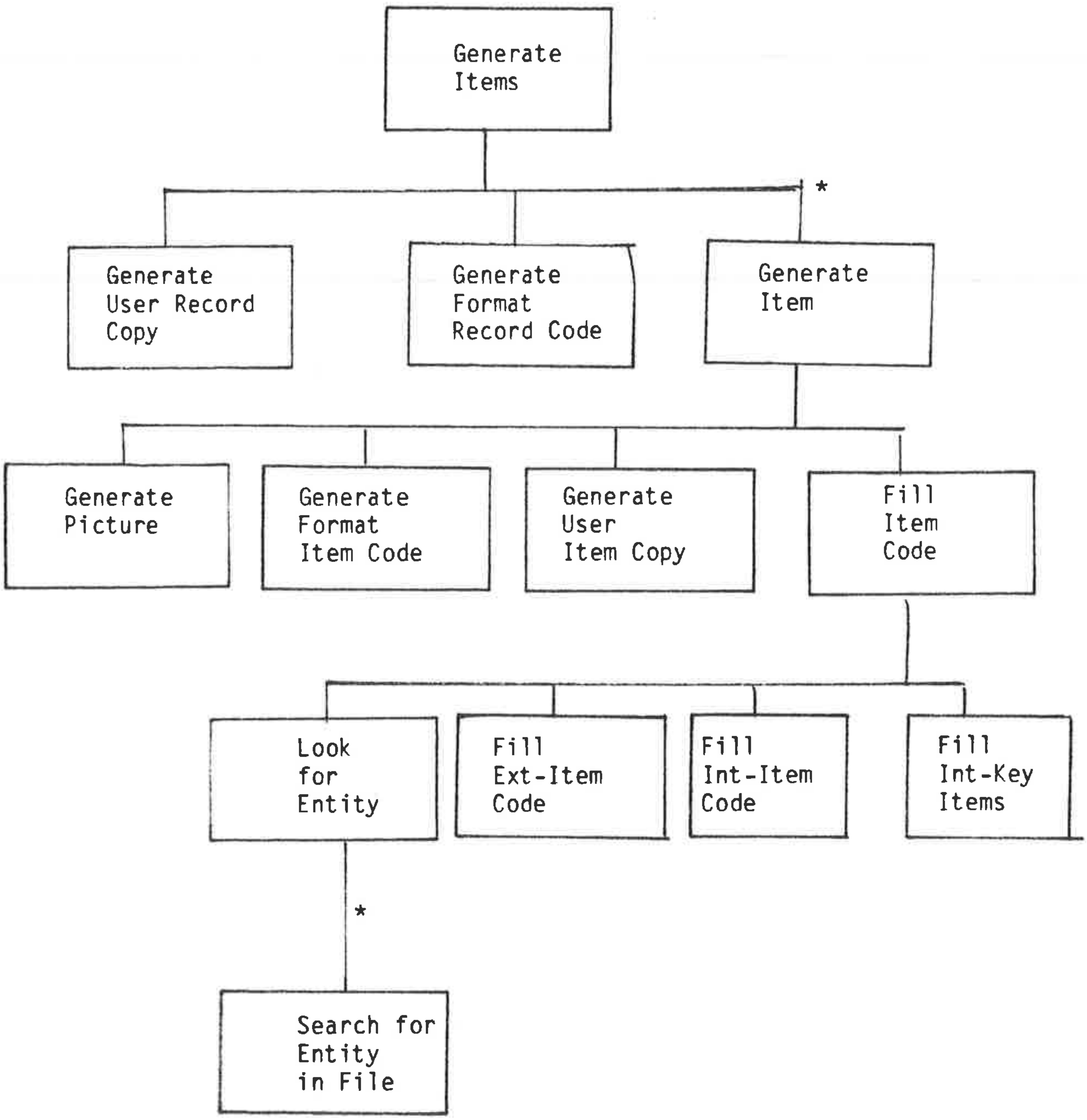


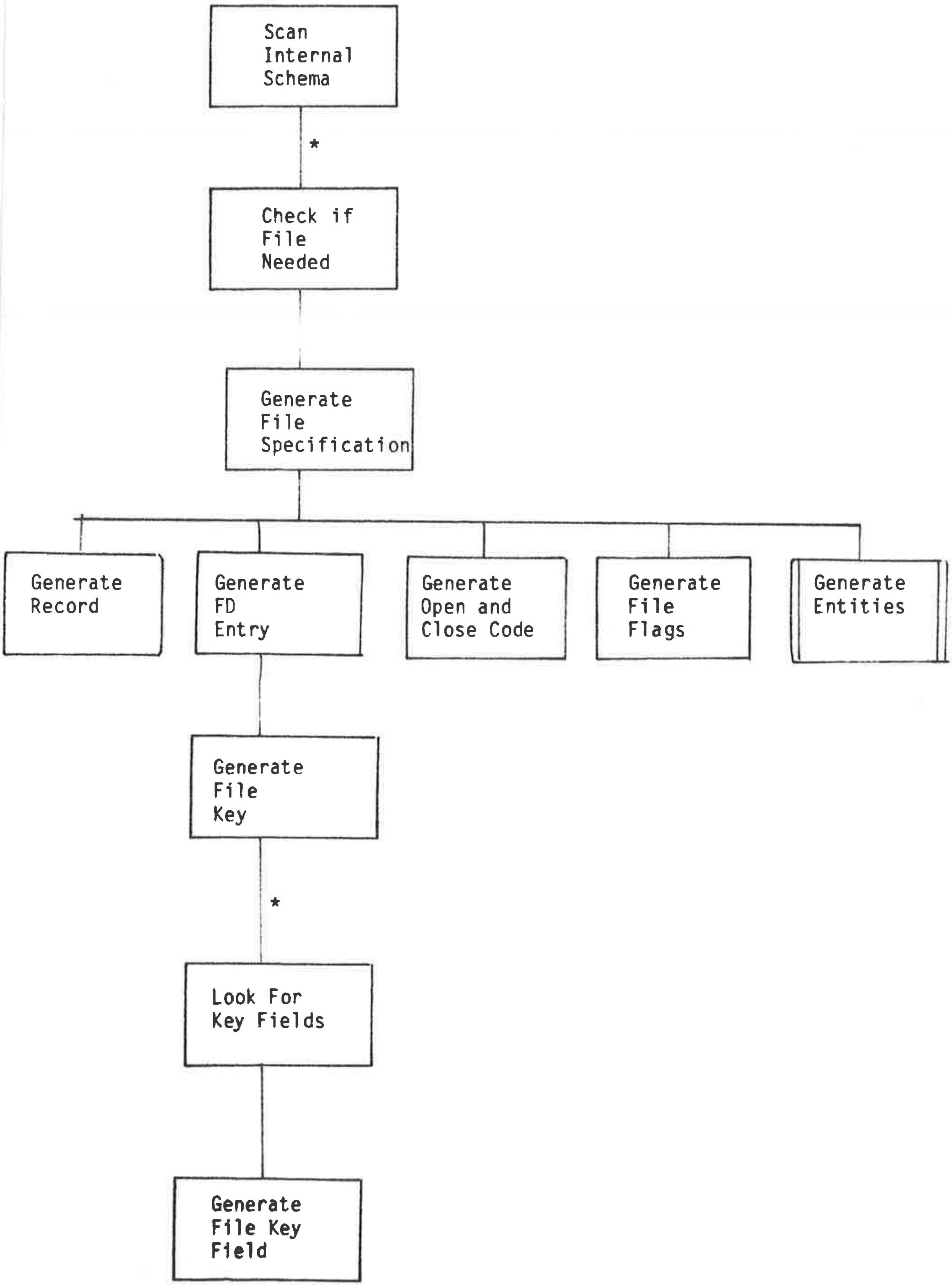

Generate Code
Generate Initial Code
Fill File Table
Scan External Schema
Scan Internal Schema
*
Look for File Table Entry
Generates Final Code
Generate Program Heading
Generate Input-Output Section
Generate Data Division Header
Generate Working Storage Header
Generate Final Procedures
Generate Initial Procedures
Generate Procedure Div. Header
Generate Linkage Section

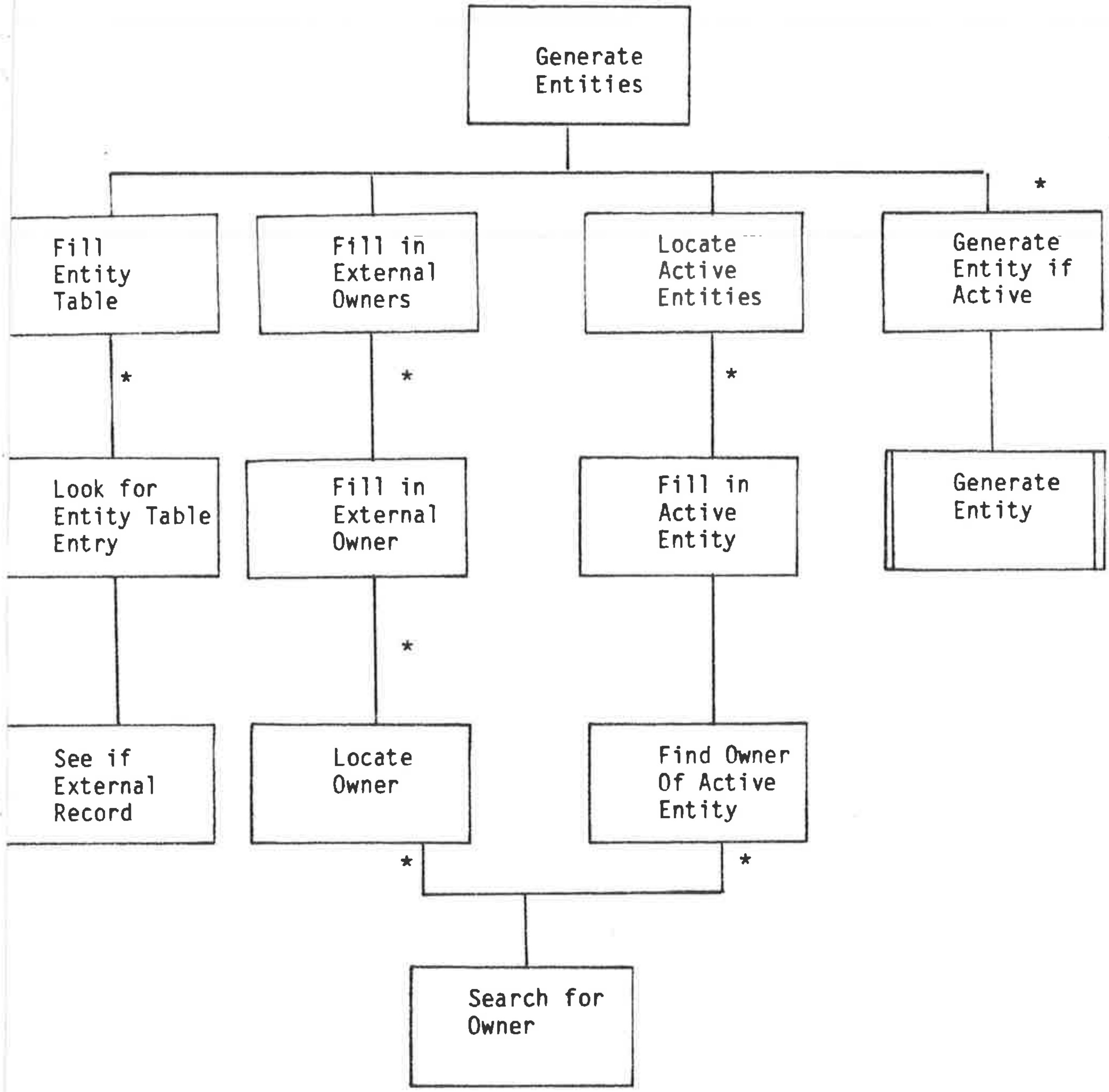

Scan External Schema
File Record Table
Find Entities
Check if all Records found
Generate Record
*
*
Look for Record Table Entry
Generate Leading Record Code
Fill Item Table
Sort Item Table
Generate Items
*
*
Look for Item Table Entry
Sort Item Table Pass
*
Sort Item Table Compare

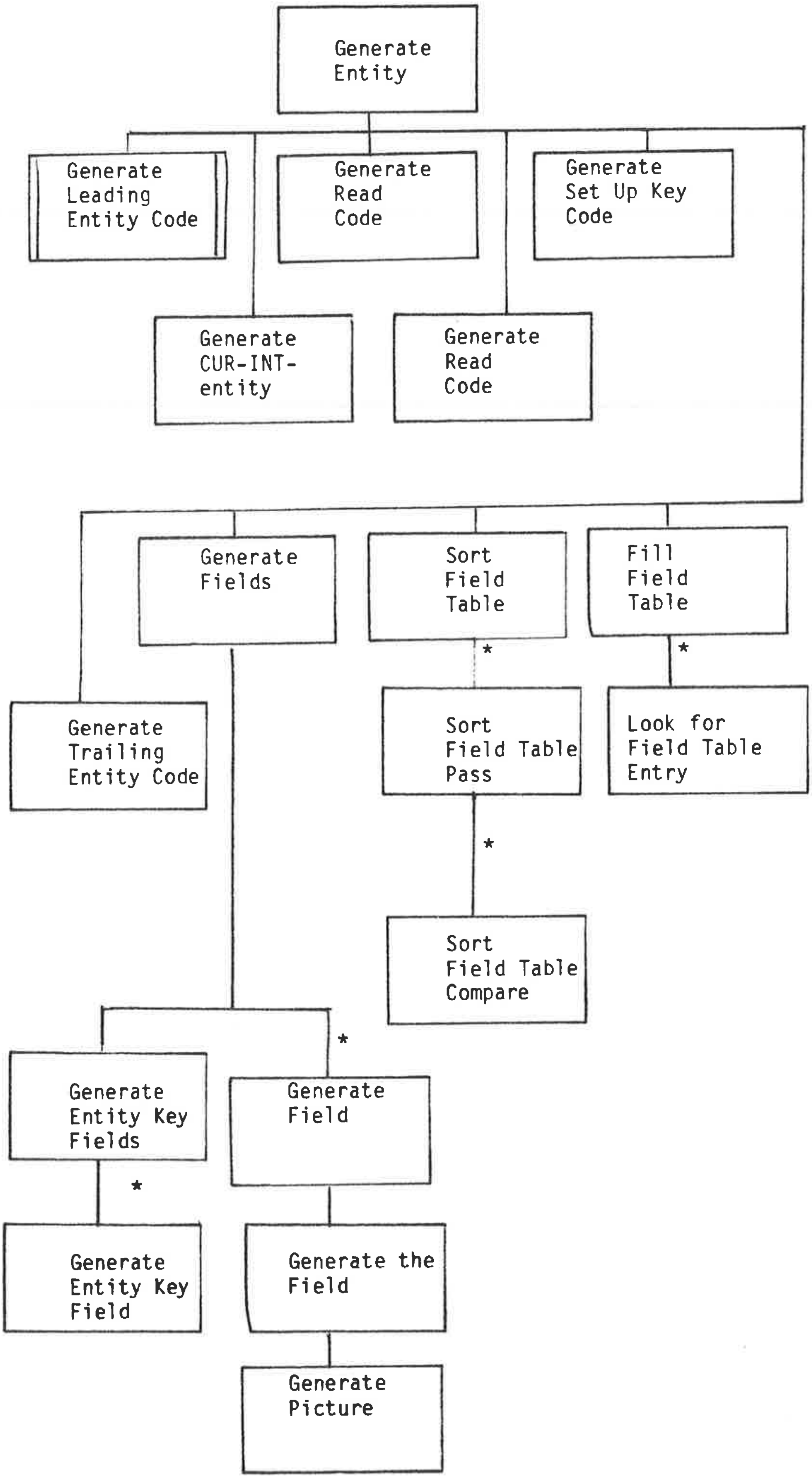

Find
Entities
*
Scan
Files

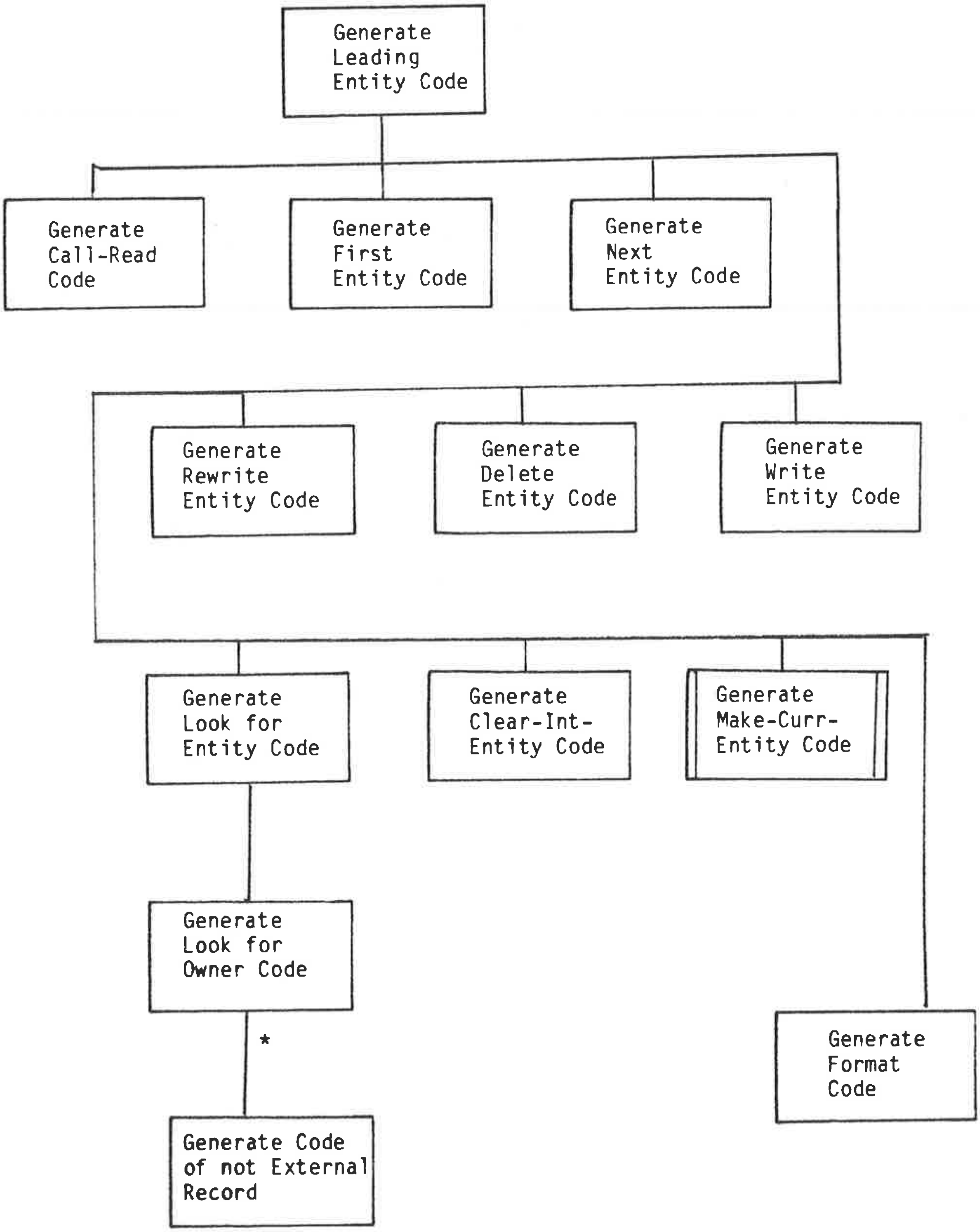

Match
Entities and
Records
*
Try and
Match Entity
and Record

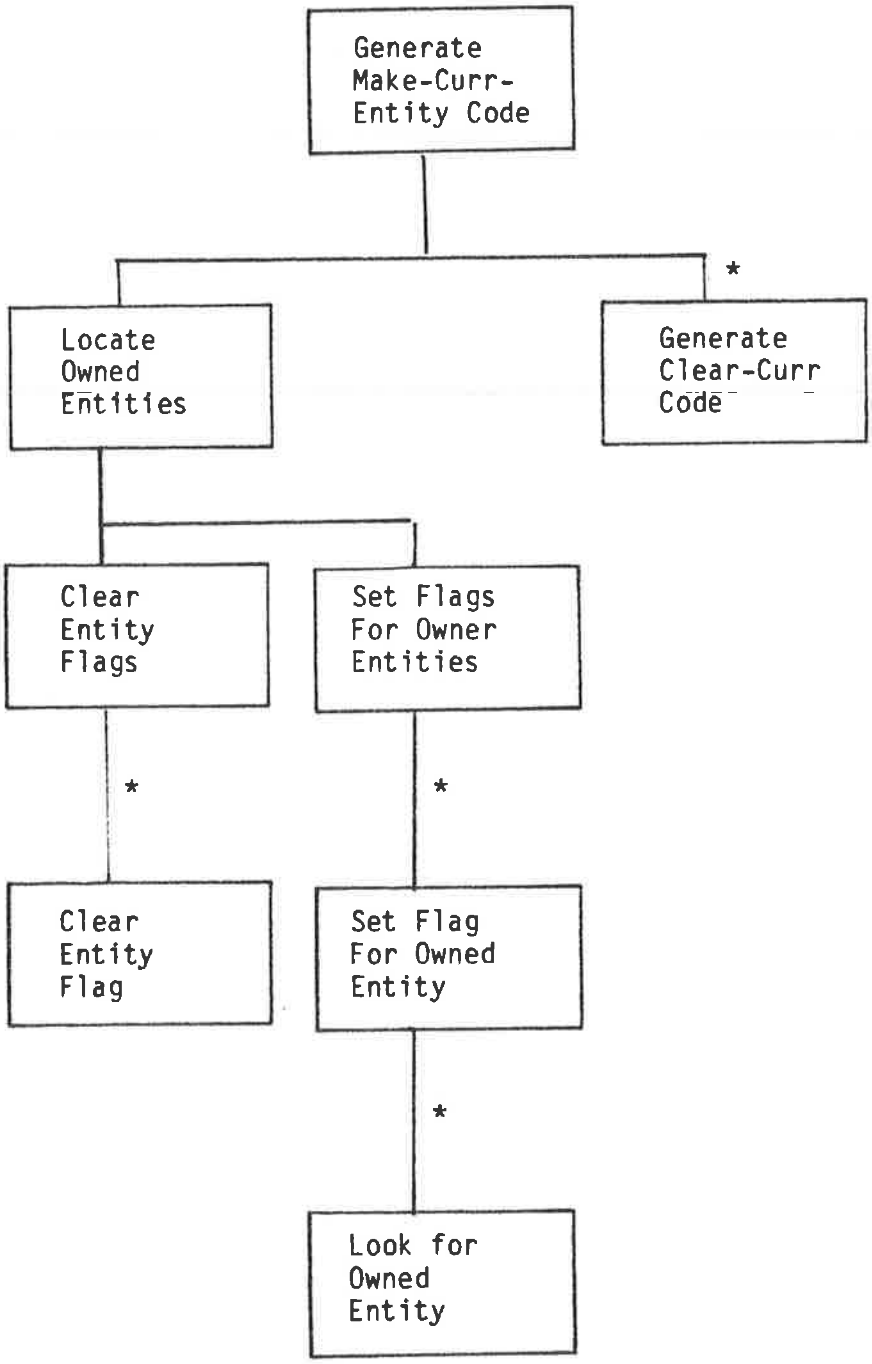

Generate Items
*
Generate User Record Copy
Generate Format Record Code
Generate Item
Generate Picture
Generate Format Item Code
Generate User Item Copy
Fill Item Code
Look for Entity
Fill Ext-Item Code
Fill Int-Item Code
Fill Int-Key Items
*
Search for Entity in File


Scan Internal Schema
*
Check if File Needed
Generate File Specification
Generate Record
Generate FD Entry
Generate Open and Close Code
Generate File Flags
Generate Entities
Generate File Key
*
Look For Key Fields
Generate File Key Field


Generate Entities
Fill Entity Table
Fill in External Owners
Locate Active Entities
Generate Entity if Active
*
*
*
*
Look for Entity Table Entry
Fill in External Owner
Fill in Active Entity
Generate Entity
*
See if External Record
Locate Owner
Find Owner Of Active Entity
*
*
Search for Owner


Generate Entity
Generate Leading Entity Code
Generate Read Code
Generate Set Up Key Code
Generate CUR-INT-entity
Generate Read Code
Generate Fields
Sort Field Table
Fill Field Table
Generate Trailing Entity Code
Sort Field Table Pass
Look for Field Table Entry
Sort Field Table Compare
Generate Entity Key Fields
Generate Field
Generate Entity Key Field
Generate the Field
Generate Picture
*


Generate Leading Entity Code
Generate Call-Read Code
Generate First Entity Code
Generate Next Entity Code
Generate Rewrite Entity Code
Generate Delete Entity Code
Generate Write Entity Code
Generate Look for Entity Code
Generate Clear-Int-Entity Code
Generate Make-Curr-Entity Code
Generate Look for Owner Code
*
Generate Code of not External Record
Generate Format Code


Generate
Make-Curr-
Entity Code
*
Locate
Owned
Entities
Generate
Clear-Curr
Code
Clear
Entity
Flags
Set Flags
For Owner
Entities
*
*
Clear
Entity
Flag
Set Flag
For Owned
Entity
*
Look for
Owned
Entity

APPENDIX 3 - SEQUENT EXAMPLES

This appendix gives examples of the use of the SEQUENT sequential file query system.

The layout of the example file is shown in Figure A3.1.

The dictionary file is set up by the use of the CYBER CCL* procedure call

SEQUENT, BUILD

which initiates the conversational style interface for building the dictionary (refer following pages).

The chosen example has eight fields

Employee number	- a four digit numeric field
Sex	- a single character field
Marital Status	- a single character field
Pay rate	- a three digit numeric field to one decimal place
Name	- a twenty-four character field which is subdivided also into Surname and Initials
Surname	- the first twenty characters of the Name field
Initials	- the last four characters of the Name field
Maiden-name	- a twenty character field only present for married females.

The file was set up in a standard COBOL program employing that language's WRITE statement.

The final example page of this appendix gives four examples of using the SEQUENT query facility on this file using the CYBER CCL command

SEQUENT, QUERY, I = query source

Included in these examples are the use of both simple and compound relation expressions, the use of all and/or part of the Name field, and the use of the optional maiden name field.

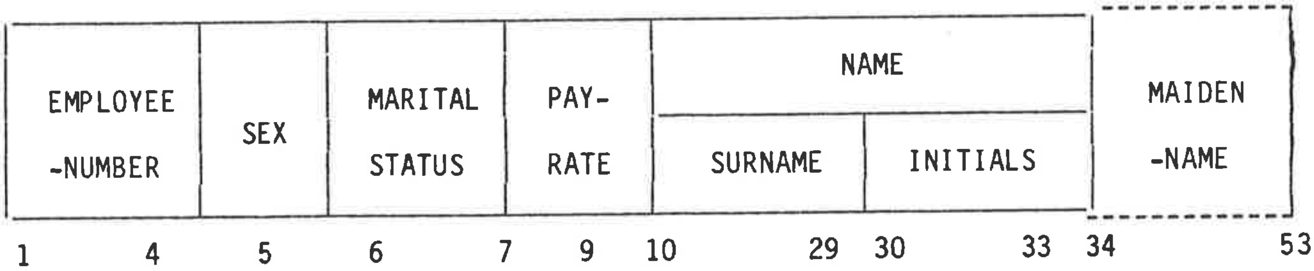


Figure A3.1: Example sequential file record layout

```
SEQUENT,BUILD

DEFINE SEQUENTIAL FILE DICTIONARY


ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME

? EMPLOYEE-NUMBER

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? N

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 004

ENTER NUMBER OF DECIMAL PLACES ( 1 DIGIT)

? 0

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0001

FIELD NAME       EMPLOYEE-NUMBER
FIELD TYPE       NUMERIC
FIELD LENGTH        4
FIELD POSITION      1

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME

? SEX

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 001

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0005

FIELD NAME       SEX
FIELD TYPE       CHARACTER
FIELD LENGTH        1
FIELD POSITION      5

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

?Y
```

```
ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME

? MARITAL-STATUS

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 001

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0006

FIELD NAME       MARITAL-STATUS
FIELD TYPE       CHARACTER
FIELD LENGTH        1
FIELD POSITION      6

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

?     0Y

ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME
? PAY-RATE

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? N

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 003

ENTER NUMBER OF DECIMAL PLACES ( 1 DIGIT)

? 1

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0007

FIELD NAME       PAY-RATE
FIELD TYPE       NUMERIC
FIELD LENGTH        3
DECIMAL PLACES      1
FIELD POSITION      7

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N

ENTER FIELD NAME

? SURNAME
```

```
ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 20020

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0010

FIELD NAME        SURNAME
FIELD TYPE        CHARACTER
FIELD LENGTH        20
FIELD POSITION      10

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME

? INITIALS

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 004

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0030

FIELD NAME        INITIALS
FIELD TYPE        CHARACTER
FIELD LENGTH         4
FIELD POSITION      30

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME

? NAME

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 024

ENTER FIELD POSITION (4 DIGITS FROM 0001)
```

```
? 0010

FIELD NAME       NAME
FIELD TYPE       CHARACTER
FIELD LENGTH       24
FIELD POSITION     10

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N
? Y

ENTER FIELD NAME
? MAIDEN-NAME

ENTER FIELD TYPE - C (CHARACTER) OR N (NUMERIC)

? C

ENTER LENGTH OF FIELD ( 3 DIGITS )

? 020

ENTER FIELD POSITION (4 DIGITS FROM 0001)

? 0034

FIELD NAME       MAIDEN-NAME
FIELD TYPE       CHARACTER
FIELD LENGTH       20
FIELD POSITION     34

ENTER Y TO ADD THIS FIELD TO THE DICTIONARY

? Y

ANY MORE FIELDS - ENTER Y OR N
? N

 8 FIELDS CREATED IN DICTIONARY

REVERT. BUILD.
/
```

```
SEQUENT,QUERY,I=TESTSQ1

*    THIS QUERY PRINTS OUT THE MARITAL STATUS,
*    AND NAME OF ALL MALES.

WHERE SEX = M PRINT MARITAL-STATUS, NAME.

S  SMITH                  J
M  WILSON                 KDS
REVERT. QUERY.
```

```
SEQUENT,QUERY,I=TESTSQ2

*    THIS QUERY PRINTS OUT THE MARITAL STATUS,
*    AND NAME OF ALL FEMALES.

WHERE SEX = F PRINT MARITAL-STATUS, NAME.

M  JONES                  KR
W  SMITH                  PA
REVERT. QUERY.
```

```
SEQUENT,QUERY,I=TESTSQ3

*    THIS QUERY PRINTS OUT THE FULL NAME OF
*    ALL EMPLOYEES WITH THE SURNAME SMITH

*    NOTE - SURNAME IS A SUB-FIELD OF FULL NAME
WHERE SURNAME = SMITH PRINT NAME.

SMITH                  J
SMITH                  PA
REVERT. QUERY.
```

```
SEQUENT,QUERY,I=TESTSQ4

*    THIS QUERY PRINTS OUT THE MAIDEN NAME OF
*    AND EMPLOYEE NUMBER OF ALL MARRIED FEMALES.

*    NOTE _ MAIDEN NAME IS ONLY SPECIFIED FOR
*           MARRIED WOMEN.

WHERE SEX = F AND MARITAL-STATUS = M
    PRINT EMPLOYEE-NUMBER, MAIDEN-NAME.

1257  WILSON
REVERT. QUERY.
```

APPENDIX 4 - INVERSE EXAMPLES

This appendix gives examples of the INVERSE inverted file query/update sub-system.

The first page of computer printout gives the DDL for building the inverted index. To reduce the size of example output only the first 60 of the 400,000 records on the file were indexed.

The original file contained details of all property sales in South Australia over a two year period. The records are 400 characters long (giving a file size of 16 mega-bytes) but only a few fields were described in the dictionary, and only a selection of these few were indexed. Again this was to reduce the complexity of the example for inclusion here. Of the eleven fields, only LGA, ZONING-CODE and LAND-USE-CODE were indexed.

The second and subsequent computer printout pages of this appendix give ten query/update requests that demonstrate many of the range of features available in the QUILL language used by the INVERSE system.

The ten queries demonstrate the following features

- simple and complex relational conditions including both equality and inequality
- print format control - page size
  - headings
  - page numbering
- updating selected records
- extraction of information onto "hit files"

```
INVERSE,BUILD,I=TESTV2
*    THIS SET OF "INVERSE" DDL HAS BEEN APPLIED TO
*    RECORDS 1 TO 60 (RATHER THAN THE WHOLE FILE OF
*    400,000 RECORDS) IN ORDER TO RESTRICT AMOUNT OF
*    OUTPUT FROM EACH EXAMPLE QUERY.

*    THE FOLLOWING FIELDS ARE DESCRIBED
*       LGA                  THE LOCAL GOVERNMENT AREA NUMBER
*       ZONING-CODE          LIN = LIGHT INDUSTRIAL
*                            GIN = GENERAL INDUSTRIAL
*       SALE-DATE            FORMAT YYMMDD
*       SALE-PRICE
*       FRONTAGE
*       LAND-USE-CODE        CURRENT USE OF LAND
*       GRAPHIC-INDEX
*       IMPROVEMENTS-CODE BUILDINGS ON SITE
*       AREA-HECTARES
*       OLD-NAME             FORMER OWNER
*       NEW-NAME             CURRENT OWNER

INVERT FROM 1 TO 60 .
PRINT SUMMARY.
INDEX FIELD NAME IS LGA
      POSITION IS 1
      TYPE IS ALPHA
      LENGTH IS 2.
INDEX FIELD NAME IS ZONING-CODE
      POSITION IS 205
      TYPE IS ALPHA
      LENGTH IS 3.
FIELD NAME IS SALE-DATE
      POSITION IS 11 TYPE IS NUMERIC LENGTH IS 6.
FIELD NAME IS SALE-PRICE POSITION IS 23 TYPE IS NUMERIC LENGTH IS 8.
FIELD NAME IS FRONTAGE POSITION IS 50 TYPE IS NUMERIC LENGTH IS 5.
INDEX FIELD NAME IS LAND-USE-CODE POSITION IS 164 TYPE IS NUMERIC
                    LENGTH IS 4.
FIELD NAME IS GRAPHIC-INDEX POSITION IS 168 TYPE IS ALPHA
                    LENGTH IS 10.
FIELD NAME IS IMPROVEMENTS-CODE POSITION IS 178 TYPE IS ALPHA
                    LENGTH IS 15.
FIELD NAME IS AREA-HECTARES POSITION IS 193 TYPE IS NUMERIC
                    LENGTH IS 8.
FIELD NAME IS OLD-NAME POSITION IS 214 TYPE IS ALPHA
                    LENGTH IS 60.
FIELD NAME IS NEW-NAME POSITION IS 274 TYPE IS ALPHA
                    LENGTH IS 60.
REVERT. BUILD.
```

```
INVERSE,QUERY,I=TESTQ1
WHERE LAND-USE-CODE = 1100 PRINT LGA, ZONING-CODE,
SALE-DATE
HEADING "LGA ZONE  DATE".
LGA ZONE  DATE
22   GIN   800729
22   GIN   791120
22   GIN   800922
22   GIN   790907
22   LIN   790627
22   LIN   801110
22   LIN   810112
22   LIN   800102
22   GIN   791018
22   LIN   800714
22   LIN   800501
22   LIN   801224
22   LIN   790119
22   GIN   790704
22   GIN   810116
22   LIN   790529
22   GIN   790801
22   GIN   800130
22   GIN   791126
22   GIN   790126
22   GIN   800702
22   GIN   790517
22   GIN   810130
22   GIN   810127
22   GIN   790412
22   GIN   800130
22   GIN   800911
22   GIN   800430
REVERT.QUERY.
/
```

```
INVERSE,QUERY,I=TESTQ2
 *   THIS QUERY PRINTS OUT THE LGA AND OWNER NAME
 *   OF ALL SALES OF LAND CURRENTLY USED AS A QUARRY
 *   AND ZONED LIGHT INDUSTRIAL.
 *   THE PAGE LENGTH HAS BEEN SET TO 15 LINES AND
 *   THE PAGE NUMBER IS TO BE PRINTED IN COL. 40.
 *   A THREE LINE HEADING IS TO BE PRINTED ON EACH
 *   PAGE.
WHERE LAND-USE-CODE = 1100 AND ZONING-CODE = LIN PRINT LGA, NEW-NAME
HEADING "LGA NEW NAME" ON LINE 1
HEADING "--- --- ----" ON LINE 2
HEADING "" ON LINE 3
CONTROL PAGE LENGTH 15
CONTROL PAGE NUMBER 40.
1                                         PAGE     1
LGA NEW NAME
--- --- ----

22  MR  C S + P A CARAPETIS 4 JAMES ST THEBARTON
22  W G + I M HARLEY 6 PATRICIA AVE CAMDEN                      50
22  R & P MATHIEU 74 MARIA STREET THEBARTON                      5
22  GRANDAL NOMINEES PTY LTD 33 WEST THEBARTON RD THEBARTON
22  MR  A ELALI 77 LINDSAY ST PERTH                      6000
22  PANVIC PTY LTD C/O 54 BURLINGTON ST WALKERVILLE
22  J R POPE 3 WHITING ST SEACOMBE HEIGHTS                      50
22  MR  D H + J A MATHEWS 3 JAMES ST THEBARTON
22  MR  G + T MAZARAKOS 120 WRIGHT ST ADELAIDE
REVERT.QUERY.
```

```
INVERSE,QUERY,I=TESTQ3
WHERE LAND-USE-CODE = 1100 PRINT LGA, ZONING-CODE, SALE-PRICE
HEADING "LGA ZONE  SALE-PRICE".
1
LGA ZONE  SALE-PRICE
22   GIN  00021000
22   GIN  00025000
22   GIN  00028000
22   GIN  00185000
22   LIN  00015000
22   LIN  00025500
22   LIN  00030500
22   LIN  00023000
22   GIN  00016000
22   LIN  00009500
22   LIN  00025000
22   LIN  00022000
22   LIN  00028500
22   GIN  00027500
22   GIN  00027500
22   LIN  00023500
22   GIN  00025500
22   GIN  00020500
22   GIN  00027500
22   GIN  00025000
22   GIN  00050000
22   GIN  00037000
22   GIN  00060000
22   GIN  00030000
22   GIN  00060000
22   GIN  00023000
22   GIN  00030000
22   GIN  00028000
REVERT. QUERY.
```

```
INVERSE,QUERY,I=TEST04
?
*    THIS QUERY PRINTS OUT ALL LAND CURRENTLY USED
*    FOR QUARRIES AND ALSO UPDATES THE SALE PRICE
*    BY 500.
WHERE LAND-USE-CODE = 1100 PRINT LGA, ZONING-CODE, SALE-PRICE
INCREASE SALE-PRICE BY 500
HEADING "LGA ZONE SALE-PRICE".
?
LGA ZONE SALE-PRICE
22   GIN   00021500
22   GIN   00025500
22   GIN   00028500
22   GIN   00185500
22   LIN   00015500
22   LIN   00026000
22   LIN   00031000
22   LIN   00023500
22   GIN   00016500
22   LIN   00010000
22   LIN   00025500
22   LIN   00022500
22   LIN   00029000
22   GIN   00028000
22   GIN   00028000
22   LIN   00024000
22   GIN   00026000
22   GIN   00021000
22   GIN   00028000
22   GIN   00025500
22   GIN   00050500
22   GIN   00037500
22   GIN   00060500
22   GIN   00030500
22   GIN   00060500
22   GIN   00023500
22   GIN   00030500
22   GIN   00028500
REVERT.QUERY.
?
```

```
INVERSE,QUERY,I=TESTQ5
*    THIS QUERY ILLUSTRATES THE USE OF AN
*    INEQUALITY RELATIONSHIP.

WHERE LAND-USE-CODE < 1200 AND ZONING-CODE = LIN
PRINT LGA, SALE-PRICE, NEW-NAME.

1
22   00015000  MR  C S + P A CARAPETIS 4 JAMES ST THEBARTON
22   00025500  W G + I M HARLEY 6 PATRICIA AVE CAMDEN                     5
22   00030500  R & P MATHIEU 74 MARIA STREET THEBARTON
22   00023000  GRANDAL NOMINEES PTY LTD 33 WEST THEBARTON RD THEBARTON
22   00009500  MR  A ELALI 77 LINDSAY ST PERTH                        6000
22   00025000  PANVIC PTY LTD C/O 54 BURLINGTON ST WALKERVILLE
22   00022000  J R POPE 3 WHITING ST SEACOMBE HEIGHTS                     5
22   00028500  MR  D H + J A MATHEWS 3 JAMES ST THEBARTON
22   00023500  MR  G + T MAZARAKOS 120 WRIGHT ST ADELAIDE
REVERT.QUERY.
```

```
INVERSE,QUERY,I=TESTQ6

*    THIS QUERY PRINTS OUT THE LGA AND OWNER NAME
*    OF ALL SALES OF LAND CURRENTLY USED AS A QUARRY
*    AND ZONED GENERAL INDUSTRIAL.

*    THE PAGE LENGTH HAS BEEN SET TO 15 LINES AND
*    THE PAGE NUMBER IS TO BE PRINTED IN COL. 40.

*    A THREE LINE HEADING IS TO BE PRINTED ON EACH
*    PAGE.

*    THIS QUERY ALSO EXTRACTS FOUR FIELDS FROM
*    EACH SELECTED RECORD AND WRITES THEM TO
*    AN EXTRACT FILE.
WHERE LAND-USE-CODE = 1100 AND ZONING-CODE = GIN
PRINT LGA, NEW-NAME
HEADING "LGA NEW NAME" ON LINE 1
HEADING "--- --- ----" ON LINE 2
HEADING "" ON LINE 3
EXTRACT LGA, LAND-USE-CODE, ZONING-CODE AND NEW-NAME.
1
LGA NEW NAME
--- --- ----

22   M/S J LILITH 30 KINTORE ST THEBARTON                              5031
22   MR   H + A AMANATIDIS 22 KINTORE ST THEBARTON
22   ZIFF PTY LTD C/O 9 BLUELAKE CT TENNYSON                              5
22   ZIFF PTY LTD C/O TOUCHE ROSS + CO 45 GRENFELL ST ADELAIDE
22   MR   D + P PARSALIDIS 29 LIGHT TCE THEBARTON
22   MR   J D PHILLIPS 26 JAMES ST THEBARTON                              50
22   P & M IOANNOU 34 PHILLIPS ST THEBARTON                               50
22   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE                               5
22   EVANGELISTA NOMINEES PTY LTD 227 RUNDLE ST ADELAIDE
22   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE                               5
22   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE                               5
22   S A BREWING CO LTD 224 HINDLEY ST ADELAIDE
22   GALICIA PTY LTD 33 PIRIE ST ADELAIDE                               5000
22   DIVERSE PRODUCTS LTD 39 PORT RD THEBARTON
22   DUNEDIN NOMINEES PTY LTD 456 PULTENEY ST ADELAIDE
22   DIVERSE PRODUCTS LTD 37 PORT RD THEBARTON
22   MR   O + A CARRABS 63 CUDMORE TCE MARLESTON
22   MR   I G + C D FRASER 50 WEST THEBARTON RD THEBARTON
22   MR   B + S E GLEDHILL 3 WARE ST THEBARTON
REVERT. QUERY.
```

```
INVERSE,QUERY,I=TESTQ7

*    THIS QUERY PRINTS OUT THE LGA AND OWNER NAME
*    OF ALL SALES OF LAND CURRENTLY USED AS A QUARRY
*    AND ZONED LIGHT INDUSTRIAL.
*    THE PAGE LENGTH HAS BEEN SET TO 7 LINES AND
*    THE PAGE NUMBER IS TO BE PRINTED IN COL. 40.

*    A THREE LINE HEADING IS TO BE PRINTED ON EACH
*    PAGE.

*    THE PRINT ACTION USES THE 'SPACE' OPTION IN ORDER
*    TO OVERRIDE THE DEFAULT SPACING.

WHERE LAND-USE-CODE = 1100 AND ZONING-CODE = LIN
PRINT LGA, SPACE 2 NEW-NAME
HEADING 'LGA NEW NAME' ON LINE 1
HEADING '--- --- ----' ON LINE 2
HEADING '' ON LINE 3
CONTROL PAGE LENGTH 7
CONTROL PAGE NUMBER 40.
1                                  PAGE    1
LGA NEW NAME
--- --- ----

22     MR  C S + P A CARAPETIS 4 JAMES ST THEBARTON
22     W G + I M HARLEY 6 PATRICIA AVE CAMDEN                     50
22     R & P MATHIEU 74 MARIA STREET THEBARTON                     5
22     GRANDAL NOMINEES PTY LTD 33 WEST THEBARTON RD THEBARTON
1                                  PAGE    2
LGA NEW NAME
--- --- ----

22     MR  A ELALI 77 LINDSAY ST PERTH                      6000
22     PANVIC PTY LTD C/O 54 BURLINGTON ST WALKERVILLE
22     J R POPE 3 WHITING ST SEACOMBE HEIGHTS                     50
22     MR  D H + J A MATHEWS 3 JAMES ST THEBARTON
1                                  PAGE    3
LGA NEW NAME
--- --- ----

22     MR  G + T MAZARAKOS 120 WRIGHT ST ADELAIDE
REVERT.QUERY.
/
```

```
INVERSE,QUERY,I=TESTQ8

*   THIS QUERY DEMONSTRATES A SIMPLE BOOLEAN EXPRESSION

WHERE LAND-USE-CODE = 1200
PRINT LGA, LAND-USE-CODE, ZONING-CODE, NEW-NAME.
122   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   S A HOUSING TRUST 17 ANGAS ST ADELAIDE
 22   1200   GIN   GALICIA PTY LTD 33 PIRIE ST ADELAIDE                      50
 22   1200   GIN   CLOVERCREST FINANCE + INV PTY LTD 1032 PORT RD ALBERT PARK
REVERT.QUERY.

INVERSE,QUERY,I=TESTQ9

*   THIS QUERY DEMONSTRATES A COMPOUND BOOLEAN EXPRESSION

WHERE LAND-USE-CODE = 1100 AND ZONING-CODE = GIN
PRINT LGA, LAND-USE-CODE, ZONING-CODE, NEW-NAME.
1
 22   1100   GIN   M/S J LILITH 30 KINTORE ST THEBARTON                      50
 22   1100   GIN   MR  H + A AMANATIDIS 22 KINTORE ST THEBARTON
 22   1100   GIN   ZIFF PTY LTD C/O 9 BLUELAKE CT TENNYSON
 22   1100   GIN   ZIFF PTY LTD C/O TOUCHE ROSS + CO 45 GRENFELL ST ADELAIDE
 22   1100   GIN   MR  D + P PARSALIDIS 29 LIGHT TCE THEBARTON
 22   1100   GIN   MR  J D PHILLIPS 26 JAMES ST THEBARTON
 22   1100   GIN   P & M IOANNOU 34 PHILLIPS ST THEBARTON
 22   1100   GIN   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
 22   1100   GIN   EVANGELISTA NOMINEES PTY LTD 227 RUNDLE ST ADELAIDE
 22   1100   GIN   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
 22   1100   GIN   HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
 22   1100   GIN   S A BREWING CO LTD 224 HINDLEY ST ADELAIDE
 22   1100   GIN   GALICIA PTY LTD 33 PIRIE ST ADELAIDE                      50
 22   1100   GIN   DIVERSE PRODUCTS LTD 39 PORT RD THEBARTON
 22   1100   GIN   DUNEDIN NOMINEES PTY LTD 456 PULTENEY ST ADELAIDE
 22   1100   GIN   DIVERSE PRODUCTS LTD 37 PORT RD THEBARTON
 22   1100   GIN   MR  O + A CARRABS 63 CUDMORE TCE MARLESTON
 22   1100   GIN   MR  I G + C D FRASER 50 WEST THEBARTON RD THEBARTON
 22   1100   GIN   MR  B + S E GLEDHILL 3 WARE ST THEBARTON
REVERT.QUERY.
```

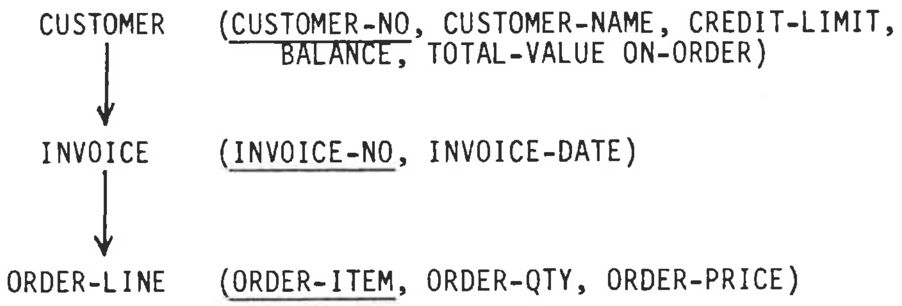
```
INVERSE,QUERY,I=TESTQ10
*   THIS QUERY DEMONSTRATES THE USE OF PARENTHESES
*   TO CONTROL THE ORDER OF EVALUATION OF COMPLEX
*   BOOLEAN EXPRESSIONS

WHERE ((LAND-USE-CODE = 1100 AND ZONING-CODE = GIN) OR
(LAND-USE-CODE = 1200))
PRINT LGA, LAND-USE-CODE, ZONING-CODE, NEW-NAME.
1
22  1100  GIN  M/S J LILITH 30 KINTORE ST THEBARTON                          50:
22  1100  GIN  MR  H + A AMANATIDIS 22 KINTORE ST THEBARTON
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1100  GIN  ZIFF PTY LTD C/O 9 BLUELAKE CT TENNYSON
22  1100  GIN  ZIFF PTY LTD C/O TOUCHE ROSS + CO 45 GRENFELL ST ADELAIDE
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1200  GIN  S A HOUSING TRUST 17 ANGAS ST ADELAIDE                          !
22  1100  GIN  MR  D + P PARSALIDIS 29 LIGHT TCE THEBARTON
22  1100  GIN  MR  J D PHILLIPS 26 JAMES ST THEBARTON                          !
22  1100  GIN  P & M IOANNOU 34 PHILLIPS ST THEBARTON                          !
22  1100  GIN  HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
22  1100  GIN  EVANGELISTA NOMINEES PTY LTD 227 RUNDLE ST ADELAIDE
22  1100  GIN  HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
22  1100  GIN  HIGHWAYS DEPT 33 WARWICK ST WALKERVILLE
22  1100  GIN  S A BREWING CO LTD 224 HINDLEY ST ADELAIDE
22  1100  GIN  GALICIA PTY LTD 33 PIRIE ST ADELAIDE                           50
22  1100  GIN  DIVERSE PRODUCTS LTD 39 PORT RD THEBARTON
22  1200  GIN  GALICIA PTY LTD 33 PIRIE ST ADELAIDE                           50
22  1100  GIN  DUNEDIN NOMINEES PTY LTD 456 PULTENEY ST ADELAIDE
22  1200  GIN  CLOVERCREST FINANCE + INV PTY LTD 1032 PORT RD ALBERT PARK
22  1100  GIN  DIVERSE PRODUCTS LTD 37 PORT RD THEBARTON
22  1100  GIN  MR  O + A CARRABS 63 CUDMORE TCE MARLESTON
22  1100  GIN  MR  I G + C D FRASER 50 WEST THEBARTON RD THEBARTON
22  1100  GIN  MR  B + S E GLEDHILL 3 WARE ST THEBARTON
REVERT.QUERY.
/
```

APPENDIX 5 - PYRAMID EXAMPLES

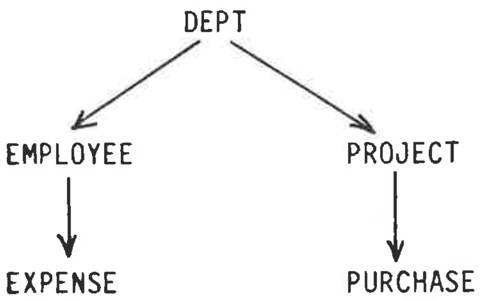
This appendix gives examples of the use of the PYRAMID hierarchical database subsystem.

The internal dictionary has been set up using five sets of Internal Schema DDL.

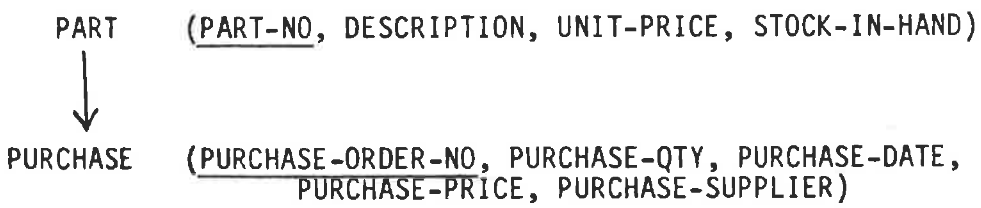
TESTI1 describes the Customer database with the 3 entities



TESTI2 is an unrelated database with a multi-leg hierarchy

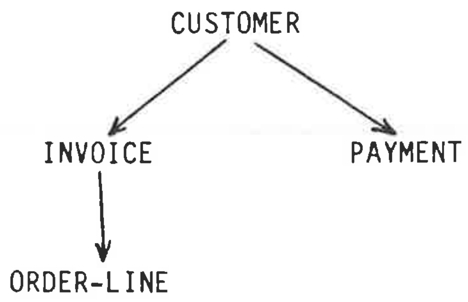


TESTI3 describes the layout of the Inventory database and its two entities.



TESTI4 combines the Customers and Inventory databases of TESTI1 and TESTI3. It is set up for the order-entry and invoice print External Schemas.

TESTI5 is an extension of the Customers data base of TESTI1, with the Payment entity being added to convert the single leg hierarchy to a multiple leg hierarchy.



This Payment entity has been added to illustrate the ability of PYRAMID databases to have extra entity types added without making existing databases redundant. By adding six spaces (for PAYMENT-DATE) into all existing records, the same data records can be matched to the new internal schema.

In a real-life situation TESTI1 and TESTI4 could co-exist for different applications, but the advent of the changes in TESTI1 to create TESTI5 would require corresponding alterations to TESTI4.

The external dictionary has been set up for seven user interfaces.

TESTE1 is an interface to the CUSTOMERS file. It was set up for the initial order-entry process when that program accepted orders without checking the stock-in-hand of the ordered parts. Notice that the user processes a 40 character customer name while the database uses a 30 character field. Notice also that the internal entity ORDER-LINE has been renamed ORDER for the user interface, and also that the internal attribute ORDER-QTY has been renamed as the user field QTY.

TESTE2 is an interface suitable for maintaining the file of parts, including stock levels and the history of purchases to replenish these stock levels.

TESTE3 collapses the three level hierarchy of TESTE1 into a single user record. Its primary use is for incorporation with QLSCE so that the QUILL language can be used to interrogate the file.

TESTE4 is an example of converting a three level internal schema into a two level external schema. It is thus an interface part way between the extremes of TESTE1 and TESTE3.

TESTE5 and TESTE6 are interfaces to the COMPANY internal schema of TESTI2. TESTE5 uses a single external name (NAME) for the internal names SURNAME and INITIALS.

TESTE7 is the revision of TESTE1 to allow the order entry program to check the stock-in-hand of the part records. The PURCHASE record is not really required, but has been included in case a further enhancement to the order entry program needs to make purchases as "back-orders".

After the twelve sets of DDL, the appendix contains five example user programs for creating parts and customers, taking orders, and printing invoices (see Figure A5.1).

Program CRCUST is the main subprogram of the CREATE program.

The purpose of the program is to create the CUSTOMERS database. This is achieved by the DBMS call

```
MOVE "RELEASE" TO FUNCTION.
CALL "DBMS" USING FUNCTION, RECORD-NAME, BUFFER, RESULT.
```

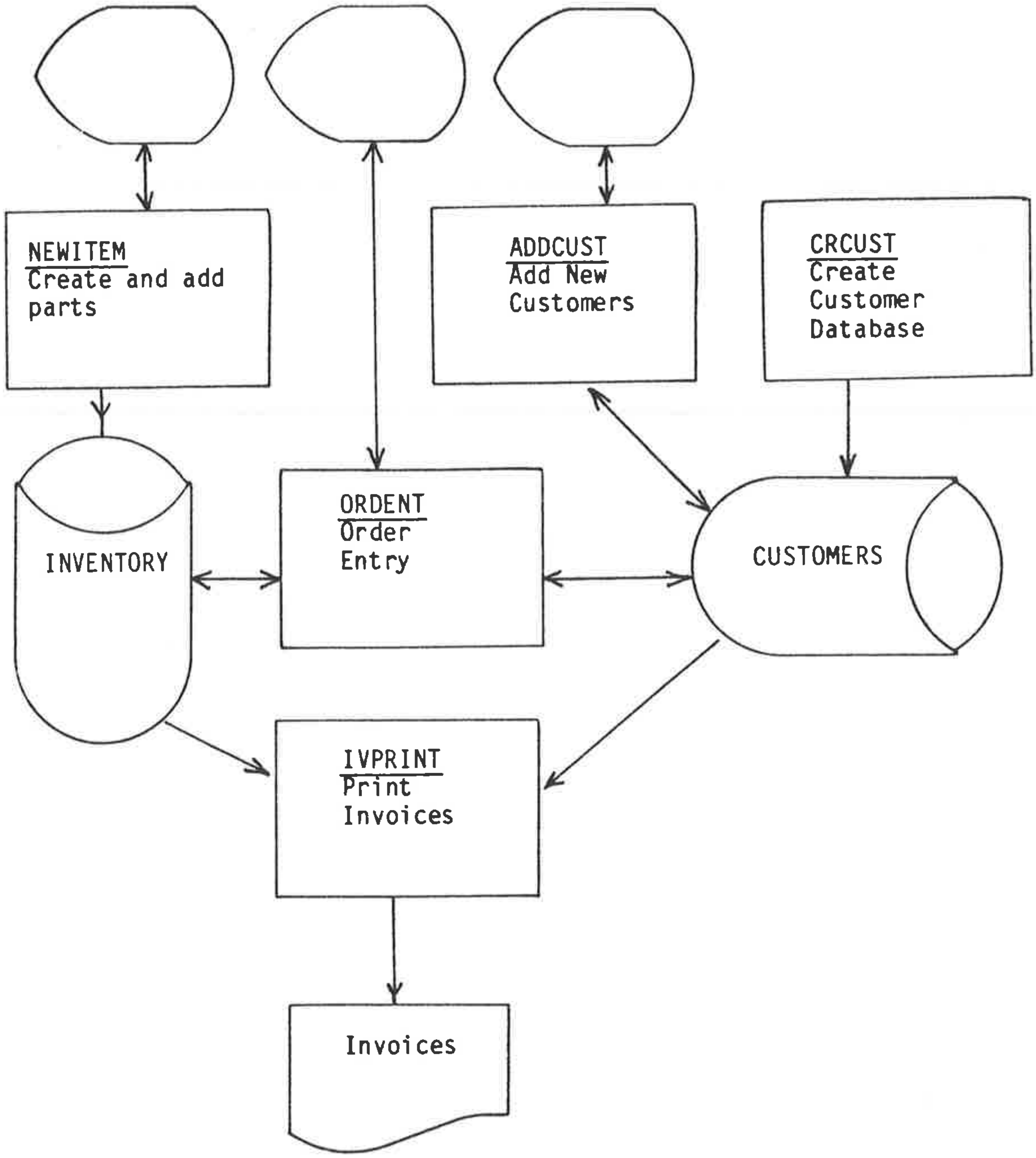


Figure A5.1: Order-entry system chart

Following this the empty database now exists and can be used by other programs to add, modify and retrieve order entry data.

Program ADDCUST picks up the CUSTOMERS database (either empty or partially full) and adds new customers to it.

The database is opened by the DBMS call

```
MOVE "OLD" TO FUNCTION
CALL "DBMS" USING FUNCTION, RECORD-NAME, BUFFER, RESULT.
```

New customers are written using the DBMS call

```
MOVE "WRITE" TO FUNCTION.
MOVE "CUSTOMER" TO RECORD-NAME.
CALL "DBMS" USING FUNCTION, RECORD-NAME, BUFFER, RESULT.
```

Program NEWITEM combines the activities of the above two programs and both creates and loads the INVENTORY database.

Program ORDENT updates the CUSTOMERS and INVENTORY databases with the details of orders taken. PART records are read with the DBMS call

```
ACCEPT ORDER-ITEM.
MOVE ORDER-ITEM TO PART-NO.
CALL DBMS USING READ-FUNCTION, PART-RECORD, PART, RESULT
```

and the record with STOCK-IN-HAND adjusted is replaced using the DBMS call

```
CALL "DBMS" USING REWRITE-FUNCTION, PART-RECORD, PART, RESULT.
```

Program IVPRINT reads sequences of records to form invoices. It includes DBMS calls of the form

```
CALL "DBMS" USING NEXT-FUNCTION, ORDER-RECORD, ORDER-LINE, RESULT.
```

Following the five programs referred to above the next page of the appendix gives 3 examples of the use of the QUILL query language on the CUSTOMERS database. In the CYBER CCL call

PYRAMID, QUERY, I= TESTPQ1, D = ORDERS

the ORDERS is the catalogue name for the CUSTOMERS database.

Finally the appendix contains a selection of database interface subprograms generated by PYRAMID. Each of these subprograms is introduced by a page explaining its potential use.

```
PYRAMID,INTDDL,I=TEST11

NEW DICTIONARY.

INTERNAL SCHEMA NAME IS MANUFACTURING.

FILE NAME IS CUSTOMERS; ORGANIZATION IS INDEXED;
ASSIGN TO ORDERS.

ENTITY NAME IS CUSTOMER; KEY IS CUSTOMER-NO
(CUSTOMER-NO/C 6,CUSTOMER-NAME/C 30,CREDIT-LIMIT/N 8.2,
BALANCE/N 10.2,TOTAL-VALUE-ON-ORDER/N 8.2 ).

ENTITY NAME IS INVOICE; OWNER IS CUSTOMER; KEY IS INVOICE-NO
(INVOICE-NO/C 6,INVOICE-DATE/N 6).

ENTITY NAME IS ORDER-LINE; KEY IS ORDER-ITEM;
OWNER IS INVOICE;
(ORDER-ITEM/C 4,ORDER-QTY/N 6,ORDER-PRICE/N 5.2).
REVERT.INTDDL.
/
```

```
PYRAMID,INTDDL,I=TEST12

 *   THIS SET OF "PYRAMID" INTERNAL SCHEMA DDL DESCRIBES A
 *   SINGLE DATABASE WHICH IS A MULTI-LEG HIERARCHY.
 *
 *                     DEPARTMENT
 *                          .
 *                        .   .
 *                      .       .
 *                    .           .
 *             EMPLOYEE       PROJECT
 *                  .             .
 *                  .             .
 *                  .             .
 *             EXPENSE       PURCHASE
 *
 INTERNAL SCHEMA NAME IS COMPANY.

 FILE NAME IS COMPANY; ORGANIZATION IS INDEXED;
 ASSIGN TO COMPANY.

 ENTITY NAME IS DEPARTMENT; KEY IS DEPT-NO
 (DEPT-NO/C 2,DEPT-NAME/C 20).

 ENTITY NAME IS EMPLOYEE; OWNER IS DEPARTMENT;
 KEY IS EMP-NO(EMP-NO/C 4,NAME(SURNAME/C 20,INITIALS/C 4),
 SEX/C 1,SALARY/N 5).

 ENTITY NAME IS PROJECT;OWNER IS DEPARTMENT;
 KEY IS PROJ-NO(PROJ-NO/C 6,PROJ-NAME/C 20,BUDGET/N 7).

 ENTITY NAME IS PURCHASE;OWNER IS PROJECT;
 KEY IS PURCHASE-ORDER-NO(PURCHASE-ORDER-NO/N 5,
 AMOUNT/N 8.2).

 ENTITY NAME IS EXPENSE; OWNER IS EMPLOYEE;
 KEY IS EXPENSE-CODE(EXPENSE-CODE/C 1,RATE/N 4.2).
REVERT.INTDDL.
/
```

```
      PYRAMID,INTDDL,I=TEST13

 *   THIS SET OF "PYRAMID" INTERNAL SCHEMA DDL DESCRIBES
 *   THE LAYOUT OF THE INVENTORY DATA BASE WHICH CONTAINS
 *   PART ENTITIES OWNING PURCHASE ENTITIES.

 INTERNAL SCHEMA NAME IS INVENTORY.

 FILE NAME IS INVENTORY; ORGANIZATION IS INDEXED;
 ASSIGN TO PARTS.

 ENTITY NAME IS PART; KEY IS PART-NO
 (PART-NO/C 4,DESCRIPTION/C 40,UNIT-PRICE/N 6.2,
 STOCK-IN-HAND/N 6).

 ENTITY NAME IS PURCHASE; OWNER IS PART; KEY IS PURCHASE-ORDER-NO
 (PURCHASE-ORDER-NO/C 4,PURCHASE-QTY/N 6,PURCHASE-DATE/N 6,
 PURCHASE-PRICE/N 6.2, PURCHASE-SUPPLIER-NO/C 4).
REVERT.INTDDL.
/*
```

```
PYRAMID, INTDDL, I=TEST14

*    THIS SET OF "PYRAMID" INTERNAL SCHEMA DDL DESCRIBES
*    THE LAYOUT OF THE TWO DATA BASES WHICH CONTAIN
*    PART ENTITIES OWNING PURCHASE ENTITIES, AND CUSTOMERS
*    OWNING INVOICES OWNING ORDER LINES.
INTERNAL SCHEMA NAME IS DOUBLE.
 FILE NAME IS INVENTORY; ORGANIZATION IS INDEXED;
 ASSIGN TO PARTS.

 ENTITY NAME IS PART; KEY IS PART-NO
 (PART-NO/C 4,DESCRIPTION/C 40,UNIT-PRICE/N 6.2,
 STOCK-IN-HAND/N 6).

 ENTITY NAME IS PURCHASE; OWNER IS PART; KEY IS PURCHASE-ORDER-NO
 (PURCHASE-ORDER-NO/C 4,PURCHASE-QTY/N 6,PURCHASE-DATE/N 6,
 PURCHASE-PRICE/N 6.2, PURCHASE-SUPPLIER-NO/C 4).
 FILE NAME IS CUSTOMERS; ORGANIZATION IS INDEXED;
 ASSIGN TO ORDERS.

 ENTITY NAME IS CUSTOMER; KEY IS CUSTOMER-NO
 (CUSTOMER-NO/C 6,CUSTOMER-NAME/C 30,CREDIT-LIMIT/N 8.2,
 BALANCE/N 10.2,TOTAL-VALUE-ON-ORDER/N 8.2 ).

 ENTITY NAME IS INVOICE; OWNER IS CUSTOMER; KEY IS INVOICE-NO
 (INVOICE-NO/C 6,INVOICE-DATE/N 6).

 ENTITY NAME IS ORDER-LINE; KEY IS ORDER-ITEM;
 OWNER IS INVOICE;
 (ORDER-ITEM/C 4,ORDER-QTY/N 6,ORDER-PRICE/N 5.2).
REVERT.INTDDL.
/
```

```
PYRAMID,INTDDL,I=TESTI5

*    THIS SET OF "PYRAMID" INTERNAL SCHEMA DDL DESCRIBES A
*    SINGLE DATABASE WHICH IS A MULTI-LEG HIERARCHY.
*
*                        DEPARTMENT
*                             .
*                          .     .
*                       .           .
*                    .                 .
*                 INVOICE        PAYMENT
*                    .
*                    .
*                    .
*              ORDER-LINE
*

INTERNAL SCHEMA NAME IS ACCOUNTING.

FILE NAME IS CUSTOMERS; ORGANIZATION IS INDEXED;
ASSIGN TO ORDERS.

ENTITY NAME IS CUSTOMER; KEY IS CUSTOMER-NO
(CUSTOMER-NO/C 6,CUSTOMER-NAME/C 30,CREDIT-LIMIT/N 8.2,
BALANCE/N 10.2,TOTAL-VALUE-ON-ORDER/N 8.2 ).

ENTITY NAME IS INVOICE; OWNER IS CUSTOMER; KEY IS INVOICE-NO
(INVOICE-NO/C 6,INVOICE-DATE/N 6).

ENTITY NAME IS ORDER-LINE; KEY IS ORDER-ITEM;
OWNER IS INVOICE;
(ORDER-ITEM/C 4,ORDER-QTY/N 6,ORDER-PRICE/N 5.2).

ENTITY NAME IS PAYMENT; KEY IS PAYMENT-DATE;
OWNER IS CUSTOMER;
(PAYMENT-DATE/C 6, PAYMENT-AMOUNT/N 6.2).

REVERT.INTDDL.
/
```

```
PYRAMID,EXTDDL,I=TESTE1
*   THIS SET OF "PYRAMID" EXTERNAL SCHEMA DDL DESCRIBES
*   A THREE LEVEL STRUCTURE ( THE SAME AS THE INTERNAL
*   SCHEMA).

*   NOTE - THE FIELD TOTAL-VALUE-ON-ORDER HAS BEEN RENAMED
*          TOT-VAL FOR SHORT.

*          THE RECORD ORDER-LINE HAS BEEN RENAMED TO ORDER

*   THE NEW DICTIONARY STATEMENT HAS BEEN INCLUDED
*   AS THIS IS THE FIRST EXTERNAL VIEW TO BE PLACED
*   IN THE EXTERNAL VIEW DICTIONARY.

NEW DICTIONARY.

EXTERNAL SCHEMA NAME IS ORDER-ENTRY
PERMIT ACCESS FOR UPDATE,RETRIEVE, CREATE, FORMAT.

RECORD NAME IS CUSTOMER(CUSTOMER-NAME/C 40,CUSTOMER-NO/C 6,
CREDIT-LIMIT/N 8.2,TOTAL-VALUE-ON-ORDER=TOT-VAL/N 8.2 ).

RECORD NAME IS INVOICE(INVOICE-NO/C 6,INVOICE-DATE/N 6).

RECORD ORDER-LINE = ORDER(ORDER-ITEM/C 4,ORDER-PRICE/N 5.2,
ORDER-QTY=QTY/N 6).
REVERT.EXTDDL.
/
```

```
PYRAMID,EXTDDL,I=TESTE2

*    THIS SET OF "PYRAMID" EXTERNAL SCHEMA DDL DESCRIBES
*    THE VIEW OF THE DATA BASE USED FOR MAINTAINING PART
*    DETAILS AND FOR RECORDING PURCHASES OF STOCK INTO
*    THE INVENTORY.

EXTERNAL SCHEMA NAME IS PURCHASES
PERMIT ACCESS FOR UPDATE,RETRIEVE, CREATE, FORMAT.

RECORD NAME IS PART(DESCRIPTION/C 40,PART-NO/C 4,
UNIT-PRICE/N 6.2,STOCK-IN-HAND/N 6).

RECORD NAME IS PURCHASE(PURCHASE-ORDER-NO/C 4,PURCHASE-DATE/N 6,
PURCHASE-QTY/N 6,PURCHASE-PRICE/N 6.2,PURCHASE-SUPPLIER-NO/C 4).

REVERT.EXTDDL.
/
```

```
PYRAMID,EXTDDL,I=TESTE3
*    THIS SET OF "PYRAMID" EXTERNAL SCHEMA DDL IS A SINGLE
*    LEVEL VIEW OF THE THREE LEVEL INTERNAL SCHEMA. IT IS
*    USED PRIMARILY FOR INCORPORATION IN THE QUERY PROGRAM
*    PQUERY WHICH ALLOWS USERS TO ACCESS THE DATA BASE
*    USING THE "QUILL" LANGUAGE.
 EXTERNAL SCHEMA NAME IS INVOICE-QUERY
 PERMIT ACCESS FOR UPDATE,RETRIEVE, CREATE, FORMAT.

 RECORD NAME IS ORDER-LINE=QUERY-RECORD(CUSTOMER-NAME/C 40,
 CUSTOMER-NO/C 6,CREDIT-LIMIT/N 8.2,INVOICE-NO/C 6,INVOICE-DATE/N 6,
 ORDER-ITEM/C 4,ORDER-PRICE/N 5.2,ORDER-QTY=QTY/N 6).
REVERT.EXTDDL.
/
```

```
PYRAMID,EXTDDL,I=TESTE4
EXTERNAL SCHEMA NAME IS ORDER-ITEMS
PERMIT ACCESS FOR UPDATE,RETRIEVE, CREATE, FORMAT.

RECORD NAME IS CUSTOMER(CUSTOMER-NAME/C 40,CUSTOMER-NO/C 6).

RECORD ORDER-LINE = ORDER(INVOICE-NO/C 6,ORDER-ITEM/C 4,
ORDER-PRICE/N 5.2,ORDER-QTY=QTY/N 6).
REVERT.EXTDDL.
```

```
PYRAMID,EXTDDL,I=TESTE5
EXTERNAL SCHEMA NAME IS PAYROLL
PERMIT ACCESS FOR UPDATE,RETRIEVE.

RECORD NAME IS DEPARTMENT(DEPT-NO/C 2,
DEPT-NAME/C 30).
RECORD NAME IS EMPLOYEE(EMP-NO/C 4,NAME/C 24,
SALARY/N 5).
REVERT.EXTDDL.
```

```
PYRAMID,EXTDDL,I=TESTE6
EXTERNAL SCHEMA NAME IS EMPLOYEE-LIST
PERMIT ACCESS FOR UPDATE,RETRIEVE.

RECORD NAME IS EMPLOYEE(DEPT-NO/C 2,EMP-NO/C 4,NAME/C 24,
SALARY/N 5).
EVERT.EXTDDL.
```

```
YRAMID,EXTDDL,I=TESTE7

*   THIS SET OF "PYRAMID" EXTERNAL SCHEMA DDL DESCRIBES
*    THE VIEW OF THE DATA BASE USED FOR ORDER-ENTRY.


EXTERNAL SCHEMA NAME IS TROUBLE
PERMIT ACCESS FOR UPDATE,RETRIEVE, CREATE, FORMAT.
RECORD NAME IS CUSTOMER(CUSTOMER-NAME/C 40,CUSTOMER-NO/C 6,
CREDIT-LIMIT/N 8.2,TOTAL-VALUE-ON-ORDER=TOT-VAL/N 8.2 ).

RECORD NAME IS INVOICE(INVOICE-NO/C 6,INVOICE-DATE/N 6).
RECORD ORDER-LINE = ORDER(ORDER-ITEM/C 4,ORDER-PRICE/N 5.2,
ORDER-QTY=QTY/N 6).

RECORD NAME IS PART(DESCRIPTION/C 40,PART-NO/C 4,
UNIT-PRICE/N 6.2,STOCK-IN-HAND/N 6).

RECORD NAME IS PURCHASE(PURCHASE-ORDER-NO/C 4,PURCHASE-DATE/N 6,
PURCHASE-QTY/N 6,PURCHASE-PRICE/N 6.2,PURCHASE-SUPPLIER-NO/C 4).

REVERT.EXTDDL.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CRCUST.
*
*   THIS PROGRAM IS USED TO SET UP AN EMPTY
*   CUSTOMER DATABASE.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   FUNCTION PIC X(10).
 01   RECORD-NAME PIC X(20).
 01   BUFFER PIC X(512).
 01   RESULT PIC 999.
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
      MOVE "NEW" TO FUNCTION.
      PERFORM CALL-DBMS.
      DISPLAY "DATA BASE CREATE RESULT = ", RESULT.
      MOVE "RELEASE" TO FUNCTION.
      PERFORM CALL-DBMS.
      DISPLAY "DATA BASE RELEASE RESULT = ", RESULT.
      STOP RUN.
 CALL-DBMS.
      CALL "DBMS" USING FUNCTION, RECORD-NAME,
                          BUFFER, RESULT.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ADDCUST.
*
*   THIS PROGRAM IS USED TO ADD CUSTOMERS TO
*   AN EXISTING CUSTOMER DATA BASE.
*
*   THE INVOICE AND ORDER-LINE RECORDS ON THE DATABASE
*   ARE NOT USED.
*
*   (NOTE THAT BY DEFAULT THE FIELD TOTAL-VALUE-ON-ORDER
*    IS SET TO ZERO ON ALL CREATED CUSTOMER RECORDS).
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   FINISHED PIC XXX.
 01   REPLY PIC XXX.
 01   FUNCTION PIC X(10).
 01   RECORD-NAME PIC X(20).
 01   CUSTOMER.
      02   CUSTOMER-NAME PIC X(40).
      02   CUSTOMER-NUMBER PIC X(6).
      02   CREDIT-LIMIT PIC 9(8).
      02   TOTAL-VALUE-ON-ORDER PIC 9(8).
      02   FILLER PIC X(450).
 01   RESULT PIC 999.
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
      MOVE "OLD" TO FUNCTION.
      PERFORM CALL-DBMS.
      DISPLAY "DATA BASE OPEN RESULT = ", RESULT.
      MOVE "NO" TO FINISHED.
      PERFORM ADD-CUSTOMER UNTIL FINISHED = "YES".
      MOVE "RELEASE" TO FUNCTION.
      PERFORM CALL-DBMS.
      DISPLAY "DATA BASE RELEASE RESULT = ", RESULT.
      STOP RUN.
 ADD-CUSTOMER.
      DISPLAY "ANY MORE CUSTOMERS TO BE ADDED".
      ACCEPT REPLY.
      IF REPLY = "YES"
           PERFORM GET-CUSTOMER-DETAILS
      ELSE MOVE "YES" TO FINISHED.
 GET-CUSTOMER-DETAILS.
      DISPLAY "ENTER CUSTOMER NUMBER @@@@@@".
      ACCEPT CUSTOMER-NUMBER.
      DISPLAY "ENTER CUSTOMER NAME".
      ACCEPT CUSTOMER-NAME.
      DISPLAY "ENTER CREDIT LIMIT ########".
      ACCEPT CREDIT-LIMIT.
      MOVE ZERO TO TOTAL-VALUE-ON-ORDER.
      MOVE "WRITE" TO FUNCTION.
      MOVE "CUSTOMER" TO RECORD-NAME.
      PERFORM CALL-DBMS.
      DISPLAY "WRITE  RESULT = ", RESULT.
 CALL-DBMS.
      CALL "DBMS" USING FUNCTION, RECORD-NAME,
                             CUSTOMER, RESULT.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. NEWITEM.
*
*   THIS PROGRAM IS A TAKE-ON PROGRAM TO SETUP THE
*   INITIAL PART ENTITIES ON THE PARTS FILE. THE
*   PURCHASE ENTITIES ARE NOT USED.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   FINISHED PIC XXX.
 01   REPLY PIC XXX.
 01   FUNCTION PIC X(10).
 01   RECORD-NAME PIC X(20).
 01   PART.
      02   DESCRIPTION PIC X(40).
      02   PART-NO PIC X(4).
      02   UNIT-PRICE PIC 9999V99.
      02   STOCK-IN-HAND PIC 9(6).
      02   FILLER PIC X(456).
 01   RESULT PIC 999.
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
      MOVE "NEW" TO FUNCTION.
      PERFORM CALL-DBMS.
      IF RESULT NOT = 0
           DISPLAY "ERROR ON OPENING DATA BASE"
           STOP RUN.
      MOVE "NO" TO FINISHED.
      PERFORM ADD-PART UNTIL FINISHED = "YES".
      MOVE "RELEASE" TO FUNCTION.
      PERFORM CALL-DBMS.
      IF RESULT NOT = 0
           DISPLAY "ERROR ON RELEASING DATA BASE".
      STOP RUN.
 ADD-PART.
      DISPLAY "ANY MORE PARTS TO BE ADDED".
      ACCEPT REPLY.
      IF REPLY = "YES"
           PERFORM GET-PART-DETAILS
      ELSE MOVE "YES" TO FINISHED.
 GET-PART-DETAILS.
      DISPLAY "ENTER PART NUMBER @@@@".
      ACCEPT PART-NO.
      DISPLAY "ENTER DESCRIPTION".
      ACCEPT DESCRIPTION.
      DISPLAY "ENTER INITIAL STOCK ######".
      ACCEPT STOCK-IN-HAND.
      MOVE ZERO TO UNIT-PRICE.
      MOVE "WRITE" TO FUNCTION.
      MOVE "PART" TO RECORD-NAME.
      PERFORM CALL-DBMS.
      IF RESULT NOT = ZERO
           DISPLAY "ERROR ON WRITING PART TO DATA BASE".
 CALL-DBMS.
      CALL "DBMS" USING FUNCTION, RECORD-NAME,
                           PART, RESULT.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ORDENT.
*
*   THIS PROGRAM IS THE ON-LINE ORDER-ENTRY PROGRAM.
*
*   IT ACCESSES TWO PHYSICAL DATABASES.
*
*              ORDERS     (CUSTOMER/INVOICE/ORDER)
*
*       AND      PARTS      (PART/PURCHASE)
*
*   THE ORDERS DATABASE HAS ORDER ENTITIES ADDED TO IT,
*   WHILE THE PARTS DATABASE HAS PART ENTITIES UPDATED
*   WITH THE QUANTITIES ACTUALLY ORDERED. (NOTE THAT
*   THE PURCHASE ENTITIES ARE NOT USED).
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  FINISHED PIC XXX.
 01  MORE-ITEMS PIC XXX.
 01  REPLY PIC XXX.
 01  WRITE-FUNCTION PIC X(10) VALUE IS "WRITE".
 01  READ-FUNCTION PIC X(10) VALUE IS "READ".
 01  OPEN-OLD-FUNCTION PIC X(10) VALUE IS "OLD".
 01  RELEASE-FUNCTION PIC X(10) VALUE IS "RELEASE".
 01  REWRITE-FUNCTION PIC X(10) VALUE IS "REWRITE".
 01  CUSTOMER-RECORD PIC X(20) VALUE IS "CUSTOMER".
 01  INVOICE-RECORD PIC X(20) VALUE IS "INVOICE".
 01  ORDER-RECORD PIC X(20) VALUE IS "ORDER".
 01  PART-RECORD PIC X(20) VALUE IS "PART".
 01  DUMMY-RECORD PIC X(20) VALUE IS SPACES.
 01  CUSTOMER.
     02  CUSTOMER-NAME PIC X(40).
     02  CUSTOMER-NUMBER PIC X(6).
     02  CREDIT-LIMIT PIC 9(8).
     02  TOTAL-VALUE-ON-ORDER PIC 9(8).
     02  FILLER PIC X(450).
 01  DUMMY-BUFFER REDEFINES CUSTOMER PIC X(512).
 01  INVOICE.
     02  INVOICE-NUMBER PIC X(6).
     02  INVOICE-DATE PIC 9(6).
     02  FILLER PIC X(500).
 01  ORDER-LINE.
     02  ORDER-ITEM PIC XXXX.
     02  ORDER-PRICE PIC 999.99.
     02  ORDER-QTY PIC 9(6).
     02  FILLER PIC X(497).
 01  PART.
     02  DESCRIPTION PIC X(40).
     02  PART-NO PIC X(4).
     02  UNIT-PRICE PIC 9999V99.
     02  STOCK-IN-HAND PIC 9(6).
     02  FILLER PIC X(456).
 01  RESULT PIC 999.
 PROCEDURE DIVISION.
 MAIN-PARAGRAPH.
     CALL "DBMS" USING OPEN-OLD-FUNCTION, DUMMY-RECORD,
                          DUMMY-BUFFER, RESULT.
```

```
    IF RESULT NOT = ZERO
         DISPLAY "ERROR ON OPENING DATA BASE"
         STOP RUN.
    MOVE "NO" TO FINISHED.
    PERFORM PROCESS-CUSTOMER UNTIL FINISHED = "YES".
    CALL "DBMS" USING RELEASE-FUNCTION, DUMMY-RECORD,
                           DUMMY-BUFFER, RESULT.
    IF RESULT NOT = ZERO
         DISPLAY "ERROR ON RELEASING DATA BASE".
    STOP RUN.
PROCESS-CUSTOMER.
    DISPLAY "ANY MORE ORDERS".
    ACCEPT REPLY.
    IF REPLY = "YES"
         PERFORM GET-CUSTOMER-DETAILS
    ELSE MOVE "YES" TO FINISHED.
GET-CUSTOMER-DETAILS.
    DISPLAY "ENTER CUSTOMER NUMBER @@@@@@".
    ACCEPT CUSTOMER-NUMBER.
    CALL "DBMS" USING READ-FUNCTION, CUSTOMER-RECORD,
                           CUSTOMER, RESULT.
    IF RESULT NOT = ZERO
         DISPLAY "ERROR ON READING CUSTOMER RECORD".
    DISPLAY "CUSTOMER NAME = ", CUSTOMER-NAME.
    DISPLAY "CORRECT CUSTOMER".
    ACCEPT REPLY.
    IF REPLY = "YES" PERFORM PROCESS-INVOICE.
PROCESS-INVOICE.
    DISPLAY "ENTER INVOICE NUMBER ######".
    ACCEPT INVOICE-NUMBER.
    DISPLAY "ENTER INVOICE DATE YYMMDD".
    ACCEPT INVOICE-DATE.
    CALL "DBMS" USING WRITE-FUNCTION, INVOICE-RECORD,
                           INVOICE, RESULT.
    DISPLAY "WRITE RESULT = ", RESULT.
    MOVE "YES" TO MORE-ITEMS.
    PERFORM PROCESS-ITEM UNTIL MORE-ITEMS = "NO".
PROCESS-ITEM.
    DISPLAY "ANY MORE ITEMS".
    ACCEPT REPLY.
    IF REPLY = "YES"
         PERFORM GET-ITEM-DETAILS
    ELSE MOVE "NO" TO MORE-ITEMS.
GET-ITEM-DETAILS.
    DISPLAY "ENTER ITEM NUMBER ####".
    ACCEPT ORDER-ITEM.
    MOVE ORDER-ITEM TO PART-NO.
    CALL "DBMS" USING READ-FUNCTION, PART-RECORD,
                           PART, RESULT.
    IF RESULT NOT = 0
         DISPLAY "NO SUCH PART"
    ELSE DISPLAY "DESCRITPTION = ", DESCRIPTION
         DISPLAY "CORRECT ITEM ?"
         ACCEPT REPLY
         IF REPLY = "YES"
              PERFORM GET-ITEM-QUANTITY.
GET-ITEM-QUANTITY.
    DISPLAY "ENTER QUANTITY ###".
    ACCEPT ORDER-QTY.
    SUBTRACT ORDER-QTY FROM STOCK-IN-HAND.
    IF STOCK-IN-HAND < ZERO
```

```
           DISPLAY "NOT ENOUGH STOCK"
      ELSE PERFORM RECORD-ORDER-DETAILS.
 RECORD-ORDER-DETAILS.
      CALL "DBMS" USING REWRITE-FUNCTION, PART-RECORD,
                        PART, RESULT.
      CALL "DBMS" USING WRITE-FUNCTION, ORDER-RECORD,
                        ORDER-LINE, RESULT.
      IF RESULT NOT = ZERO
           DISPLAY "ERROR ON WRITING ORDER RECORD".
/
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. IVPRINT.
*
*   THIS PROGRAM IS THE INVOICE PRINT PROGRAM.
*
*   IT ACCESSES TWO PHYSICAL DATABASES.
*
*              ORDERS    (CUSTOMER/INVOICE/ORDER)
*
*       AND      PARTS     (PART/PURCHASE)
*
*   THE ORDERS DATABASE HAS ORDER ENTITIES ADDED TO IT,
*   WHILE THE PARTS DATABASE HAS PART ENTITIES UPDATED
*   WITH THE QUANTITIES ACTUALLY ORDERED. (NOTE THAT
*   THE PURCHASE ENTITIES ARE NOT USED).
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. CYBER.
 OBJECT-COMPUTER. CYBER.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT INVOICES ASSIGN TO "OUTPUT".
 DATA DIVISION.
 FILE SECTION.
 FD  INVOICES LABEL RECORDS OMITTED.
 01  INVOICE-LINE.
     02  FILLER PIC X.
     02  FILLER PIC X(132).
 WORKING-STORAGE SECTION.
 01  MORE-CUSTOMERS PIC XXX.
 01  MORE-INVOICES PIC XXX.
 01  MORE-ORDER-LINES PIC XXX.
 01  REPLY PIC XXX.
 01  WRITE-FUNCTION PIC X(10) VALUE IS "WRITE".
 01  READ-FUNCTION PIC X(10) VALUE IS "READ".
 01  NEXT-FUNCTION PIC X(10) VALUE IS "NEXT".
 01  OPEN-OLD-FUNCTION PIC X(10) VALUE IS "OLD".
 01  RELEASE-FUNCTION PIC X(10) VALUE IS "RELEASE".
 01  REWRITE-FUNCTION PIC X(10) VALUE IS "REWRITE".
 01  CUSTOMER-RECORD PIC X(20) VALUE IS "CUSTOMER".
 01  INVOICE-RECORD PIC X(20) VALUE IS "INVOICE".
 01  ORDER-RECORD PIC X(20) VALUE IS "ORDER".
 01  PART-RECORD PIC X(20) VALUE IS "PART".
 01  DUMMY-RECORD PIC X(20) VALUE IS SPACES.
 01  CUSTOMER.
     02  CUSTOMER-NAME PIC X(40).
     02  CUSTOMER-NUMBER PIC X(6).
     02  CREDIT-LIMIT PIC 9(8).
     02  TOTAL-VALUE-ON-ORDER PIC 9(8).
     02  FILLER PIC X(450).
 01  DUMMY-BUFFER REDEFINES CUSTOMER PIC X(512).
 01  INVOICE.
     02  INVOICE-NUMBER PIC X(6).
     02  INVOICE-DATE PIC 9(6).
     02  FILLER PIC X(500).
 01  ORDER-LINE.
     02  ORDER-ITEM PIC XXXX.
     02  ORDER-PRICE PIC 999.99.
     02  ORDER-QTY PIC 9(6).
     02  FILLER PIC X(497).
 01  PART.
```

```
        02  DESCRIPTION PIC X(40).
        02  PART-NO PIC X(4).
        02  UNIT-PRICE PIC 9999V99.
        02  STOCK-IN-HAND PIC 9(6).
        02  FILLER PIC X(456).
    01  RESULT PIC 999.
    01  HEADING-LINE-ONE.
        02  FILLER PIC X.
        02  FILLER PIC X(4) VALUE "CUST".
        02  FILLER PIC X(6) VALUE SPACES.
        02  FILLER PIC X(8) VALUE "CUSTOMER".
        02  FILLER PIC X(20) VALUE SPACES.
        02  FILLER PIC X(7) VALUE "INVOICE".
        02  FILLER PIC X(10).
    01  HEADING-LINE-TWO.
        02  FILLER PIC X.
        02  FILLER PIC X(4) VALUE " NO ".
        02  FILLER PIC X(6) VALUE SPACES.
        02  FILLER PIC X(8) VALUE "  NAME  ".
        02  FILLER PIC X(20) VALUE SPACES.
        02  FILLER PIC X(7) VALUE "NUMBER ".
        02  FILLER PIC X(10).
    01  HEADING-LINE-THREE.
        02  FILLER PIC X.
        02  CUSTOMER-NUMBER-OUT PIC X(6).
        02  FILLER PIC XX VALUE SPACES.
        02  CUSTOMER-NAME-OUT PIC X(40).
        02  FILLER PIC XXXX VALUE SPACES.
        02  INVOICE-NUMBER-OUT PIC X(6).
        02  FILLER PIC X(10).
    01  HEADING-LINE-FOUR.
        02  FILLER PIC X.
        02  FILLER PIC X(4) VALUE "ITEM".
        02  FILLER PIC XX VALUE SPACES.
        02  FILLER PIC X(11) VALUE "DESCRIPTION".
        02  FILLER PIC XXXX VALUE SPACES.
        02  FILLER PIC X(5) VALUE "ORDER".
        02  FILLER PIC XXXX VALUE SPACES.
        02  FILLER PIC XXXX VALUE "UNIT".
    01  HEADING-LINE-FIVE.
        02  FILLER PIC X.
        02  FILLER PIC XXXX VALUE " NO ".
        02  FILLER PIC X(15) VALUE SPACES.
        02  FILLER PIC X(5) VALUE " QTY ".
        02  FILLER PIC XXXX VALUE SPACES.
        02  FILLER PIC X(5) VALUE " NO. ".
    01  DETAIL-LINE.
        02  FILLER PIC X.
        02  PART-NUMBER-OUT PIC X(4).
        02  FILLER PIC XX VALUE SPACES.
        02  DESCRIPTION-OUT PIC X(40).
        02  FILLER PIC XXXX VALUE SPACES.
        02  ORDER-QTY-OUT PIC Z(5)9.
        02  FILLER PIC XXXX VALUE SPACES.
        02  UNIT-PRICE-OUT PIC ZZZ9.99.
    PROCEDURE DIVISION.
    MAIN-PARAGRAPH.
        OPEN OUTPUT INVOICES.
        CALL "DBMS" USING OPEN-OLD-FUNCTION, DUMMY-RECORD,
                           DUMMY-BUFFER, RESULT.
        IF RESULT NOT = ZERO
```

```
            DISPLAY "ERROR ON OPENING DATA BASE"
            STOP RUN.
       MOVE "YES" TO MORE-CUSTOMERS.
       PERFORM PROCESS-CUSTOMER UNTIL MORE-CUSTOMERS = "NO".
       CALL "DBMS" USING RELEASE-FUNCTION, DUMMY-RECORD,
                             DUMMY-BUFFER, RESULT.
       IF RESULT NOT = ZERO
            DISPLAY "ERROR ON RELEASING DATA BASE".
       CLOSE INVOICES.
       STOP RUN.
   PROCESS-CUSTOMER.
       CALL "DBMS" USING NEXT-FUNCTION, CUSTOMER-RECORD,
                            CUSTOMER, RESULT.
       IF RESULT = 111
            MOVE "NO" TO MORE-CUSTOMERS
       ELSE IF RESULT = ZERO
                 PERFORM PROCESS-INVOICES-FOR-CUSTOMER
            ELSE DISPLAY "NEXT CUSTOMER ERROR ", RESULT
                 STOP RUN.
   PROCESS-INVOICES-FOR-CUSTOMER.
       DISPLAY " ".
       MOVE CUSTOMER-NUMBER TO CUSTOMER-NUMBER-OUT.
       MOVE CUSTOMER-NAME TO CUSTOMER-NAME-OUT.
       MOVE "YES" TO MORE-INVOICES.
       PERFORM PROCESS-INVOICE UNTIL MORE-INVOICES = "NO".
   PROCESS-INVOICE.
       CALL "DBMS" USING NEXT-FUNCTION, INVOICE-RECORD,
                            INVOICE, RESULT.
       IF RESULT = 111
            MOVE "NO" TO MORE-INVOICES
       ELSE IF RESULT = ZERO
                 PERFORM PROCESS-INVOICE-ITEMS
            ELSE DISPLAY "NEXT INVOICE ERROR ", RESULT
                 STOP RUN.
   PROCESS-INVOICE-ITEMS.
       MOVE INVOICE-NUMBER TO INVOICE-NUMBER-OUT.
       WRITE INVOICE-LINE FROM HEADING-LINE-ONE.
       WRITE INVOICE-LINE FROM HEADING-LINE-TWO.
       WRITE INVOICE-LINE FROM HEADING-LINE-THREE.
       WRITE INVOICE-LINE FROM HEADING-LINE-FOUR
                             AFTER ADVANCING 2 LINES.
       WRITE INVOICE-LINE FROM HEADING-LINE-FIVE.
       MOVE "YES" TO MORE-ORDER-LINES.
       PERFORM PROCESS-ORDER-LINE UNTIL MORE-ORDER-LINES = "NO".
   PROCESS-ORDER-LINE.
       CALL "DBMS" USING NEXT-FUNCTION, ORDER-RECORD,
                            ORDER-LINE, RESULT.
       IF RESULT = 111
            MOVE "NO" TO MORE-ORDER-LINES
       ELSE IF RESULT = ZERO
                  PERFORM PRINT-ORDER-DETAILS
            ELSE DISPLAY "NEXT ORDER ERROR ", RESULT
                 STOP RUN.
   PRINT-ORDER-DETAILS.
       MOVE PART-NO TO PART-NUMBER-OUT.
       MOVE ORDER-QTY TO ORDER-QTY-OUT.
       MOVE ORDER-ITEM TO PART-NO.
       CALL "DBMS" USING READ-FUNCTION, PART-RECORD,
                            PART, RESULT.
       IF RESULT = 23
            MOVE ALL "*" TO DESCRIPTION
```

```
ELSE IF RESULT NOT = ZERO
        DISPLAY "READ PART ", ORDER-ITEM,
                " ERROR ", RESULT
        STOP RUN.
MOVE DESCRIPTION TO DESCRIPTION-OUT.
MOVE UNIT-PRICE TO UNIT-PRICE-OUT.
WRITE INVOICE-LINE FROM DETAIL-LINE.
```

```
PYRAMID, QUERY, I=TESTPQ1, D=ORDERS

 *   THIS QUERY PRINTS OUT THE CUSTOMER NAME AND
 *   QUANTITY ON ORDER FOR ALL CURRENT ORDERS
 *   FOR ITEM 7979.
WHERE ORDER-ITEM = 7979 PRINT ORDER-QTY, CUSTOMER-NAME.

 020    JONES
 100    GODFREY
REVERT. QUERY.


PYRAMID, QUERY, I=TESTPQ2, D=ORDERS

 *    THIS QUERY PRINTS OUT THE ITEM NUMBERS
 *    AND QUANTITIES FOR INVOICE 121212

 *    HOWEVER THE FIELD INVOICE-NO HAS BEEN CALLED INVOICE-NUMBER

WHERE INVOICE-NUMBER = 121212 PRINT ORDER-ITEM, ORDER-QTY.
NO SUCH FIELD AS INVOICE-NUMBER
SEARCH ABANDONED
FIELD NAME


SOURCE-REJECTED

REVERT. QUERY.


PYRAMID, QUERY, I=TESTPQ3, D=ORDERS

 *    THIS QUERY PRINTS OUT THE ITEM NUMBERS
 *    AND QUANTITIES FOR INVOICE 121212

WHERE INVOICE-NO = 121212 PRINT ORDER-ITEM, ORDER-QTY;
HEADING "ITEM    QTY" HEADING "----    ---" ON LINE 2
HEADING " " ON LINE 3.

ITEM   QTY
----   ---

6767   015
7979   100
REVERT. QUERY.
```

Mapping Code Example 1

The following COBOL code was generated by the PYRAMID mapping code generator using the source code

```
INTERNAL SCHEMA NAME IS DOUBLE.
EXTERNAL SCHEMA NAME IS TROUBLE.
```

The generated code is used by the ORDENT, IVPRINT, ADDCUST AND CRCUST programs.

```
A0001 IDENTIFICATION DIVISION.
A0002 PROGRAM-ID. T[illegible].
A0075*
A0076*         EXTERNAL SCHEMA NAME IS TROUBLE
A0077*
A0285*
A0286*         INTERNAL SCHEMA NAME IS TRUBL*
A0287*
A0003 ENVIRONMENT DIVISION.
A0004 CONFIGURATION SECTION.
A0005 SOURCE-COMPUTER. CYBER.
A0006 OBJECT-COMPUTER. CYBER.
B0007 INPUT-OUTPUT SECTION.
B0008 FILE-CONTROL.
BB0288     SELECT INTERNAL-CUSTOMERS
BB0289             ASSIGN TO "ORDERS"
BB0290     ORGANIZATION IS INDEXED
BB0291         ACCESS MODE IS DYNAMIC
BB0292     RECORD KEY IS DBMS-KEY-CUSTOMERS
BB0688     FILE STATUS IS FILE-STATUS.
BB0689     SELECT INTERNAL-INVENTORY
BB0690             ASSIGN TO "PARTS"
BB0691     ORGANIZATION IS INDEXED
BB0692         ACCESS MODE IS DYNAMIC
BB0693     RECORD KEY IS DBMS-KEY-INVENTORY
BB0958     FILE STATUS IS FILE-STATUS.
CA0009 DATA DIVISION.
CA0010 FILE SECTION.
CB0293 FD  INTERNAL-CUSTOMERS
CB0294     LABEL RECORDS OMITTED.
CB0295 01  DBMS-RCD-CUSTOMERS.
CB0296     02  DBMS-KEY-CUSTOMERS.
CB0297         03 DBMS-CUSTOMER-NO              PICTURE IS X(6).
CB0298         03 DBMS-INVOICE-NO               PICTURE IS X(6).
CB0299         03 DBMS-ORDER-ITEM               PICTURE IS X(4).
CB0300     02  ENTITY-CODE PICTURE IS 99.
CB0431 01  DBMS-REC-CUSTOMER PICTURE IS X(72).
CB0559 01  DBMS-REC-INVOICE PICTURE IS X(22).
CB0687 01  DBMS-REC-ORDER-LINE PICTURE IS X(27).
CB0694 FD  INTERNAL-INVENTORY
CB0695     LABEL RECORDS OMITTED.
CB0696 01  DBMS-RCD-INVENTORY.
CB0697     02  DBMS-KEY-INVENTORY.
CB0698         03 DBMS-PART-NO                  PICTURE IS X(4).
CB0699         03 DBMS-PURCHASE-ORDER-NO        PICTURE IS X(4).
CB0700     02  ENTITY-CODE PICTURE IS 99.
CB0826 01  DBMS-REC-PART PICTURE IS X(60).
CB0957 01  DBMS-REC-PURCHASE PICTURE IS X(30).
CC0011 WORKING-STORAGE SECTION.
CC0012 01  FILE-STATUS PICTURE IS XX.
CC0013 01  DATA-BASE-OPEN-FLAG PIC X(3)
CC0014             VALUE IS "NO".
CC0015 01  SEARCH-FLAG PICTURE IS XXX.
CC0016 01  CURRENT-ENTITY-CODE PIC 99.
CC0017 01  SAME-OWNER              PICTURE IS XXX.
CG0081 01  DBMS-CUR-CUSTOMER PICTURE IS XXX VALUE IS "NO".
CG0126 01  DBMS-CUR-INVOICE PICTURE IS XXX VALUE IS "NO".
CG0153 01  DBMS-CUR-ORDER-LINE PICTURE IS XXX VALUE IS "NO".
CG0189 01  DBMS-CUR-PART PICTURE IS XXX VALUE IS "NO".
CG0234 01  DBMS-CUR-PURCHASE PICTURE IS XXX VALUE IS "NO".
CJ0082 01  DBMS-EXT-CUSTOMER.
```

```
J0087       02 DBMS-EXT-CUSTOMER-NAME PICTURE IS X(40).
J0096       02 DBMS-EXT-CUSTOMER-NO PICTURE IS X(6).
J0105       02 DBMS-EXT-CREDIT-LIMIT PICTURE IS 9(8).
J0114       02 DBMS-EXT-TOTAL-VALUE-ON-ORDER PICTURE IS 9(8).
J0127 01    DBMS-EXT-INVOICE.
J0132       02 DBMS-EXT-INVOICE-NO PICTURE IS X(6).
J0141       02 DBMS-EXT-INVOICE-DATE PICTURE IS 9(6).
J0154 01    DBMS-EXT-ORDER-LINE.
J0159       02 DBMS-EXT-ORDER-ITEM PICTURE IS X(4).
J0168       02 DBMS-EXT-ORDER-PRICE PICTURE IS 9(5).
J0177       02 DBMS-EXT-ORDER-QTY PICTURE IS 9(6).
J0190 01    DBMS-EXT-PART.
CJ0195      02 DBMS-EXT-DESCRIPTION PICTURE IS X(40).
CJ0204      02 DBMS-EXT-PART-NO PICTURE IS X(4).
CJ0213      02 DBMS-EXT-UNIT-PRICE PICTURE IS 9(6).
CJ0222      02 DBMS-EXT-STOCK-IN-HAND PICTURE IS 9(6).
CJ0235 01   DBMS-EXT-PURCHASE.
CJ0240      02 DBMS-EXT-PURCHASE-ORDER-NO PICTURE IS X(4).
CJ0249      02 DBMS-EXT-PURCHASE-DATE PICTURE IS 9(6).
CJ0258      02 DBMS-EXT-PURCHASE-QTY PICTURE IS 9(6).
CJ0267      02 DBMS-EXT-PURCHASE-PRICE PICTURE IS 9(6).
CJ0276      02 DBMS-EXT-PURCHASE-SUPPLIER-NO PICTURE IS X(4).
CK0415 01   DBMS-INT-CUSTOMER.
CK0416      02  DBMS-INT-CUSTOMER-NO PICTURE IS X(6).
CK0417      02  DBMS-KEY-001 PICTURE IS X(6).
CK0418      02  DBMS-KEY-002 PICTURE IS X(4).
CK0419      02  FILLER PICTURE IS 99.
CK0420      02 DBMS-INT-CUSTOMER-NAME PICTURE IS X(30).
CK0422      02 DBMS-INT-CREDIT-LIMIT PICTURE IS 9(8).
CK0424      02 DBMS-INT-BALANCE PICTURE IS 9(10).
CK0426      02 DBMS-INT-TOTAL-VALUE-ON-ORDER PICTURE IS 9(8).
CK0549 01   DBMS-INT-INVOICE.
CK0550      02  DBMS-KEY-003 PICTURE IS X(6).
CK0551      02  DBMS-INT-INVOICE-NO PICTURE IS X(6).
CK0552      02  DBMS-KEY-004 PICTURE IS X(4).
CK0553      02  FILLER PICTURE IS 99.
CK0554      02 DBMS-INT-INVOICE-DATE PICTURE IS 9(6).
CK0675 01   DBMS-INT-ORDER-LINE.
CK0676      02  DBMS-KEY-005 PICTURE IS X(6).
CK0677      02  DBMS-KEY-006 PICTURE IS X(6).
CK0678      02  DBMS-INT-ORDER-ITEM PICTURE IS X(4).
CK0679      02  FILLER PICTURE IS 99.
CK0680      02 DBMS-INT-ORDER-QTY PICTURE IS 9(6).
CK0682      02 DBMS-INT-ORDER-PRICE PICTURE IS 9(5).
CK0813 01   DBMS-INT-PART.
CK0814      02  DBMS-INT-PART-NO PICTURE IS X(4).
CK0815      02  DBMS-KEY-007 PICTURE IS X(4).
CK0816      02  FILLER PICTURE IS 99.
CK0817      02 DBMS-INT-DESCRIPTION PICTURE IS X(40).
CK0819      02 DBMS-INT-UNIT-PRICE PICTURE IS 9(6).
CK0821      02 DBMS-INT-STOCK-IN-HAND PICTURE IS 9(6).
CK0942 01   DBMS-INT-PURCHASE.
CK0943      02  DBMS-KEY-008 PICTURE IS X(4).
CK0944      02  DBMS-INT-PURCHASE-ORDER-NO PICTURE IS X(4).
CK0945      02  FILLER PICTURE IS 99.
CK0946      02 DBMS-INT-PURCHASE-QTY PICTURE IS 9(6).
CK0948      02 DBMS-INT-PURCHASE-DATE PICTURE IS 9(6).
CK0950      02 DBMS-INT-PURCHASE-PRICE PICTURE IS 9(6).
CK0952      02 DBMS-INT-PURCHASE-SUPPLIER-NO PICTURE IS X(4).
CL0085 01   DBMS-FMT-CUSTOMER.
CL0086      02  DBMS-NOI-CUSTOMER PICTURE IS 99 VALUE IS 04.
```

```
0088       02  FILLER PIC X(20) VALUE IS "CUSTOMER-NAME".
0089       02  FILLER PIC X VALUE IS "C".
0090       02  FILLER PIC 9999 VALUE IS 0001.
0091       02  FILLER PIC S999V99 VALUE IS 040.
0097       02  FILLER PIC X(20) VALUE IS "CUSTOMER-NO".
L0098      02  FILLER PIC X VALUE IS "C".
L0099      02  FILLER PIC 9999 VALUE IS 0041.
L0100      02  FILLER PIC S999V99 VALUE IS 006.
L0106      02  FILLER PIC X(20) VALUE IS "CREDIT-LIMIT".
L0107      02  FILLER PIC X VALUE IS "N".
L0108      02  FILLER PIC 9999 VALUE IS 0047.
L0109      02  FILLER PIC S999V99 VALUE IS 008.
L0115      02  FILLER PIC X(20) VALUE IS "TOTAL-VALUE-ON-ORDER".
L0116      02  FILLER PIC X VALUE IS "N".
L0117      02  FILLER PIC 9999 VALUE IS 0055.
L0118      02  FILLER PIC S999V99 VALUE IS 008.
L0130 01   DBMS-FMT-INVOICE.
L0131      02  DBMS-NOI-INVOICE PICTURE IS 99 VALUE IS 02.
L0133      02  FILLER PIC X(20) VALUE IS "INVOICE-NO".
L0134      02  FILLER PIC X VALUE IS "C".
L0135      02  FILLER PIC 9999 VALUE IS 0001.
L0136      02  FILLER PIC S999V99 VALUE IS 006.
L0142      02  FILLER PIC X(20) VALUE IS "INVOICE-DATE".
L0143      02  FILLER PIC X VALUE IS "N".
L0144      02  FILLER PIC 9999 VALUE IS 0007.
L0145      02  FILLER PIC S999V99 VALUE IS 006.
CL0157 01  DBMS-FMT-ORDER-LINE.
CL0158     02  DBMS-NOI-ORDER-LINE PICTURE IS 99 VALUE IS 03.
CL0160     02  FILLER PIC X(20) VALUE IS "ORDER-ITEM".
CL0161     02  FILLER PIC X VALUE IS "C".
CL0162     02  FILLER PIC 9999 VALUE IS 0001.
CL0163     02  FILLER PIC S999V99 VALUE IS 004.
CL0169     02  FILLER PIC X(20) VALUE IS "ORDER-PRICE".
CL0170     02  FILLER PIC X VALUE IS "N".
CL0171     02  FILLER PIC 9999 VALUE IS 0005.
CL0172     02  FILLER PIC S999V99 VALUE IS 005.
CL0178     02  FILLER PIC X(20) VALUE IS "ORDER-QTY".
CL0179     02  FILLER PIC X VALUE IS "N".
CL0180     02  FILLER PIC 9999 VALUE IS 0010.
CL0181     02  FILLER PIC S999V99 VALUE IS 006.
CL0193 01  DBMS-FMT-PART.
CL0194     02  DBMS-NOI-PART PICTURE IS 99 VALUE IS 04.
CL0196     02  FILLER PIC X(20) VALUE IS "DESCRIPTION".
CL0197     02  FILLER PIC X VALUE IS "C".
CL0198     02  FILLER PIC 9999 VALUE IS 0001.
CL0199     02  FILLER PIC S999V99 VALUE IS 040.
CL0205     02  FILLER PIC X(20) VALUE IS "PART-NO".
CL0206     02  FILLER PIC X VALUE IS "C".
CL0207     02  FILLER PIC 9999 VALUE IS 0041.
CL0208     02  FILLER PIC S999V99 VALUE IS 004.
CL0214     02  FILLER PIC X(20) VALUE IS "UNIT-PRICE".
CL0215     02  FILLER PIC X VALUE IS "N".
CL0216     02  FILLER PIC 9999 VALUE IS 0045.
CL0217     02  FILLER PIC S999V99 VALUE IS 006.
CL0223     02  FILLER PIC X(20) VALUE IS "STOCK-IN-HAND".
CL0224     02  FILLER PIC X VALUE IS "N".
CL0225     02  FILLER PIC 9999 VALUE IS 0051.
CL0226     02  FILLER PIC S999V99 VALUE IS 006.
CL0238 01  DBMS-FMT-PURCHASE.
CL0239     02  DBMS-NOI-PURCHASE PICTURE IS 99 VALUE IS 05.
CL0241     02  FILLER PIC X(20) VALUE IS "PURCHASE-ORDER-NO".
```

```
CL0242        02   FILLER PIC X VALUE IS "C".
CL0243        02   FILLER PIC 9999 VALUE IS 0001.
CL0244        02   FILLER PIC S999V99 VALUE IS 004.
CL0250        02   FILLER PIC X(20) VALUE IS "PURCHASE-DATE".
CL0251        02   FILLER PIC X VALUE IS "N".
CL0252        02   FILLER PIC 9999 VALUE IS 0005.
CL0253        02   FILLER PIC S999V99 VALUE IS 006.
CL0259        02   FILLER PIC X(20) VALUE IS "PURCHASE-QTY".
CL0260        02   FILLER PIC X VALUE IS "N".
CL0261        02   FILLER PIC 9999 VALUE IS 0011.
CL0262        02   FILLER PIC S999V99 VALUE IS 006.
CL0268        02   FILLER PIC X(20) VALUE IS "PURCHASE-PRICE".
CL0269        02   FILLER PIC X VALUE IS "N".
CL0270        02   FILLER PIC 9999 VALUE IS 0017.
CL0271        02   FILLER PIC S999V99 VALUE IS 006.
CL0277        02   FILLER PIC X(20) VALUE IS "PURCHASE-SUPPLIER-NO".
CL0278        02   FILLER PIC X VALUE IS "C".
CL0279        02   FILLER PIC 9999 VALUE IS 0023.
CL0280        02   FILLER PIC S999V99 VALUE IS 004.
CS0359 01   CUR-INT-CUSTOMER PICTURE IS XXX VALUE IS "NO".
CS0531 01   CUR-INT-INVOICE PICTURE IS XXX VALUE IS "NO".
CS0657 01   CUR-INT-ORDER-LINE PICTURE IS XXX VALUE IS "NO".
CS0797 01   CUR-INT-PART PICTURE IS XXX VALUE IS "NO".
CS0924 01   CUR-INT-PURCHASE PICTURE IS XXX VALUE IS "NO".
CX0304 01   BUFFER-CUSTOMERS PICTURE IS X(5).
CX0704 01   BUFFER-INVENTORY PICTURE IS X(5).
CZ0018 LINKAGE SECTION.
CZ0019 01   FUNCTION PIC X(10).
CZ0020 01   THE-RECORD-NAME PIC X(20).
CZ0021 01   RESULT PIC 999.
CZ0022 01   UWA PIC X(512).
DA0023 PROCEDURE DIVISION USING FUNCTION,
DA0024      THE-RECORD-NAME, UWA,
DA0025      RESULT.
EA0026 INITIAL-PARAGRAPH.
EA0027      MOVE ZERO TO RESULT.
EA0028      IF FUNCTION = "NEW "
EA0029          PERFORM NEW-DATA-BASE
EA0030      ELSE IF FUNCTION = "OLD "
EA0031          PERFORM OLD-DATA-BASE
EA0032          ELSE IF FUNCTION = "RELEASE "
EA0033              PERFORM RELEASE-DATA-BASE
EA0034              ELSE PERFORM BRANCH-ON-RECORD-NAME.
EA0035 FINAL-PARAGRAPH.
EA0036      EXIT PROGRAM.
FA0037 BRANCH-ON-RECORD-NAME.
FA0078          IF THE-RECORD-NAME = "CUSTOMER"
FA0079              PERFORM USE-CUSTOMER
FA0080      ELSE
FA0123          IF THE-RECORD-NAME = "INVOICE"
FA0124              PERFORM USE-INVOICE
FA0125      ELSE
FA0150          IF THE-RECORD-NAME = "ORDER"
FA0151              PERFORM USE-ORDER-LINE
FA0152      ELSE
FA0186          IF THE-RECORD-NAME = "PART"
FA0187              PERFORM USE-PART
FA0188      ELSE
FA0231          IF THE-RECORD-NAME = "PURCHASE"
FA0232              PERFORM USE-PURCHASE
FA0233      ELSE
```

```
HA0089          PERFORM NO-SUCH-RECORD.
HA0093 FILL-INT-CUSTOMER.
HA0094     MOVE DBMS-EXT-CUSTOMER-NAME
HA0095       TO DBMS-INT-CUSTOMER-NAME.
HA0103     MOVE DBMS-EXT-CUSTOMER-NO
HA0104       TO DBMS-INT-CUSTOMER-NO.
HA0112     MOVE DBMS-EXT-CREDIT-LIMIT
HA0113       TO DBMS-INT-CREDIT-LIMIT.
HA0121     MOVE DBMS-EXT-TOTAL-VALUE-ON-ORDER
HA0122       TO DBMS-INT-TOTAL-VALUE-ON-ORDER.
HA0128 FILL-INT-INVOICE.
HA0139     MOVE DBMS-EXT-INVOICE-NO
HA0140       TO DBMS-INT-INVOICE-NO.
HA0148     MOVE DBMS-EXT-INVOICE-DATE
HA0149       TO DBMS-INT-INVOICE-DATE.
HA0155 FILL-INT-ORDER-LINE.
HA0166     MOVE DBMS-EXT-ORDER-ITEM
HA0167       TO DBMS-INT-ORDER-ITEM.
HA0175     MOVE DBMS-EXT-ORDER-PRICE
HA0176       TO DBMS-INT-ORDER-PRICE.
HA0184     MOVE DBMS-EXT-ORDER-QTY
HA0185       TO DBMS-INT-ORDER-QTY.
HA0191 FILL-INT-PART.
HA0202     MOVE DBMS-EXT-DESCRIPTION
HA0203       TO DBMS-INT-DESCRIPTION.
HA0211     MOVE DBMS-EXT-PART-NO
HA0212       TO DBMS-INT-PART-NO.
HA0220     MOVE DBMS-EXT-UNIT-PRICE
HA0221       TO DBMS-INT-UNIT-PRICE.
HA0229     MOVE DBMS-EXT-STOCK-IN-HAND
HA0230       TO DBMS-INT-STOCK-IN-HAND.
HA0236 FILL-INT-PURCHASE.
HA0247     MOVE DBMS-EXT-PURCHASE-ORDER-NO
HA0248       TO DBMS-INT-PURCHASE-ORDER-NO.
HA0256     MOVE DBMS-EXT-PURCHASE-DATE
HA0257       TO DBMS-INT-PURCHASE-DATE.
HA0265     MOVE DBMS-EXT-PURCHASE-QTY
HA0266       TO DBMS-INT-PURCHASE-QTY.
HA0274     MOVE DBMS-EXT-PURCHASE-PRICE
HA0275       TO DBMS-INT-PURCHASE-PRICE.
HA0283     MOVE DBMS-EXT-PURCHASE-SUPPLIER-NO
HA0284       TO DBMS-INT-PURCHASE-SUPPLIER-NO.
HB0084 FILL-EXT-CUSTOMER.
HB0092     MOVE DBMS-INT-CUSTOMER-NAME
HB0093       TO DBMS-EXT-CUSTOMER-NAME.
HB0101     MOVE DBMS-INT-CUSTOMER-NO
HB0102       TO DBMS-EXT-CUSTOMER-NO.
HB0110     MOVE DBMS-INT-CREDIT-LIMIT
HB0111       TO DBMS-EXT-CREDIT-LIMIT.
HB0119     MOVE DBMS-INT-TOTAL-VALUE-ON-ORDER
HB0120       TO DBMS-EXT-TOTAL-VALUE-ON-ORDER.
HB0129 FILL-EXT-INVOICE.
HB0137     MOVE DBMS-INT-INVOICE-NO
HB0138       TO DBMS-EXT-INVOICE-NO.
HB0146     MOVE DBMS-INT-INVOICE-DATE
HB0147       TO DBMS-EXT-INVOICE-DATE.
HB0156 FILL-EXT-ORDER-LINE.
HB0164     MOVE DBMS-INT-ORDER-ITEM
HB0165       TO DBMS-EXT-ORDER-ITEM.
HB0173     MOVE DBMS-INT-ORDER-PRICE
HB0174       TO DBMS-EXT-ORDER-PRICE.
```

```
HB0182      MOVE DBMS-INT-ORDER-QTY
HB0183        TO DBMS-EXT-ORDER-QTY.
HB0192 FILL-EXT-PART.
HB0200      MOVE DBMS-INT-DESCRIPTION
HB0201        TO DBMS-EXT-DESCRIPTION.
HB0209      MOVE DBMS-INT-PART-NO
HB0210        TO DBMS-EXT-PART-NO.
HB0218      MOVE DBMS-INT-UNIT-PRICE
HB0219        TO DBMS-EXT-UNIT-PRICE.
HB0227      MOVE DBMS-INT-STOCK-IN-HAND
HB0228        TO DBMS-EXT-STOCK-IN-HAND.
HB0237 FILL-EXT-PURCHASE.
HB0245      MOVE DBMS-INT-PURCHASE-ORDER-NO
HB0246        TO DBMS-EXT-PURCHASE-ORDER-NO.
HB0254      MOVE DBMS-INT-PURCHASE-DATE
HB0255        TO DBMS-EXT-PURCHASE-DATE.
HB0263      MOVE DBMS-INT-PURCHASE-QTY
HB0264        TO DBMS-EXT-PURCHASE-QTY.
HB0272      MOVE DBMS-INT-PURCHASE-PRICE
HB0273        TO DBMS-EXT-PURCHASE-PRICE.
HB0281      MOVE DBMS-INT-PURCHASE-SUPPLIER-NO
HB0282        TO DBMS-EXT-PURCHASE-SUPPLIER-NO.
PA0305 USE-CUSTOMER.
PA0306      PERFORM SET-CURR-CUSTOMER.
PA0307      MOVE 01 TO CURRENT-ENTITY-CODE.
PA0308      PERFORM INN-CUSTOMER.
PA0313      IF FUNCTION = "READ "
PA0314          PERFORM READ-CUSTOMER
PA0315      ELSE
PA0316      IF FUNCTION = "FIRST "
PA0317          PERFORM FIRST-CUSTOMER
PA0318      ELSE
PA0322      IF FUNCTION = "NEXT "
PA0323          PERFORM NEXT-CUSTOMER
PA0324      ELSE
PA0345      IF FUNCTION = "WRITE "
PA0346          PERFORM WRITE-CUSTOMER
PA0347      ELSE
PA0360      IF FUNCTION = "DELETE "
PA0361          PERFORM DELETE-CUSTOMER
PA0362      ELSE
PA0366      IF FUNCTION = "REWRITE "
PA0367          PERFORM REWRITE-CUSTOMER
PA0368      ELSE
PA0394      IF FUNCTION = "FORMAT "
PA0395          PERFORM FORMAT-CUSTOMER
PA0396      ELSE
PA0428          PERFORM NO-SUCH-FUNCTION.
PA0429      IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0430          PERFORM OUT-CUSTOMER.
PA0432 USE-INVOICE.
PA0433*
PA0434*     TEST IF OWNING ENTITY CURRENT
PA0435*
PA0436      IF DBMS-CUR-CUSTOMER = "YES"
PA0437          PERFORM PROCESS-INVOICE
PA0438      ELSE MOVE 199 TO RESULT.
PA0439 PROCESS-INVOICE.
PA0440      PERFORM SET-CURR-INVOICE.
PA0441      MOVE 02 TO CURRENT-ENTITY-CODE.
PA0442      PERFORM INN-INVOICE.
```

```
PA0447      IF FUNCTION = "READ "
PA0448          PERFORM READ-INVOICE
PA0449      ELSE
PA0450      IF FUNCTION = "FIRST "
PA0451          PERFORM FIRST-INVOICE
PA0452      ELSE
PA0456      IF FUNCTION = "NEXT "
PA0457          PERFORM NEXT-INVOICE
PA0458      ELSE
PA0479      IF FUNCTION = "WRITE "
PA0480          PERFORM WRITE-INVOICE
PA0481      ELSE
PA0494      IF FUNCTION = "DELETE "
PA0495          PERFORM DELETE-INVOICE
PA0496      ELSE
PA0500      IF FUNCTION = "REWRITE "
PA0501          PERFORM REWRITE-INVOICE
PA0502      ELSE
PA0526      IF FUNCTION = "FORMAT "
PA0527          PERFORM FORMAT-INVOICE
PA0528      ELSE
PA0556          PERFORM NO-SUCH-FUNCTION.
PA0557      IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0558          PERFORM OUT-INVOICE.
PA0560 USE-ORDER-LINE.
PA0561*
PA0562*     TEST IF OWNING ENTITY CURRENT
PA0563*
PA0564      IF DBMS-CUR-INVOICE = "YES"
PA0565          PERFORM PROCESS-ORDER-LINE
PA0566      ELSE MOVE 199 TO RESULT.
PA0567 PROCESS-ORDER-LINE.
PA0568      PERFORM SET-CURR-ORDER-LINE.
PA0569      MOVE 03 TO CURRENT-ENTITY-CODE.
PA0570      PERFORM INN-ORDER-LINE.
PA0575      IF FUNCTION = "READ "
PA0576          PERFORM READ-ORDER-LINE
PA0577      ELSE
PA0578      IF FUNCTION = "FIRST "
PA0579          PERFORM FIRST-ORDER-LINE
PA0580      ELSE
PA0584      IF FUNCTION = "NEXT "
PA0585          PERFORM NEXT-ORDER-LINE
PA0586      ELSE
PA0607      IF FUNCTION = "WRITE "
PA0608          PERFORM WRITE-ORDER-LINE
PA0609      ELSE
PA0622      IF FUNCTION = "DELETE "
PA0623          PERFORM DELETE-ORDER-LINE
PA0624      ELSE
PA0628      IF FUNCTION = "REWRITE "
PA0629          PERFORM REWRITE-ORDER-LINE
PA0630      ELSE
PA0652      IF FUNCTION = "FORMAT "
PA0653          PERFORM FORMAT-ORDER-LINE
PA0654      ELSE
PA0684          PERFORM NO-SUCH-FUNCTION.
PA0685      IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0686          PERFORM OUT-ORDER-LINE.
PA0705 USE-PART.
PA0706      PERFORM SET-CURR-PART.
```

```
PA0707     MOVE 01 TO CURRENT-ENTITY-CODE.
PA0708     PERFORM INN-PART.
PA0713     IF FUNCTION = "READ "
PA0714         PERFORM READ-PART
PA0715     ELSE
PA0716     IF FUNCTION = "FIRST "
PA0717         PERFORM FIRST-PART
PA0718     ELSE
PA0722     IF FUNCTION = "NEXT "
PA0723         PERFORM NEXT-PART
PA0724     ELSE
PA0745     IF FUNCTION = "WRITE "
PA0746         PERFORM WRITE-PART
PA0747     ELSE
PA0760     IF FUNCTION = "DELETE "
PA0761         PERFORM DELETE-PART
PA0762     ELSE
PA0766     IF FUNCTION = "REWRITE "
PA0767         PERFORM REWRITE-PART
PA0768     ELSE
PA0792     IF FUNCTION = "FORMAT "
PA0793         PERFORM FORMAT-PART
PA0794     ELSE
PA0823         PERFORM NO-SUCH-FUNCTION.
PA0824     IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0825         PERFORM OUT-PART.
PA0827 USE-PURCHASE.
PA0828*
PA0829*    TEST IF OWNING ENTITY CURRENT
PA0830*
PA0831     IF DBMS-CUR-PART = "YES"
PA0832         PERFORM PROCESS-PURCHASE
PA0833     ELSE MOVE 199 TO RESULT.
PA0834 PROCESS-PURCHASE.
PA0835     PERFORM SET-CURR-PURCHASE.
PA0836     MOVE 02 TO CURRENT-ENTITY-CODE.
PA0837     PERFORM INN-PURCHASE.
PA0842     IF FUNCTION = "READ "
PA0843         PERFORM READ-PURCHASE
PA0844     ELSE
PA0845     IF FUNCTION = "FIRST "
PA0846         PERFORM FIRST-PURCHASE
PA0847     ELSE
PA0851     IF FUNCTION = "NEXT "
PA0852         PERFORM NEXT-PURCHASE
PA0853     ELSE
PA0874     IF FUNCTION = "WRITE "
PA0875         PERFORM WRITE-PURCHASE
PA0876     ELSE
PA0889     IF FUNCTION = "DELETE "
PA0890         PERFORM DELETE-PURCHASE
PA0891     ELSE
PA0895     IF FUNCTION = "REWRITE "
PA0896         PERFORM REWRITE-PURCHASE
PA0897     ELSE
PA0919     IF FUNCTION = "FORMAT "
PA0920         PERFORM FORMAT-PURCHASE
PA0921     ELSE
PA0954         PERFORM NO-SUCH-FUNCTION.
PA0955     IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0956         PERFORM OUT-PURCHASE.
```

```
PC0319 FIRST-CUSTOMER.
PC0320     MOVE "NO" TO CUR-INT-CUSTOMER.
PC0321     PERFORM NEXT-CUSTOMER.
PC0325 NEXT-CUSTOMER.
PC0326     IF CUR-INT-CUSTOMER = "NO"
PC0327         PERFORM SETUP-KEY-CUSTOMER
PC0328         MOVE SPACES TO DBMS-INT-CUSTOMER-NO.
PC0329     MOVE DBMS-INT-CUSTOMER
PC0330         TO DBMS-REC-CUSTOMER.
PC0331     MOVE "YES" TO CUR-INT-CUSTOMER.
PC0332     PERFORM MAKE-CURR-CUSTOMER.
PC0333     START INTERNAL-CUSTOMERS
PC0334         KEY IS GREATER THAN DBMS-KEY-CUSTOMERS
PC0335             INVALID KEY MOVE "NO" TO CUR-INT-CUSTOMER.
PC0336     IF CUR-INT-CUSTOMER = "YES"
PC0337         MOVE "YES" TO SEARCH-FLAG
PC0338         PERFORM LOOK-FOR-CUSTOMER
PC0339             UNTIL SEARCH-FLAG = "END".
PC0340     IF CUR-INT-CUSTOMER = "YES"
PC0341         MOVE DBMS-REC-CUSTOMER
PC0342         TO DBMS-INT-CUSTOMER
PC0343         PERFORM FILL-EXT-CUSTOMER
PC0344     ELSE MOVE 111 TO RESULT.
PC0348 WRITE-CUSTOMER.
PC0349     MOVE "YES" TO DBMS-CUR-CUSTOMER.
PC0350     PERFORM SETUP-KEY-CUSTOMER.
PC0351     PERFORM CLEAR-INT-CUSTOMER.
PC0352     PERFORM FILL-INT-CUSTOMER.
PC0353     MOVE DBMS-INT-CUSTOMER
PC0354         TO DBMS-REC-CUSTOMER.
PC0355     MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
PC0356         OF DBMS-RCD-CUSTOMERS
PC0357     WRITE DBMS-REC-CUSTOMER
PC0358         INVALID KEY PERFORM WRITE-INVALID-KEY
PC0359                     MOVE "NO" TO DBMS-CUR-CUSTOMER.
PC0363 DELETE-CUSTOMER.
PC0364     DELETE INTERNAL-CUSTOMERS
PC0365         INVALID KEY PERFORM DELETE-INVALID-KEY.
PC0369 REWRITE-CUSTOMER.
PC0370     REWRITE DBMS-REC-CUSTOMER
PC0371         FROM DBMS-INT-CUSTOMER
PC0372         INVALID KEY PERFORM REWRITE-INVALID-KEY.
PC0397 FORMAT-CUSTOMER.
PC0398     MOVE DBMS-FMT-CUSTOMER TO UWA.
PC0400 READ-CUSTOMER.
PC0401     MOVE "YES" TO CUR-INT-CUSTOMER.
PC0402     PERFORM SETUP-KEY-CUSTOMER.
PC0403     MOVE DBMS-INT-CUSTOMER
PC0404         TO DBMS-REC-CUSTOMER.
PC0405     READ INTERNAL-CUSTOMERS
PC0406         INVALID KEY PERFORM READ-INVALID-KEY
PC0407                     MOVE "NO" TO CUR-INT-CUSTOMER.
PC0408     IF CUR-INT-CUSTOMER = "YES"
PC0409         MOVE DBMS-REC-CUSTOMER
PC0410             TO DBMS-INT-CUSTOMER
PC0411         PERFORM FILL-EXT-CUSTOMER.
PC0453 FIRST-INVOICE.
PC0454     MOVE "NO" TO CUR-INT-INVOICE.
PC0455     PERFORM NEXT-INVOICE.
PC0459 NEXT-INVOICE.
PC0460     IF CUR-INT-INVOICE = "NO"
```

```
            PERFORM SETUP-KEY-INVOICE
            MOVE SPACES TO DBMS-INT-INVOICE-NO.
        MOVE DBMS-INT-INVOICE
            TO DBMS-REC-INVOICE.
        MOVE "YES" TO CUR-INT-INVOICE.
        PERFORM MAKE-CURR-INVOICE.
        START INTERNAL-CUSTOMERS
            KEY IS GREATER THAN DBMS-KEY-CUSTOMERS
                INVALID KEY MOVE "NO" TO CUR-INT-INVOICE.
        IF CUR-INT-INVOICE = "YES"
            MOVE "YES" TO SEARCH-FLAG
            PERFORM LOOK-FOR-INVOICE
                UNTIL SEARCH-FLAG = "END".
        IF CUR-INT-INVOICE = "YES"
            MOVE DBMS-REC-INVOICE
            TO DBMS-INT-INVOICE
            PERFORM FILL-EXT-INVOICE
        ELSE MOVE 111 TO RESULT.
    WRITE-INVOICE.
        MOVE "YES" TO DBMS-CUR-INVOICE.
        PERFORM SETUP-KEY-INVOICE.
        PERFORM CLEAR-INT-INVOICE.
        PERFORM FILL-INT-INVOICE.
        MOVE DBMS-INT-INVOICE
            TO DBMS-REC-INVOICE.
        MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
            OF DBMS-RCD-CUSTOMERS
        WRITE DBMS-REC-INVOICE
            INVALID KEY PERFORM WRITE-INVALID-KEY
                        MOVE "NO" TO DBMS-CUR-INVOICE.
    DELETE-INVOICE.
        DELETE INTERNAL-CUSTOMERS
            INVALID KEY PERFORM DELETE-INVALID-KEY.
    REWRITE-INVOICE.
        REWRITE DBMS-REC-INVOICE
            FROM DBMS-INT-INVOICE
            INVALID KEY PERFORM REWRITE-INVALID-KEY.
    FORMAT-INVOICE.
        MOVE DBMS-FMT-INVOICE TO UWA.
    READ-INVOICE.
        MOVE "YES" TO CUR-INT-INVOICE.
        PERFORM SETUP-KEY-INVOICE.
        MOVE DBMS-INT-INVOICE
            TO DBMS-REC-INVOICE.
        READ INTERNAL-CUSTOMERS
            INVALID KEY PERFORM READ-INVALID-KEY
                        MOVE "NO" TO CUR-INT-INVOICE.
        IF CUR-INT-INVOICE = "YES"
            MOVE DBMS-REC-INVOICE
                TO DBMS-INT-INVOICE
            PERFORM FILL-EXT-INVOICE.
    FIRST-ORDER-LINE.
        MOVE "NO" TO CUR-INT-ORDER-LINE.
        PERFORM NEXT-ORDER-LINE.
    NEXT-ORDER-LINE.
        IF CUR-INT-ORDER-LINE = "NO"
            PERFORM SETUP-KEY-ORDER-LINE
            MOVE SPACES TO DBMS-INT-ORDER-ITEM.
        MOVE DBMS-INT-ORDER-LINE
            TO DBMS-REC-ORDER-LINE.
        MOVE "YES" TO CUR-INT-ORDER-LINE.
```

```
PC0594      PERFORM MAKE-CURR-ORDER-LINE.
PC0595      START INTERNAL-CUSTOMERS
PC0596          KEY IS GREATER THAN DBMS-KEY-CUSTOMERS
PC0597              INVALID KEY MOVE "NO" TO CUR-INT-ORDER-LINE.
PC0598      IF CUR-INT-ORDER-LINE = "YES"
PC0599          MOVE "YES" TO SEARCH-FLAG
PC0600          PERFORM LOOK-FOR-ORDER-LINE
PC0601              UNTIL SEARCH-FLAG = "END".
PC0602      IF CUR-INT-ORDER-LINE = "YES"
PC0603          MOVE DBMS-REC-ORDER-LINE
PC0604          TO DBMS-INT-ORDER-LINE
PC0605          PERFORM FILL-EXT-ORDER-LINE
PC0606      ELSE MOVE 111 TO RESULT.
PC0610  WRITE-ORDER-LINE.
PC0611      MOVE "YES" TO DBMS-CUR-ORDER-LINE.
PC0612      PERFORM SETUP-KEY-ORDER-LINE.
PC0613      PERFORM CLEAR-INT-ORDER-LINE.
PC0614      PERFORM FILL-INT-ORDER-LINE.
PC0615      MOVE DBMS-INT-ORDER-LINE
PC0616          TO DBMS-REC-ORDER-LINE.
PC0617      MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
PC0618          OF DBMS-RCD-CUSTOMERS
PC0619      WRITE DBMS-REC-ORDER-LINE
PC0620          INVALID KEY PERFORM WRITE-INVALID-KEY
PC0621                      MOVE "NO" TO DBMS-CUR-ORDER-LINE.
PC0625  DELETE-ORDER-LINE.
PC0626      DELETE INTERNAL-CUSTOMERS
PC0627          INVALID KEY PERFORM DELETE-INVALID-KEY.
PC0631  REWRITE-ORDER-LINE.
PC0632      REWRITE DBMS-REC-ORDER-LINE
PC0633          FROM DBMS-INT-ORDER-LINE
PC0634          INVALID KEY PERFORM REWRITE-INVALID-KEY.
PC0655  FORMAT-ORDER-LINE.
PC0656      MOVE DBMS-FMT-ORDER-LINE TO UWA.
PC0658  READ-ORDER-LINE.
PC0659      MOVE "YES" TO CUR-INT-ORDER-LINE.
PC0660      PERFORM SETUP-KEY-ORDER-LINE.
PC0661      MOVE DBMS-INT-ORDER-LINE
PC0662          TO DBMS-REC-ORDER-LINE.
PC0663      READ INTERNAL-CUSTOMERS
PC0664          INVALID KEY PERFORM READ-INVALID-KEY
PC0665                      MOVE "NO" TO CUR-INT-ORDER-LINE.
PC0666      IF CUR-INT-ORDER-LINE = "YES"
PC0667          MOVE DBMS-REC-ORDER-LINE
PC0668              TO DBMS-INT-ORDER-LINE
PC0669          PERFORM FILL-EXT-ORDER-LINE.
PC0719  FIRST-PART.
PC0720      MOVE "NO" TO CUR-INT-PART.
PC0721      PERFORM NEXT-PART.
PC0725  NEXT-PART.
PC0726      IF CUR-INT-PART = "NO"
PC0727          PERFORM SETUP-KEY-PART
PC0728          MOVE SPACES TO DBMS-INT-PART-NO.
PC0729      MOVE DBMS-INT-PART
PC0730          TO DBMS-REC-PART.
PC0731      MOVE "YES" TO CUR-INT-PART.
PC0732      PERFORM MAKE-CURR-PART.
PC0733      START INTERNAL-INVENTORY
PC0734          KEY IS GREATER THAN DBMS-KEY-INVENTORY
PC0735              INVALID KEY MOVE "NO" TO CUR-INT-PART.
PC0736      IF CUR-INT-PART = "YES"
```

```
                MOVE "YES" TO SEARCH-FLAG
                PERFORM LOOK-FOR-PART
                    UNTIL SEARCH-FLAG = "END".
            IF CUR-INT-PART = "YES"
                MOVE DBMS-REC-PART
                TO DBMS-INT-PART
                PERFORM FILL-EXT-PART
            ELSE MOVE 111 TO RESULT.
        WRITE-PART.
            MOVE "YES" TO DBMS-CUR-PART.
            PERFORM SETUP-KEY-PART.
            PERFORM CLEAR-INT-PART.
            PERFORM FILL-INT-PART.
            MOVE DBMS-INT-PART
                TO DBMS-REC-PART.
            MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
                OF DBMS-RCD-INVENTORY
            WRITE DBMS-REC-PART
                INVALID KEY PERFORM WRITE-INVALID-KEY
                                MOVE "NO" TO DBMS-CUR-PART.
        DELETE-PART.
            DELETE INTERNAL-INVENTORY
                INVALID KEY PERFORM DELETE-INVALID-KEY.
        REWRITE-PART.
            REWRITE DBMS-REC-PART
                FROM DBMS-INT-PART
                INVALID KEY PERFORM REWRITE-INVALID-KEY.
        FORMAT-PART.
            MOVE DBMS-FMT-PART TO UWA.
        READ-PART.
            MOVE "YES" TO CUR-INT-PART.
            PERFORM SETUP-KEY-PART.
            MOVE DBMS-INT-PART
                TO DBMS-REC-PART.
            READ INTERNAL-INVENTORY
                INVALID KEY PERFORM READ-INVALID-KEY
                                MOVE "NO" TO CUR-INT-PART.
            IF CUR-INT-PART = "YES"
                MOVE DBMS-REC-PART
                    TO DBMS-INT-PART
                PERFORM FILL-EXT-PART.
        FIRST-PURCHASE.
            MOVE "NO" TO CUR-INT-PURCHASE.
            PERFORM NEXT-PURCHASE.
        NEXT-PURCHASE.
            IF CUR-INT-PURCHASE = "NO"
                PERFORM SETUP-KEY-PURCHASE
                MOVE SPACES TO DBMS-INT-PURCHASE-ORDER-NO.
            MOVE DBMS-INT-PURCHASE
                TO DBMS-REC-PURCHASE.
            MOVE "YES" TO CUR-INT-PURCHASE.
            PERFORM MAKE-CURR-PURCHASE.
            START INTERNAL-INVENTORY
                KEY IS GREATER THAN DBMS-KEY-INVENTORY
                    INVALID KEY MOVE "NO" TO CUR-INT-PURCHASE.
            IF CUR-INT-PURCHASE = "YES"
                MOVE "YES" TO SEARCH-FLAG
                PERFORM LOOK-FOR-PURCHASE
                    UNTIL SEARCH-FLAG = "END".
            IF CUR-INT-PURCHASE = "YES"
                MOVE DBMS-REC-PURCHASE
```

```
 0871             TO DBMS-INT-PURCHASE
 0872             PERFORM FILL-EXT-PURCHASE
 0873         ELSE MOVE 111 TO RESULT.
 0877 WRITE-PURCHASE.
 0878         MOVE "YES" TO DBMS-CUR-PURCHASE.
 0879         PERFORM SETUP-KEY-PURCHASE.
C0880         PERFORM CLEAR-INT-PURCHASE.
C0881         PERFORM FILL-INT-PURCHASE.
C0882         MOVE DBMS-INT-PURCHASE
C0883             TO DBMS-REC-PURCHASE.
C0884         MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
C0885             OF DBMS-RCD-INVENTORY
C0886         WRITE DBMS-REC-PURCHASE
C0887             INVALID KEY PERFORM WRITE-INVALID-KEY
C0888                         MOVE "NO" TO DBMS-CUR-PURCHASE.
C0892 DELETE-PURCHASE.
C0893         DELETE INTERNAL-INVENTORY
C0894             INVALID KEY PERFORM DELETE-INVALID-KEY.
C0898 REWRITE-PURCHASE.
C0899         REWRITE DBMS-REC-PURCHASE
C0900             FROM DBMS-INT-PURCHASE
C0901             INVALID KEY PERFORM REWRITE-INVALID-KEY.
C0922 FORMAT-PURCHASE.
PC0923        MOVE DBMS-FMT-PURCHASE TO UWA.
PC0925 READ-PURCHASE.
PC0926        MOVE "YES" TO CUR-INT-PURCHASE.
PC0927        PERFORM SETUP-KEY-PURCHASE.
PC0928        MOVE DBMS-INT-PURCHASE
PC0929            TO DBMS-REC-PURCHASE.
PC0930        READ INTERNAL-INVENTORY
PC0931            INVALID KEY PERFORM READ-INVALID-KEY
PC0932                        MOVE "NO" TO CUR-INT-PURCHASE.
PC0933        IF CUR-INT-PURCHASE = "YES"
PC0934            MOVE DBMS-REC-PURCHASE
PC0935                TO DBMS-INT-PURCHASE
PC0936            PERFORM FILL-EXT-PURCHASE.
QA0373 LOOK-FOR-CUSTOMER.
QA0374        READ INTERNAL-CUSTOMERS NEXT RECORD
QA0375                AT END MOVE "END" TO SEARCH-FLAG
QA0376                       MOVE "NO" TO CUR-INT-CUSTOMER.
QA0377        IF SEARCH-FLAG = "YES"
QA0378            IF DBMS-CUSTOMER-NO = SPACES
QA0379                MOVE SPACES TO SEARCH-FLAG
QA0380            ELSE IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 01
QA0381                    MOVE "END" TO SEARCH-FLAG.
QA0382        IF SEARCH-FLAG = SPACES
QA0383            MOVE "END" TO SEARCH-FLAG
QA0384            MOVE "NO" TO CUR-INT-CUSTOMER.
QA0507 LOOK-FOR-INVOICE.
QA0508        READ INTERNAL-CUSTOMERS NEXT RECORD
QA0509                AT END MOVE "END" TO SEARCH-FLAG
QA0510                       MOVE "NO" TO CUR-INT-INVOICE.
QA0511        IF SEARCH-FLAG = "YES"
QA0512            IF DBMS-INVOICE-NO = SPACES
QA0513                MOVE SPACES TO SEARCH-FLAG
QA0514            ELSE IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 02
QA0515                    MOVE "END" TO SEARCH-FLAG.
QA0516        IF SEARCH-FLAG = SPACES
QA0517            MOVE "END" TO SEARCH-FLAG
QA0518            MOVE "NO" TO CUR-INT-INVOICE.
QA0635 LOOK-FOR-ORDER-LINE.
```

```
QA0636      READ INTERNAL-CUSTOMERS NEXT RECORD
QA0637              AT END MOVE "END" TO SEARCH-FLAG
QA0638                     MOVE "NO" TO CUR-INT-ORDER-LINE.
QA0639      IF SEARCH-FLAG = "YES"
QA0640          IF DBMS-ORDER-ITEM = SPACES
QA0641              MOVE SPACES TO SEARCH-FLAG
QA0642          ELSE IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 03
QA0643                  MOVE "END" TO SEARCH-FLAG.
QA0644      IF SEARCH-FLAG = SPACES
QA0645          MOVE "END" TO SEARCH-FLAG
QA0646          MOVE "NO" TO CUR-INT-ORDER-LINE.
QA0773 LOOK-FOR-PART.
QA0774      READ INTERNAL-INVENTORY NEXT RECORD
QA0775              AT END MOVE "END" TO SEARCH-FLAG
QA0776                     MOVE "NO" TO CUR-INT-PART.
QA0777      IF SEARCH-FLAG = "YES"
QA0778          IF DBMS-PART-NO = SPACES
QA0779              MOVE SPACES TO SEARCH-FLAG
QA0780          ELSE IF ENTITY-CODE OF DBMS-RCD-INVENTORY = 01
QA0781                  MOVE "END" TO SEARCH-FLAG.
QA0782      IF SEARCH-FLAG = SPACES
QA0783          MOVE "END" TO SEARCH-FLAG
QA0784          MOVE "NO" TO CUR-INT-PART.
QA0902 LOOK-FOR-PURCHASE.
QA0903      READ INTERNAL-INVENTORY NEXT RECORD
QA0904              AT END MOVE "END" TO SEARCH-FLAG
QA0905                     MOVE "NO" TO CUR-INT-PURCHASE.
QA0906      IF SEARCH-FLAG = "YES"
QA0907          IF DBMS-PURCHASE-ORDER-NO = SPACES
QA0908              MOVE SPACES TO SEARCH-FLAG
QA0909          ELSE IF ENTITY-CODE OF DBMS-RCD-INVENTORY = 02
QA0910                  MOVE "END" TO SEARCH-FLAG.
QA0911      IF SEARCH-FLAG = SPACES
QA0912          MOVE "END" TO SEARCH-FLAG
QA0913          MOVE "NO" TO CUR-INT-PURCHASE.
QB0412 SETUP-KEY-CUSTOMER.
QB0413      MOVE DBMS-EXT-CUSTOMER-NO
QB0414          TO DBMS-INT-CUSTOMER-NO.
QB0544 SETUP-KEY-INVOICE.
QB0545      MOVE DBMS-INT-CUSTOMER
QB0546          TO DBMS-INT-INVOICE.
QB0547      MOVE DBMS-EXT-INVOICE-NO
QB0548          TO DBMS-INT-INVOICE-NO.
QB0670 SETUP-KEY-ORDER-LINE.
QB0671      MOVE DBMS-INT-INVOICE
QB0672          TO DBMS-INT-ORDER-LINE.
QB0673      MOVE DBMS-EXT-ORDER-ITEM
QB0674          TO DBMS-INT-ORDER-ITEM.
QB0810 SETUP-KEY-PART.
QB0811      MOVE DBMS-EXT-PART-NO
QB0812          TO DBMS-INT-PART-NO.
QB0937 SETUP-KEY-PURCHASE.
QB0938      MOVE DBMS-INT-PART
QB0939          TO DBMS-INT-PURCHASE.
QB0940      MOVE DBMS-EXT-PURCHASE-ORDER-NO
QB0941          TO DBMS-INT-PURCHASE-ORDER-NO.
QC0385 CLEAR-INT-CUSTOMER.
QC0421      MOVE SPACES TO DBMS-INT-CUSTOMER-NAME.
QC0423      MOVE ZEROS TO DBMS-INT-CREDIT-LIMIT.
QC0425      MOVE ZEROS TO DBMS-INT-BALANCE.
QC0427      MOVE ZEROS TO DBMS-INT-TOTAL-VALUE-ON-ORDER.
```

```
C0519 CLEAR-INT-INVOICE.
C0555     MOVE ZEROS TO DBMS-INT-INVOICE-DATE.
C0647 CLEAR-INT-ORDER-LINE.
C0681     MOVE ZEROS TO DBMS-INT-ORDER-QTY.
C0683     MOVE ZEROS TO DBMS-INT-ORDER-PRICE.
C0785 CLEAR-INT-PART.
C0818     MOVE SPACES TO DBMS-INT-DESCRIPTION.
C0820     MOVE ZEROS TO DBMS-INT-UNIT-PRICE.
C0822     MOVE ZEROS TO DBMS-INT-STOCK-IN-HAND.
C0914 CLEAR-INT-PURCHASE.
C0947     MOVE ZEROS TO DBMS-INT-PURCHASE-QTY.
C0949     MOVE ZEROS TO DBMS-INT-PURCHASE-DATE.
C0951     MOVE ZEROS TO DBMS-INT-PURCHASE-PRICE.
C0953     MOVE SPACES TO DBMS-INT-PURCHASE-SUPPLIER-NO.
QD0386 MAKE-CURR-CUSTOMER.
QD0387     MOVE "YES" TO CUR-INT-CUSTOMER.
QD0390     MOVE "NO" TO CUR-INT-INVOICE.
QD0392     MOVE "NO" TO CUR-INT-ORDER-LINE.
QD0520 MAKE-CURR-INVOICE.
QD0521     MOVE "YES" TO CUR-INT-INVOICE.
QD0524     MOVE "NO" TO CUR-INT-ORDER-LINE.
QD0648 MAKE-CURR-ORDER-LINE.
QD0649     MOVE "YES" TO CUR-INT-ORDER-LINE.
QD0786 MAKE-CURR-PART.
QD0787     MOVE "YES" TO CUR-INT-PART.
QD0790     MOVE "NO" TO CUR-INT-PURCHASE.
QD0915 MAKE-CURR-PURCHASE.
QD0916     MOVE "YES" TO CUR-INT-PURCHASE.
QE0388 SET-CURR-CUSTOMER.
QE0389     MOVE "YES" TO DBMS-CUR-CUSTOMER.
QE0391     MOVE "NO" TO DBMS-CUR-INVOICE.
QE0393     MOVE "NO" TO DBMS-CUR-ORDER-LINE.
QE0522 SET-CURR-INVOICE.
QE0523     MOVE "YES" TO DBMS-CUR-INVOICE.
QE0525     MOVE "NO" TO DBMS-CUR-ORDER-LINE.
QE0650 SET-CURR-ORDER-LINE.
QE0651     MOVE "YES" TO DBMS-CUR-ORDER-LINE.
QE0788 SET-CURR-PART.
QE0789     MOVE "YES" TO DBMS-CUR-PART.
QE0791     MOVE "NO" TO DBMS-CUR-PURCHASE.
QE0917 SET-CURR-PURCHASE.
QE0918     MOVE "YES" TO DBMS-CUR-PURCHASE.
SA0309 INN-CUSTOMER.
SA0310     MOVE UWA TO DBMS-EXT-CUSTOMER.
SA0311 OUT-CUSTOMER.
SA0312     MOVE DBMS-EXT-CUSTOMER TO UWA.
SA0443 INN-INVOICE.
SA0444     MOVE UWA TO DBMS-EXT-INVOICE.
SA0445 OUT-INVOICE.
SA0446     MOVE DBMS-EXT-INVOICE TO UWA.
SA0571 INN-ORDER-LINE.
SA0572     MOVE UWA TO DBMS-EXT-ORDER-LINE.
SA0573 OUT-ORDER-LINE.
SA0574     MOVE DBMS-EXT-ORDER-LINE TO UWA.
SA0709 INN-PART.
SA0710     MOVE UWA TO DBMS-EXT-PART.
SA0711 OUT-PART.
SA0712     MOVE DBMS-EXT-PART TO UWA.
SA0838 INN-PURCHASE.
SA0839     MOVE UWA TO DBMS-EXT-PURCHASE.
SA0840 OUT-PURCHASE.
```

```
[illegible]         [illegible] UKR.
A0038 NEW-DATA-BASE.
A0039      IF DATA-BASE-OPEN-FLAG = "YES"
A0040           MOVE 101 TO RESULT
A0041      ELSE
A0042           PERFORM CREATE-DATA-BASE
A0043           PERFORM CLOSE-DATA-BASE
A0044           PERFORM UPDATE-DATA-BASE
A0045            MOVE "YES" TO DATA-BASE-OPEN-FLAG.
A0046 OLD-DATA-BASE.
A0047      IF DATA-BASE-OPEN-FLAG = "YES"
A0048           MOVE 101 TO RESULT
A0049      ELSE PERFORM UPDATE-DATA-BASE
A0050            MOVE "YES" TO DATA-BASE-OPEN-FLAG.
A0051 RELEASE-DATA-BASE.
A0052      IF DATA-BASE-OPEN-FLAG = "NO"
A0053           MOVE 102 TO RESULT
A0054      ELSE PERFORM CLOSE-DATA-BASE
A0055            MOVE "NO" TO DATA-BASE-OPEN-FLAG.
B0056 CREATE-DATE-BASE.
B0301      OPEN OUTPUT INTERNAL-CUSTOMERS.
B0701      OPEN OUTPUT INTERNAL-INVENTORY.
C0057 UPDATE-DATA-BASE.
C0302      OPEN I-O INTERNAL-CUSTOMERS.
C0702      OPEN I-O INTERNAL-INVENTORY.
D0058 CLOSE-DATA-BASE.
D0303      CLOSE INTERNAL-CUSTOMERS.
D0703      CLOSE INTERNAL-INVENTORY.
A0059 NO-SUCH-FUNCTION.
A0060      MOVE 105 TO RESULT.
A0061 READ-INVALID-KEY.
A0062      MOVE FILE-STATUS TO RESULT.
A0063 READ-AT-END.
A0064      MOVE FILE-STATUS TO RESULT.
A0065 WRITE-INVALID-KEY.
A0066      MOVE 107 TO RESULT.
A0067 DELETE-INVALID-KEY.
A0068      MOVE 108 TO RESULT.
A0069 REWRITE-INVALID-KEY.
A0070      MOVE 109 TO RESULT.
A0071 NO-SUCH-RECORD.
A0072      MOVE 104 TO RESULT.
A0073 START-ERROR.
A0074      MOVE 111 TO RESULT.
```

Mapping Code Example 2

The following COBOL code was generated by the PYRAMID mapping code generator using the source code

```
INTERNAL SCHEMA NAME IS MANUFACTURING.
EXTERNAL SCHEMA NAME IS INVOICE-QUERY.
```

The generated code can be incorporated in the PYRAMID Query Program PQUERY to allow the QUILL query language to be used to interrogate CUSTOMERS, INVOICES and ORDER-LINES.

```
A0001 IDENTIFICATION DIVISION.
A0002 PROGRAM-ID. DBMS.
A0075*
A0076*         EXTERNAL SCHEMA NAME IS INVOICE-QUERY
A0077*
A0149*
A0150*         INTERNAL SCHEMA NAME IS MANUFACTURING
A0151*
A0003 ENVIRONMENT DIVISION.
A0004 CONFIGURATION SECTION.
A0005 SOURCE-COMPUTER. CYBER.
A0006 OBJECT-COMPUTER. CYBER.
B0007 INPUT-OUTPUT SECTION.
B0008 FILE-CONTROL.
B0152      SELECT INTERNAL-CUSTOMERS
B0153              ASSIGN TO "ORDERS"
B0154      ORGANIZATION IS INDEXED
B0155          ACCESS MODE IS DYNAMIC
B0156      RECORD KEY IS DBMS-KEY-CUSTOMERS
B0356      FILE STATUS IS FILE-STATUS.
CA0009 DATA DIVISION.
CA0010 FILE SECTION.
CB0157 FD  INTERNAL-CUSTOMERS
CB0158     LABEL RECORDS OMITTED.
CB0159 01  DBMS-RCD-CUSTOMERS.
CB0160     02  DBMS-KEY-CUSTOMERS.
CB0161         03 DBMS-CUSTOMER-NO          PICTURE IS X(6).
CB0162         03 DBMS-INVOICE-NO           PICTURE IS X(6).
CB0163         03 DBMS-ORDER-ITEM           PICTURE IS X(4).
CB0164     02  ENTITY-CODE PICTURE IS 99.
CB0197 01  DBMS-REC-CUSTOMER PICTURE IS X(77).
CB0223 01  DBMS-REC-INVOICE PICTURE IS X(22).
CB0355 01  DBMS-REC-ORDER-LINE PICTURE IS X(27).
CC0011 WORKING-STORAGE SECTION.
CC0012 01  FILE-STATUS PICTURE IS XX.
CC0013 01  DATA-BASE-OPEN-FLAG PIC X(3)
CC0014             VALUE IS "NO".
CC0015 01  SEARCH-FLAG PICTURE IS XXX.
CC0016 01  CURRENT-ENTITY-CODE PIC 99.
CC0017 01  SAME-OWNER                PICTURE IS XXX.
CG0081 01  DBMS-CUR-ORDER-LINE PICTURE IS XXX VALUE IS "NO".
CJ0082 01  DBMS-EXT-ORDER-LINE.
CJ0087     02 DBMS-EXT-CUSTOMER-NAME PICTURE IS X(40).
CJ0094     02 DBMS-EXT-CUSTOMER-NO PICTURE IS X(6).
CJ0101     02 DBMS-EXT-CREDIT-LIMIT PICTURE IS 9(8).
CJ0108     02 DBMS-EXT-INVOICE-NO PICTURE IS X(6).
CJ0115     02 DBMS-EXT-INVOICE-DATE PICTURE IS 9(6).
CJ0122     02 DBMS-EXT-ORDER-ITEM PICTURE IS X(4).
CJ0131     02 DBMS-EXT-ORDER-PRICE PICTURE IS 9(5).
CJ0140     02 DBMS-EXT-ORDER-QTY PICTURE IS 9(6).
CK0184 01  DBMS-INT-CUSTOMER.
CK0185     02  DBMS-INT-CUSTOMER-NO PICTURE IS X(6).
CK0186     02  DBMS-KEY-001 PICTURE IS X(6).
CK0187     02  DBMS-KEY-002 PICTURE IS X(4).
CK0188     02  FILLER PICTURE IS 99.
CK0189     02 DBMS-INT-CUSTOMER-NAME PICTURE IS X(30).
CK0191     02 DBMS-INT-CREDIT-LIMIT PICTURE IS 9(8).
CK0193     02 DBMS-INT-BALANCE PICTURE IS 9(10).
CK0195     02 DBMS-INT-TOTAL-VALUE-ON-ORDER PICTURE IS 9(8).
CK0216 01  DBMS-INT-INVOICE.
CK0217     02  DBMS-KEY-003 PICTURE IS X(6).
```

```
K0218      02  DBMS-PRINT-INVOICE-NO PICTURE IS X(4).
K0219      02  DBMS-KEY-004 PICTURE IS X(4).
K0220      02  FILLER PICTURE IS 99.
K0221      02 DBMS-INT-INVOICE-DATE PICTURE IS 9(6).
K0343 01   DBMS-INT-ORDER-LINE.
K0344      02  DBMS-KEY-005 PICTURE IS X(6).
K0345      02  DBMS-KEY-006 PICTURE IS X(6).
K0346      02  DBMS-INT-ORDER-ITEM PICTURE IS X(4).
K0347      02  FILLER PICTURE IS 99.
K0348      02 DBMS-INT-ORDER-QTY PICTURE IS 9(6).
K0350      02 DBMS-INT-ORDER-PRICE PICTURE IS 9(5).
L0085 01   DBMS-FMT-ORDER-LINE.
L0086      02  DBMS-NOI-ORDER-LINE PICTURE IS 99 VALUE IS 08.
CL0088     02  FILLER PIC X(20) VALUE IS "CUSTOMER-NAME".
CL0089     02  FILLER PIC X VALUE IS "C".
CL0090     02  FILLER PIC 9999 VALUE IS 0001.
CL0091     02  FILLER PIC S999V99 VALUE IS 040.
CL0095     02  FILLER PIC X(20) VALUE IS "CUSTOMER-NO".
CL0096     02  FILLER PIC X VALUE IS "C".
CL0097     02  FILLER PIC 9999 VALUE IS 0041.
CL0098     02  FILLER PIC S999V99 VALUE IS 006.
CL0102     02  FILLER PIC X(20) VALUE IS "CREDIT-LIMIT".
CL0103     02  FILLER PIC X VALUE IS "N".
CL0104     02  FILLER PIC 9999 VALUE IS 0047.
CL0105     02  FILLER PIC S999V99 VALUE IS 008.
CL0109     02  FILLER PIC X(20) VALUE IS "INVOICE-NO".
CL0110     02  FILLER PIC X VALUE IS "C".
CL0111     02  FILLER PIC 9999 VALUE IS 0055.
CL0112     02  FILLER PIC S999V99 VALUE IS 006.
CL0116     02  FILLER PIC X(20) VALUE IS "INVOICE-DATE".
CL0117     02  FILLER PIC X VALUE IS "N".
CL0118     02  FILLER PIC 9999 VALUE IS 0061.
CL0119     02  FILLER PIC S999V99 VALUE IS 006.
CL0123     02  FILLER PIC X(20) VALUE IS "ORDER-ITEM".
CL0124     02  FILLER PIC X VALUE IS "C".
CL0125     02  FILLER PIC 9999 VALUE IS 0067.
CL0126     02  FILLER PIC S999V99 VALUE IS 004.
CL0132     02  FILLER PIC X(20) VALUE IS "ORDER-PRICE".
CL0133     02  FILLER PIC X VALUE IS "N".
CL0134     02  FILLER PIC 9999 VALUE IS 0071.
CL0135     02  FILLER PIC S999V99 VALUE IS 005.
CL0141     02  FILLER PIC X(20) VALUE IS "ORDER-QTY".
CL0142     02  FILLER PIC X VALUE IS "N".
CL0143     02  FILLER PIC 9999 VALUE IS 0076.
CL0144     02  FILLER PIC S999V99 VALUE IS 006.
CS0169 01  CUR-INT-CUSTOMER PICTURE IS XXX VALUE IS "NO".
CS0198 01  CUR-INT-INVOICE PICTURE IS XXX VALUE IS "NO".
CS0324 01  CUR-INT-ORDER-LINE PICTURE IS XXX VALUE IS "NO".
CX0168 01  BUFFER-CUSTOMERS PICTURE IS X(5).
CZ0018 LINKAGE SECTION.
CZ0019 01  FUNCTION PIC X(10).
CZ0020 01  THE-RECORD-NAME PIC X(20).
CZ0021 01  RESULT PIC 999.
CZ0022 01  UWA PIC X(512).
DA0023 PROCEDURE DIVISION USING FUNCTION,
DA0024     THE-RECORD-NAME, UWA,
DA0025     RESULT.
EA0026 INITIAL-PARAGRAPH.
EA0027     MOVE ZERO TO RESULT.
EA0028     IF FUNCTION = "NEW "
EA0029          PERFORM NEW-DATA-BASE
```

```
A0030      ELSE IF FUNCTION = "HOLD "
               PERFORM HOLD-DATA-BASE
A0031
A0032          ELSE IF FUNCTION = "RELEASE "
A0033               PERFORM RELEASE-DATA-BASE
A0034               ELSE PERFORM BRANCH-ON-RECORD-NAME.
FA0035 FINAL-PARAGRAPH.
FA0036     EXIT PROGRAM.
FA0037 BRANCH-ON-RECORD-NAME.
FA0078         IF THE-RECORD-NAME = "QUERY-RECORD"
FA0079              PERFORM USE-ORDER-LINE
FA0080     ELSE
FA0357         PERFORM NO-SUCH-RECORD.
HA0083 FILL-INT-ORDER-LINE.
HA0129     MOVE DBMS-EXT-ORDER-ITEM
HA0130       TO DBMS-INT-ORDER-ITEM.
HA0138     MOVE DBMS-EXT-ORDER-PRICE
HA0139       TO DBMS-INT-ORDER-PRICE.
HA0147     MOVE DBMS-EXT-ORDER-QTY
HA0148       TO DBMS-INT-ORDER-QTY.
HB0084 FILL-EXT-ORDER-LINE.
HB0092     MOVE DBMS-INT-CUSTOMER-NAME
HB0093       TO DBMS-EXT-CUSTOMER-NAME.
HB0099     MOVE DBMS-INT-CUSTOMER-NO
HB0100       TO DBMS-EXT-CUSTOMER-NO.
HB0106     MOVE DBMS-INT-CREDIT-LIMIT
HB0107       TO DBMS-EXT-CREDIT-LIMIT.
HB0113     MOVE DBMS-INT-INVOICE-NO
HB0114       TO DBMS-EXT-INVOICE-NO.
HB0120     MOVE DBMS-INT-INVOICE-DATE
HB0121       TO DBMS-EXT-INVOICE-DATE.
HB0127     MOVE DBMS-INT-ORDER-ITEM
HB0128       TO DBMS-EXT-ORDER-ITEM.
HB0136     MOVE DBMS-INT-ORDER-PRICE
HB0137       TO DBMS-EXT-ORDER-PRICE.
HB0145     MOVE DBMS-INT-ORDER-QTY
HB0146       TO DBMS-EXT-ORDER-QTY.
PA0224 USE-ORDER-LINE.
PA0225     PERFORM SET-CURR-ORDER-LINE.
PA0226     MOVE 03 TO CURRENT-ENTITY-CODE.
PA0227     PERFORM INN-ORDER-LINE.
PA0232     IF FUNCTION = "READ "
PA0233          PERFORM READ-ORDER-LINE
PA0234     ELSE
PA0235     IF FUNCTION = "FIRST "
PA0236          PERFORM FIRST-ORDER-LINE
PA0237     ELSE
PA0241     IF FUNCTION = "NEXT "
PA0242          PERFORM NEXT-ORDER-LINE
PA0243     ELSE
PA0264     IF FUNCTION = "WRITE "
PA0265          PERFORM WRITE-ORDER-LINE
PA0266     ELSE
PA0279     IF FUNCTION = "DELETE "
PA0280          PERFORM DELETE-ORDER-LINE
PA0281     ELSE
PA0285     IF FUNCTION = "REWRITE "
PA0286          PERFORM REWRITE-ORDER-LINE
PA0287     ELSE
PA0319     IF FUNCTION = "FORMAT "
PA0320          PERFORM FORMAT-ORDER-LINE
PA0321     ELSE
```

```
A0352           PERFORM NO-SUCH-FUNCTION.
A0353       IF FUNCTION IS NOT EQUAL TO "FORMAT "
A0354           PERFORM OUT-ORDER-LINE.
C0170 READ-CUSTOMER.
C0171       MOVE "YES" TO CUR-INT-CUSTOMER.
C0172       PERFORM SETUP-KEY-CUSTOMER.
C0173       MOVE DBMS-INT-CUSTOMER
C0174           TO DBMS-REC-CUSTOMER.
C0175       READ INTERNAL-CUSTOMERS
C0176           INVALID KEY PERFORM READ-INVALID-KEY
C0177                       MOVE "NO" TO CUR-INT-CUSTOMER.
C0178       IF CUR-INT-CUSTOMER = "YES"
PC0179          MOVE DBMS-REC-CUSTOMER
PC0180              TO DBMS-INT-CUSTOMER.
PC0199 READ-INVOICE.
PC0200      PERFORM READ-CUSTOMER.
PC0201      MOVE "YES" TO CUR-INT-INVOICE.
PC0202      PERFORM SETUP-KEY-INVOICE.
PC0203      MOVE DBMS-INT-INVOICE
PC0204          TO DBMS-REC-INVOICE.
PC0205      READ INTERNAL-CUSTOMERS
PC0206          INVALID KEY PERFORM READ-INVALID-KEY
PC0207                      MOVE "NO" TO CUR-INT-INVOICE.
PC0208      IF CUR-INT-INVOICE = "YES"
PC0209          MOVE DBMS-REC-INVOICE
PC0210              TO DBMS-INT-INVOICE.
PC0238 FIRST-ORDER-LINE.
PC0239      MOVE "NO" TO CUR-INT-ORDER-LINE.
PC0240      PERFORM NEXT-ORDER-LINE.
PC0244 NEXT-ORDER-LINE.
PC0245      IF CUR-INT-ORDER-LINE = "NO"
PC0246          PERFORM SETUP-KEY-ORDER-LINE
PC0247          MOVE SPACES TO DBMS-INT-ORDER-ITEM.
PC0248      MOVE DBMS-INT-ORDER-LINE
PC0249          TO DBMS-REC-ORDER-LINE.
PC0250      MOVE "YES" TO CUR-INT-ORDER-LINE.
PC0251      PERFORM MAKE-CURR-ORDER-LINE.
PC0252      START INTERNAL-CUSTOMERS
PC0253          KEY IS GREATER THAN DBMS-KEY-CUSTOMERS
PC0254              INVALID KEY MOVE "NO" TO CUR-INT-ORDER-LINE.
PC0255      IF CUR-INT-ORDER-LINE = "YES"
PC0256          MOVE "YES" TO SEARCH-FLAG
PC0257          PERFORM LOOK-FOR-ORDER-LINE
PC0258              UNTIL SEARCH-FLAG = "END".
PC0259      IF CUR-INT-ORDER-LINE = "YES"
PC0260          MOVE DBMS-REC-ORDER-LINE
PC0261          TO DBMS-INT-ORDER-LINE
PC0262          PERFORM FILL-EXT-ORDER-LINE
PC0263      ELSE MOVE 111 TO RESULT.
PC0267 WRITE-ORDER-LINE.
PC0268      MOVE "YES" TO DBMS-CUR-ORDER-LINE.
PC0269      PERFORM SETUP-KEY-ORDER-LINE.
PC0270      PERFORM CLEAR-INT-ORDER-LINE.
PC0271      PERFORM FILL-INT-ORDER-LINE.
PC0272      MOVE DBMS-INT-ORDER-LINE
PC0273          TO DBMS-REC-ORDER-LINE.
PC0274      MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
PC0275          OF DBMS-RCD-CUSTOMERS
PC0276      WRITE DBMS-REC-ORDER-LINE
PC0277          INVALID KEY PERFORM WRITE-INVALID-KEY
PC0278                      MOVE "NO" TO DBMS-CUR-ORDER-LINE.
```

```
C0282 DELETE-ORDER-LINE.
C0283      DELETE INTERNAL-CUSTOMERS
C0284           INVALID KEY PERFORM DELETE-INVALID-KEY.
C0288 REWRITE-ORDER-LINE.
C0289      REWRITE DBMS-REC-ORDER-LINE
C0290           FROM DBMS-INT-ORDER-LINE
C0291           INVALID KEY PERFORM REWRITE-INVALID-KEY.
C0322 FORMAT-ORDER-LINE.
C0323      MOVE DBMS-FMT-ORDER-LINE TO UWA.
C0325 READ-ORDER-LINE.
C0326      PERFORM READ-INVOICE.
C0327      MOVE "YES" TO CUR-INT-ORDER-LINE.
C0328      PERFORM SETUP-KEY-ORDER-LINE.
C0329      MOVE DBMS-INT-ORDER-LINE
C0330           TO DBMS-REC-ORDER-LINE.
C0331      READ INTERNAL-CUSTOMERS
C0332           INVALID KEY PERFORM READ-INVALID-KEY
C0333                       MOVE "NO" TO CUR-INT-ORDER-LINE.
C0334      IF CUR-INT-ORDER-LINE = "YES"
C0335           MOVE DBMS-REC-ORDER-LINE
C0336                TO DBMS-INT-ORDER-LINE
C0337           PERFORM FILL-EXT-ORDER-LINE.
QA0292 LOOK-FOR-ORDER-LINE.
QA0293     READ INTERNAL-CUSTOMERS NEXT RECORD
QA0294             AT END MOVE "END" TO SEARCH-FLAG
QA0295                    MOVE "NO" TO CUR-INT-ORDER-LINE.
QA0296     IF SEARCH-FLAG = "YES"
QA0297          IF DBMS-ORDER-ITEM = SPACES
QA0298               MOVE SPACES TO SEARCH-FLAG
QA0299          ELSE IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 03
QA0300                    MOVE "END" TO SEARCH-FLAG.
QA0301     IF SEARCH-FLAG = SPACES
QA0302          IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 02
QA0303               MOVE "YES" TO SEARCH-FLAG
QA0304               MOVE DBMS-REC-INVOICE
QA0305                    TO DBMS-INT-INVOICE.
QA0306     IF SEARCH-FLAG = SPACES
QA0307          IF ENTITY-CODE OF DBMS-RCD-CUSTOMERS = 01
QA0308               MOVE "YES" TO SEARCH-FLAG
QA0309               MOVE DBMS-REC-CUSTOMER
QA0310                    TO DBMS-INT-CUSTOMER.
QA0311     IF SEARCH-FLAG = SPACES
QA0312          MOVE "END" TO SEARCH-FLAG
QA0313          MOVE "NO" TO CUR-INT-ORDER-LINE.
QB0181 SETUP-KEY-CUSTOMER.
QB0182     MOVE DBMS-EXT-CUSTOMER-NO
QB0183          TO DBMS-INT-CUSTOMER-NO.
QB0211 SETUP-KEY-INVOICE.
QB0212     MOVE DBMS-INT-CUSTOMER
QB0213          TO DBMS-INT-INVOICE.
QB0214     MOVE DBMS-EXT-INVOICE-NO
QB0215          TO DBMS-INT-INVOICE-NO.
QB0338 SETUP-KEY-ORDER-LINE.
QB0339     MOVE DBMS-INT-INVOICE
QB0340          TO DBMS-INT-ORDER-LINE.
QB0341     MOVE DBMS-EXT-ORDER-ITEM
QB0342          TO DBMS-INT-ORDER-ITEM.
QC0190     MOVE SPACES TO DBMS-INT-CUSTOMER-NAME.
QC0192     MOVE ZEROS TO DBMS-INT-CREDIT-LIMIT.
QC0194     MOVE ZEROS TO DBMS-INT-BALANCE.
QC0196     MOVE ZEROS TO DBMS-INT-TOTAL-VALUE-ON-ORDER.
```

```
QC0222      MOVE ZEROS TO [illegible]T-INVOICE-DATE.
QC0314 CLEAR-INT-ORDER-LINE.
QC0349      MOVE ZEROS TO DBMS-INT-ORDER-QTY.
QC0351      MOVE ZEROS TO DBMS-INT-ORDER-PRICE.
QD0315 MAKE-CURR-ORDER-LINE.
QD0316      MOVE "YES" TO CUR-INT-ORDER-LINE.
QE0317 SET-CURR-ORDER-LINE.
QE0318      MOVE "YES" TO DBMS-CUR-ORDER-LINE.
SA0228 INN-ORDER-LINE.
SA0229      MOVE UWA TO DBMS-EXT-ORDER-LINE.
SA0230 OUT-ORDER-LINE.
SA0231      MOVE DBMS-EXT-ORDER-LINE TO UWA.
TA0038 NEW-DATA-BASE.
TA0039      IF DATA-BASE-OPEN-FLAG = "YES"
TA0040          MOVE 101 TO RESULT
TA0041      ELSE
TA0042          PERFORM CREATE-DATE-BASE
TA0043          PERFORM CLOSE-DATA-BASE
TA0044          PERFORM UPDATE-DATA-BASE
TA0045           MOVE "YES" TO DATA-BASE-OPEN-FLAG.
TA0046 OLD-DATA-BASE.
TA0047      IF DATA-BASE-OPEN-FLAG = "YES"
TA0048          MOVE 101 TO RESULT
TA0049      ELSE PERFORM UPDATE-DATA-BASE
TA0050           MOVE "YES" TO DATA-BASE-OPEN-FLAG.
TA0051 RELEASE-DATA-BASE.
TA0052      IF DATA-BASE-OPEN-FLAG = "NO"
TA0053          MOVE 102 TO RESULT
TA0054      ELSE PERFORM CLOSE-DATA-BASE
TA0055           MOVE "NO" TO DATA-BASE-OPEN-FLAG.
TB0056 CREATE-DATE-BASE.
TB0165      OPEN OUTPUT INTERNAL-CUSTOMERS.
TC0057 UPDATE-DATA-BASE.
TC0166      OPEN I-O INTERNAL-CUSTOMERS.
TD0058 CLOSE-DATA-BASE.
TD0167      CLOSE INTERNAL-CUSTOMERS.
VA0059 NO-SUCH-FUNCTION.
VA0060      MOVE 105 TO RESULT.
VA0061 READ-INVALID-KEY.
VA0062      MOVE FILE-STATUS TO RESULT.
VA0063 READ-AT-END.
VA0064      MOVE FILE-STATUS TO RESULT.
VA0065 WRITE-INVALID-KEY.
VA0066      MOVE 107 TO RESULT.
VA0067 DELETE-INVALID-KEY.
VA0068      MOVE 108 TO RESULT.
VA0069 REWRITE-INVALID-KEY.
VA0070      MOVE 109 TO RESULT.
VA0071 NO-SUCH-RECORD.
VA0072      MOVE 104 TO RESULT.
VA0073 START-ERROR.
VA0074      MOVE 111 TO RESULT.
```

Mapping Code Example 3

The following COBOL code was generated by the PYRAMID mapping code generator using the source code

```
INTERNAL SCHEMA NAME IS INVENTORY.
EXTERNAL SCHEMA NAME IS PURCHASES.
```

It is used by program NEWITEM to maintain the PARTS database.

```
A0001 IDENTIFICATION DIVISION.
A0002 PROGRAM-ID. DBMS.
A0075*
A0076*          EXTERNAL SCHEMA NAME IS PURCHASES
A0077*
A0177*
A0178*          INTERNAL SCHEMA NAME IS INVENTORY
A0179*
A0003 ENVIRONMENT DIVISION.
A0004 CONFIGURATION SECTION.
A0005 SOURCE-COMPUTER. CYBER.
A0006 OBJECT-COMPUTER. CYBER.
B0007 INPUT-OUTPUT SECTION.
B0008 FILE-CONTROL.
B0180     SELECT INTERNAL-INVENTORY
B0181             ASSIGN TO "PARTS"
B0182     ORGANIZATION IS INDEXED
B0183         ACCESS MODE IS DYNAMIC
B0184     RECORD KEY IS DBMS-KEY-INVENTORY
B0449     FILE STATUS IS FILE-STATUS.
CA0009 DATA DIVISION.
CA0010 FILE SECTION.
CB0185 FD  INTERNAL-INVENTORY
CB0186     LABEL RECORDS OMITTED.
CB0187 01  DBMS-RCD-INVENTORY.
CB0188     02  DBMS-KEY-INVENTORY.
CB0189         03 DBMS-PART-NO                 PICTURE IS X(4).
CB0190         03 DBMS-PURCHASE-ORDER-NO       PICTURE IS X(4).
CB0191     02  ENTITY-CODE PICTURE IS 99.
CB0317 01  DBMS-REC-PART PICTURE IS X(60).
CB0448 01  DBMS-REC-PURCHASE PICTURE IS X(30).
C0011 WORKING-STORAGE SECTION.
C0012 01  FILE-STATUS PICTURE IS XX.
C0013 01  DATA-BASE-OPEN-FLAG PIC X(3)
C0014            VALUE IS "NO".
C0015 01  SEARCH-FLAG PICTURE IS XXX.
C0016 01  CURRENT-ENTITY-CODE PIC 99.
C0017 01  SAME-OWNER                PICTURE IS XXX.
J0081 01  DBMS-CUR-PART PICTURE IS XXX VALUE IS "NO".
J0126 01  DBMS-CUR-PURCHASE PICTURE IS XXX VALUE IS "NO".
J0082 01  DBMS-EXT-PART.
J0087     02 DBMS-EXT-DESCRIPTION PICTURE IS X(40).
J0096     02 DBMS-EXT-PART-NO PICTURE IS X(4).
J0105     02 DBMS-EXT-UNIT-PRICE PICTURE IS 9(6).
J0114     02 DBMS-EXT-STOCK-IN-HAND PICTURE IS 9(6).
J0127 01  DBMS-EXT-PURCHASE.
J0132     02 DBMS-EXT-PURCHASE-ORDER-NO PICTURE IS X(4).
J0141     02 DBMS-EXT-PURCHASE-DATE PICTURE IS 9(6).
J0150     02 DBMS-EXT-PURCHASE-QTY PICTURE IS 9(6).
J0159     02 DBMS-EXT-PURCHASE-PRICE PICTURE IS 9(6).
J0168     02 DBMS-EXT-PURCHASE-SUPPLIER-NO PICTURE IS X(4).
K0304 01  DBMS-INT-PART.
K0305     02  DBMS-INT-PART-NO PICTURE IS X(4).
K0306     02  DBMS-KEY-001 PICTURE IS X(4).
K0307     02  FILLER PICTURE IS 99.
K0308     02 DBMS-INT-DESCRIPTION PICTURE IS X(40).
K0310     02 DBMS-INT-UNIT-PRICE PICTURE IS 9(6).
K0312     02 DBMS-INT-STOCK-IN-HAND PICTURE IS 9(6).
K0433 01  DBMS-INT-PURCHASE.
K0434     02  DBMS-KEY-002 PICTURE IS X(4).
K0435     02  DBMS-INT-PURCHASE-ORDER-NO PICTURE IS X(4).
```

```
CK0436        02  FILLER PICTURE IS [illegible].
CK0437        02 DBMS-INT-PURCHASE-QTY PICTURE IS 9(6).
CK0439        02 DBMS-INT-PURCHASE-DATE PICTURE IS 9(6).
CK0441        02 DBMS-INT-PURCHASE-PRICE PICTURE IS 9(6).
CK0443        02 DBMS-INT-PURCHASE-SUPPLIER-NO PICTURE IS X(4).
CL0085 01    DBMS-FMT-PART.
CL0086        02   DBMS-NOI-PART PICTURE IS 99 VALUE IS 04.
CL0088        02   FILLER PIC X(20) VALUE IS "DESCRIPTION".
CL0089        02   FILLER PIC X VALUE IS "C".
CL0090        02   FILLER PIC 9999 VALUE IS 0001.
CL0091        02   FILLER PIC S999V99 VALUE IS 040.
CL0097        02   FILLER PIC X(20) VALUE IS "PART-NO".
CL0098        02   FILLER PIC X VALUE IS "C".
CL0099        02   FILLER PIC 9999 VALUE IS 0041.
CL0100        02   FILLER PIC S999V99 VALUE IS 004.
CL0106        02   FILLER PIC X(20) VALUE IS "UNIT-PRICE".
CL0107        02   FILLER PIC X VALUE IS "N".
CL0108        02   FILLER PIC 9999 VALUE IS 0045.
CL0109        02   FILLER PIC S999V99 VALUE IS 006.
CL0115        02   FILLER PIC X(20) VALUE IS "STOCK-IN-HAND".
CL0116        02   FILLER PIC X VALUE IS "N".
CL0117        02   FILLER PIC 9999 VALUE IS 0051.
CL0118        02   FILLER PIC S999V99 VALUE IS 006.
CL0130 01    DBMS-FMT-PURCHASE.
CL0131        02   DBMS-NOI-PURCHASE PICTURE IS 99 VALUE IS 05.
CL0133        02   FILLER PIC X(20) VALUE IS "PURCHASE-ORDER-NO".
CL0134        02   FILLER PIC X VALUE IS "C".
CL0135        02   FILLER PIC 9999 VALUE IS 0001.
CL0136        02   FILLER PIC S999V99 VALUE IS 004.
CL0142        02   FILLER PIC X(20) VALUE IS "PURCHASE-DATE".
CL0143        02   FILLER PIC X VALUE IS "N".
CL0144        02   FILLER PIC 9999 VALUE IS 0005.
CL0145        02   FILLER PIC S999V99 VALUE IS 006.
CL0151        02   FILLER PIC X(20) VALUE IS "PURCHASE-QTY".
CL0152        02   FILLER PIC X VALUE IS "N".
CL0153        02   FILLER PIC 9999 VALUE IS 0011.
CL0154        02   FILLER PIC S999V99 VALUE IS 006.
CL0160        02   FILLER PIC X(20) VALUE IS "PURCHASE-PRICE".
CL0161        02   FILLER PIC X VALUE IS "N".
CL0162        02   FILLER PIC 9999 VALUE IS 0017.
CL0163        02   FILLER PIC S999V99 VALUE IS 006.
CL0169        02   FILLER PIC X(20) VALUE IS "PURCHASE-SUPPLIER-NO".
CL0170        02   FILLER PIC X VALUE IS "C".
CL0171        02   FILLER PIC 9999 VALUE IS 0023.
CL0172        02   FILLER PIC S999V99 VALUE IS 004.
CS0288 01    CUR-INT-PART PICTURE IS XXX VALUE IS "NO".
CS0415 01    CUR-INT-PURCHASE PICTURE IS XXX VALUE IS "NO".
CX0195 01    BUFFER-INVENTORY PICTURE IS X(5).
CZ0018 LINKAGE SECTION.
CZ0019 01    FUNCTION PIC X(10).
CZ0020 01    THE-RECORD-NAME PIC X(20).
CZ0021 01    RESULT PIC 999.
CZ0022 01    UWA PIC X(512).
DA0023 PROCEDURE DIVISION USING FUNCTION,
DA0024       THE-RECORD-NAME, UWA,
DA0025       RESULT.
EA0026 INITIAL-PARAGRAPH.
EA0027       MOVE ZERO TO RESULT.
EA0028       IF FUNCTION = "NEW "
EA0029           PERFORM NEW-DATA-BASE
EA0030       ELSE IF FUNCTION = "OLD "
```

```
A0031            PERFORM OLD-DATA-BASE
A0032            ELSE IF FUNCTION = "RELEASE "
A0033                 PERFORM RELEASE-DATA-BASE
A0034                 ELSE PERFORM BRANCH-ON-RECORD-NAME.
A0035 FINAL-PARAGRAPH.
A0036      EXIT PROGRAM.
A0037 BRANCH-ON-RECORD-NAME.
A0078           IF THE-RECORD-NAME = "PART"
A0079                PERFORM USE-PART
A0080      ELSE
FA0123          IF THE-RECORD-NAME = "PURCHASE"
FA0124               PERFORM USE-PURCHASE
FA0125     ELSE
FA0450          PERFORM NO-SUCH-RECORD.
HA0083 FILL-INT-PART.
HA0094     MOVE DBMS-EXT-DESCRIPTION
HA0095       TO DBMS-INT-DESCRIPTION.
HA0103     MOVE DBMS-EXT-PART-NO
HA0104       TO DBMS-INT-PART-NO.
HA0112     MOVE DBMS-EXT-UNIT-PRICE
HA0113       TO DBMS-INT-UNIT-PRICE.
HA0121     MOVE DBMS-EXT-STOCK-IN-HAND
HA0122       TO DBMS-INT-STOCK-IN-HAND.
HA0128 FILL-INT-PURCHASE.
HA0139     MOVE DBMS-EXT-PURCHASE-ORDER-NO
HA0140       TO DBMS-INT-PURCHASE-ORDER-NO.
HA0148     MOVE DBMS-EXT-PURCHASE-DATE
HA0149       TO DBMS-INT-PURCHASE-DATE.
HA0157     MOVE DBMS-EXT-PURCHASE-QTY
HA0158       TO DBMS-INT-PURCHASE-QTY.
HA0166     MOVE DBMS-EXT-PURCHASE-PRICE
HA0167       TO DBMS-INT-PURCHASE-PRICE.
HA0175     MOVE DBMS-EXT-PURCHASE-SUPPLIER-NO
HA0176       TO DBMS-INT-PURCHASE-SUPPLIER-NO.
HB0084 FILL-EXT-PART.
HB0092     MOVE DBMS-INT-DESCRIPTION
HB0093       TO DBMS-EXT-DESCRIPTION.
HB0101     MOVE DBMS-INT-PART-NO
HB0102       TO DBMS-EXT-PART-NO.
HB0110     MOVE DBMS-INT-UNIT-PRICE
HB0111       TO DBMS-EXT-UNIT-PRICE.
HB0119     MOVE DBMS-INT-STOCK-IN-HAND
HB0120       TO DBMS-EXT-STOCK-IN-HAND.
HB0129 FILL-EXT-PURCHASE.
HB0137     MOVE DBMS-INT-PURCHASE-ORDER-NO
HB0138       TO DBMS-EXT-PURCHASE-ORDER-NO.
HB0146     MOVE DBMS-INT-PURCHASE-DATE
HB0147       TO DBMS-EXT-PURCHASE-DATE.
HB0155     MOVE DBMS-INT-PURCHASE-QTY
HB0156       TO DBMS-EXT-PURCHASE-QTY.
HB0164     MOVE DBMS-INT-PURCHASE-PRICE
HB0165       TO DBMS-EXT-PURCHASE-PRICE.
HB0173     MOVE DBMS-INT-PURCHASE-SUPPLIER-NO
HB0174       TO DBMS-EXT-PURCHASE-SUPPLIER-NO.
PA0196 USE-PART.
PA0197     PERFORM SET-CURR-PART.
PA0198     MOVE 01 TO CURRENT-ENTITY-CODE.
PA0199     PERFORM INN-PART.
PA0204     IF FUNCTION = "READ "
PA0205          PERFORM READ-PART
PA0206     ELSE
```

```
PA0207      IF FUNCTION = "FIRST "
PA0208          PERFORM FIRST-PART
PA0209      ELSE
PA0213      IF FUNCTION = "NEXT "
PA0214          PERFORM NEXT-PART
PA0215      ELSE
PA0236      IF FUNCTION = "WRITE "
PA0237          PERFORM WRITE-PART
PA0238      ELSE
PA0251      IF FUNCTION = "DELETE "
PA0252          PERFORM DELETE-PART
PA0253      ELSE
PA0257      IF FUNCTION = "REWRITE "
PA0258          PERFORM REWRITE-PART
PA0259      ELSE
PA0283      IF FUNCTION = "FORMAT "
PA0284          PERFORM FORMAT-PART
PA0285      ELSE
PA0314          PERFORM NC-SUCH-FUNCTION.
PA0315      IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0316          PERFORM OUT-PART.
PA0318 USE-PURCHASE.
PA0319*
PA0320*     TEST IF OWNING ENTITY CURRENT
PA0321*
PA0322      IF DBMS-CUR-PART = "YES"
PA0323          PERFORM PROCESS-PURCHASE
PA0324      ELSE MOVE 199 TO RESULT.
PA0325 PROCESS-PURCHASE.
PA0326      PERFORM SET-CURR-PURCHASE.
PA0327      MOVE 02 TO CURRENT-ENTITY-CODE.
PA0328      PERFORM INN-PURCHASE.
PA0333      IF FUNCTION = "READ "
PA0334          PERFORM READ-PURCHASE
PA0335      ELSE
PA0336      IF FUNCTION = "FIRST "
PA0337          PERFORM FIRST-PURCHASE
PA0338      ELSE
PA0342      IF FUNCTION = "NEXT "
PA0343          PERFORM NEXT-PURCHASE
PA0344      ELSE
PA0365      IF FUNCTION = "WRITE "
PA0366          PERFORM WRITE-PURCHASE
PA0367      ELSE
PA0380      IF FUNCTION = "DELETE "
PA0381          PERFORM DELETE-PURCHASE
PA0382      ELSE
PA0386      IF FUNCTION = "REWRITE "
PA0387          PERFORM REWRITE-PURCHASE
PA0388      ELSE
PA0410      IF FUNCTION = "FORMAT "
PA0411          PERFORM FORMAT-PURCHASE
PA0412      ELSE
PA0445          PERFORM NC-SUCH-FUNCTION.
PA0446      IF FUNCTION IS NOT EQUAL TO "FORMAT "
PA0447          PERFORM OLT-PURCHASE.
PC0210 FIRST-PART.
PC0211      MOVE "NO" TO CUR-INT-PART.
PC0212      PERFORM NEXT-PART.
PC0216 NEXT-PART.
PC0217      IF CUR-INT-PART = "NO"
```

```
PC0218          PERFORM SETUP-KEY-PART
PC0219          MOVE SPACES TO DBMS-INT-PART-NO.
PC0220      MOVE DBMS-INT-PART
PC0221          TO DBMS-REC-PART.
PC0222      MOVE "YES" TO CUR-INT-PART.
PC0223      PERFORM MAKE-CURR-PART.
PC0224      START INTERNAL-INVENTORY
PC0225          KEY IS GREATER THAN DBMS-KEY-INVENTORY
PC0226              INVALID KEY MOVE "NO" TO CUR-INT-PART.
PC0227      IF CUR-INT-PART = "YES"
PC0228          MOVE "YES" TO SEARCH-FLAG
PC0229          PERFORM LOOK-FOR-PART
PC0230              UNTIL SEARCH-FLAG = "END".
PC0231      IF CUR-INT-PART = "YES"
PC0232          MOVE DBMS-REC-PART
PC0233          TO DBMS-INT-PART
PC0234          PERFORM FILL-EXT-PART
PC0235      ELSE MOVE 111 TO RESULT.
PC0239  WRITE-PART.
PC0240      MOVE "YES" TO DBMS-CUR-PART.
PC0241      PERFORM SETUP-KEY-PART.
PC0242      PERFORM CLEAR-INT-PART.
PC0243      PERFORM FILL-INT-PART.
PC0244      MOVE DBMS-INT-PART
PC0245          TO DBMS-REC-PART.
PC0246      MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
PC0247          OF DBMS-RCD-INVENTORY
PC0248      WRITE DBMS-REC-PART
PC0249          INVALID KEY PERFORM WRITE-INVALID-KEY
PC0250                      MOVE "NO" TO DBMS-CUR-PART.
PC0254  DELETE-PART.
PC0255      DELETE INTERNAL-INVENTORY
PC0256          INVALID KEY PERFORM DELETE-INVALID-KEY.
PC0260  REWRITE-PART.
PC0261      REWRITE DBMS-REC-PART
PC0262          FROM DBMS-INT-PART
PC0263          INVALID KEY PERFORM REWRITE-INVALID-KEY.
PC0286  FORMAT-PART.
PC0287      MOVE DBMS-FMT-PART TO UWA.
PC0289  READ-PART.
PC0290      MOVE "YES" TO CUR-INT-PART.
PC0291      PERFORM SETUP-KEY-PART.
PC0292      MOVE DBMS-INT-PART
PC0293          TO DBMS-REC-PART.
PC0294      READ INTERNAL-INVENTORY
PC0295          INVALID KEY PERFORM READ-INVALID-KEY
PC0296                      MOVE "NO" TO CUR-INT-PART.
PC0297      IF CUR-INT-PART = "YES"
PC0298          MOVE DBMS-REC-PART
PC0299              TO DBMS-INT-PART
PC0300          PERFORM FILL-EXT-PART.
PC0339  FIRST-PURCHASE.
PC0340      MOVE "NO" TO CUR-INT-PURCHASE.
PC0341      PERFORM NEXT-PURCHASE.
PC0345  NEXT-PURCHASE.
PC0346      IF CUR-INT-PURCHASE = "NO"
PC0347          PERFORM SETUP-KEY-PURCHASE
PC0348          MOVE SPACES TO DBMS-INT-PURCHASE-ORDER-NO.
PC0349      MOVE DBMS-INT-PURCHASE
PC0350          TO DBMS-REC-PURCHASE.
PC0351      MOVE "YES" TO CUR-INT-PURCHASE.
```

```
PC0352      PERFORM MAKE-CURR-PURCHASE.
PC0353      START INTERNAL-INVENTORY
PC0354          KEY IS GREATER THAN DBMS-KEY-INVENTORY
PC0355              INVALID KEY MOVE "NO" TO CUR-INT-PURCHASE.
PC0356      IF CUR-INT-PURCHASE = "YES"
PC0357          MOVE "YES" TO SEARCH-FLAG
PC0358          PERFORM LOOK-FOR-PURCHASE
PC0359              UNTIL SEARCH-FLAG = "END".
PC0360      IF CUR-INT-PURCHASE = "YES"
PC0361          MOVE DBMS-REC-PURCHASE
PC0362          TO DBMS-INT-PURCHASE
PC0363          PERFORM FILL-EXT-PURCHASE
PC0364      ELSE MOVE 111 TO RESULT.
PC0368 WRITE-PURCHASE.
PC0369      MOVE "YES" TO DBMS-CUR-PURCHASE.
PC0370      PERFORM SETUP-KEY-PURCHASE.
PC0371      PERFORM CLEAR-INT-PURCHASE.
PC0372      PERFORM FILL-INT-PURCHASE.
PC0373      MOVE DBMS-INT-PURCHASE
PC0374          TO DBMS-REC-PURCHASE.
PC0375      MOVE CURRENT-ENTITY-CODE TO ENTITY-CODE
PC0376          OF DBMS-RCD-INVENTORY
PC0377      WRITE DBMS-REC-PURCHASE
PC0378          INVALID KEY PERFORM WRITE-INVALID-KEY
PC0379                      MOVE "NO" TO DBMS-CUR-PURCHASE.
PC0383 DELETE-PURCHASE.
PC0384      DELETE INTERNAL-INVENTORY
PC0385          INVALID KEY PERFORM DELETE-INVALID-KEY.
PC0389 REWRITE-PURCHASE.
PC0390      REWRITE DBMS-REC-PURCHASE
PC0391          FROM DBMS-INT-PURCHASE
PC0392          INVALID KEY PERFORM REWRITE-INVALID-KEY.
PC0413 FORMAT-PURCHASE.
PC0414      MOVE DBMS-FMT-PURCHASE TO UWA.
PC0416 READ-PURCHASE.
PC0417      MOVE "YES" TO CUR-INT-PURCHASE.
PC0418      PERFORM SETUP-KEY-PURCHASE.
PC0419      MOVE DBMS-INT-PURCHASE
PC0420          TO DBMS-REC-PURCHASE.
PC0421      READ INTERNAL-INVENTORY
PC0422          INVALID KEY PERFORM READ-INVALID-KEY
PC0423                      MOVE "NO" TO CUR-INT-PURCHASE.
PC0424      IF CUR-INT-PURCHASE = "YES"
PC0425          MOVE DBMS-REC-PURCHASE
PC0426              TO DBMS-INT-PURCHASE
PC0427          PERFORM FILL-EXT-PURCHASE.
QA0264 LOOK-FOR-PART.
QA0265      READ INTERNAL-INVENTORY NEXT RECORD
QA0266              AT END MOVE "END" TO SEARCH-FLAG
QA0267                     MOVE "NO" TO CUR-INT-PART.
QA0268      IF SEARCH-FLAG = "YES"
QA0269          IF DBMS-PART-NO = SPACES
QA0270              MOVE SPACES TO SEARCH-FLAG
QA0271          ELSE IF ENTITY-CODE OF DBMS-RCD-INVENTORY = 01
QA0272                  MOVE "END" TO SEARCH-FLAG.
QA0273      IF SEARCH-FLAG = SPACES
QA0274          MOVE "END" TO SEARCH-FLAG
QA0275          MOVE "NO" TO CUR-INT-PART.
QA0393 LOOK-FOR-PURCHASE.
QA0394      READ INTERNAL-INVENTORY NEXT RECORD
QA0395              AT END MOVE "END" TO SEARCH-FLAG
```

```
A0396                  MOVE "NO" TO CUR-INT-PURCHASE.
A0397      IF SEARCH-FLAG = "YES"
A0398          IF DBMS-PURCHASE-ORDER-NO = SPACES
A0399              MOVE SPACES TO SEARCH-FLAG
A0400          ELSE IF ENTITY-CODE OF DBMS-RCD-INVENTORY = 02
A0401                  MOVE "END" TO SEARCH-FLAG.
A0402      IF SEARCH-FLAG = SPACES
A0403          MOVE "END" TO SEARCH-FLAG
A0404          MOVE "NO" TO CUR-INT-PURCHASE.
B0301 SETUP-KEY-PART.
B0302      MOVE DBMS-EXT-PART-NO
B0303          TO DBMS-INT-PART-NO.
B0428 SETUP-KEY-PURCHASE.
B0429      MOVE DBMS-INT-PART
B0430          TO DBMS-INT-PURCHASE.
B0431      MOVE DBMS-EXT-PURCHASE-ORDER-NO
B0432          TO DBMS-INT-PURCHASE-ORDER-NO.
QC0276 CLEAR-INT-PART.
QC0309     MOVE SPACES TO DBMS-INT-DESCRIPTION.
QC0311     MOVE ZEROS TO DBMS-INT-UNIT-PRICE.
QC0313     MOVE ZEROS TO DBMS-INT-STOCK-IN-HAND.
QC0405 CLEAR-INT-PURCHASE.
QC0438     MOVE ZEROS TO DBMS-INT-PURCHASE-QTY.
QC0440     MOVE ZEROS TO DBMS-INT-PURCHASE-DATE.
QC0442     MOVE ZEROS TO DBMS-INT-PURCHASE-PRICE.
QC0444     MOVE SPACES TO DBMS-INT-PURCHASE-SUPPLIER-NO.
QD0277 MAKE-CURR-PART.
QD0278     MOVE "YES" TO CUR-INT-PART.
QD0281     MOVE "NO" TO CUR-INT-PURCHASE.
QD0406 MAKE-CURR-PURCHASE.
QD0407     MOVE "YES" TO CUR-INT-PURCHASE.
QE0279 SET-CURR-PART.
QE0280     MOVE "YES" TO DBMS-CUR-PART.
QE0282     MOVE "NO" TO DBMS-CUR-PURCHASE.
QE0408 SET-CURR-PURCHASE.
QE0409     MOVE "YES" TO DBMS-CUR-PURCHASE.
SA0200 INN-PART.
SA0201     MOVE UWA TO DBMS-EXT-PART.
SA0202 OUT-PART.
SA0203     MOVE DBMS-EXT-PART TO UWA.
SA0329 INN-PURCHASE.
SA0330     MOVE UWA TO DBMS-EXT-PURCHASE.
SA0331 OUT-PURCHASE.
SA0332     MOVE DBMS-EXT-PURCHASE TO UWA.
TA0038 NEW-DATA-BASE.
TA0039     IF DATA-BASE-OPEN-FLAG = "YES"
TA0040         MOVE 101 TO RESULT
TA0041     ELSE
TA0042         PERFORM CREATE-DATE-BASE
TA0043         PERFORM CLOSE-DATA-BASE
TA0044         PERFORM UPDATE-DATA-BASE
TA0045          MOVE "YES" TO DATA-BASE-OPEN-FLAG.
TA0046 OLD-DATA-BASE.
TA0047     IF DATA-BASE-OPEN-FLAG = "YES"
TA0048         MOVE 101 TO RESULT
TA0049     ELSE PERFORM UPDATE-DATA-BASE
TA0050          MOVE "YES" TO DATA-BASE-OPEN-FLAG.
TA0051 RELEASE-DATA-BASE.
TA0052     IF DATA-BASE-OPEN-FLAG = "NO"
TA0053         MOVE 102 TO RESULT
TA0054     ELSE PERFORM CLOSE-DATA-BASE
```

```
A0055             MOVE "NO" TO DATA-BASE-OPEN-FLAG.
B0056 CREATE-DATA-BASE.
B0192      OPEN OUTPUT INTERNAL-INVENTORY.
C0057 UPDATE-DATA-BASE.
C0193      OPEN I-O INTERNAL-INVENTORY.
D0058 CLOSE-DATA-BASE.
D0194      CLOSE INTERNAL-INVENTORY.
A0059 NO-SUCH-FUNCTION.
A0060      MOVE 105 TO RESULT.
A0061 READ-INVALID-KEY.
A0062      MOVE FILE-STATUS TO RESULT.
A0063 READ-AT-END.
A0064      MOVE FILE-STATUS TO RESULT.
A0065 WRITE-INVALID-KEY.
A0066      MOVE 107 TO RESULT.
A0067 DELETE-INVALID-KEY.
A0068      MOVE 108 TO RESULT.
A0069 REWRITE-INVALID-KEY.
A0070      MOVE 109 TO RESULT.
A0071 NO-SUCH-RECORD.
A0072      MOVE 104 TO RESULT.
A0073 START-ERROR.
A0074      MOVE 111 TO RESULT.
```