DETAREL: A GENERALIZED MINICOMPUTER RELATIONAL DATA BASE MANAGEMENT SYSTEM

C. Carlesi, O. Fraccalini, C. Thanos
Istituto di Elaborazione dell'Informazione
Pisa, Italy

ABSTRACT

In this paper we discuss the design and implementation of a mini computer relational data base management system: the DEcision TAble RELational system (DETAREL).
DETAREL is a generalized data base management system based on Codd's relational model of data. It is designed and implemented at the Istituto di Elaborazione della Informazione of the National Research Council and it is running on the HP 2100F computer. It has a decision table interface to enable the uset to specify requests and data manipulation commands without the aid of a programmer. All the facilities are integrated in one high level user oriented language.

The DATAREL system, in addition of being a stand-alone data base management system, gives the basis for the HP 2100 computer to be a node in a distributed data base.

INTRODUCTION

The broadening of data communications and the major advances in hardware and software technology for small machines has pushed the mini computer into the limelight of today's data processing. Representing a tremendous computing capacity at fractions of the cost of their bigger brothers, the mini has now been interfaced with not only a communications capability, but with sufficient external storage to justify the rising interest in its application to the data base system problem. The software developments have provided legitimate operating systems capable of supporting the sophisticated user, and an initial interest for the mini computer is therefore in the stand-alone dbms context. But the cost of the mini makes it a perfect candidate for consideration as a component in a functionally modularized data based data processing environment. This gives rise to such concepts as back end dbms, and data base network. That is, the mini computer is now seen as an intrisic component of the over all computer system configuration, with responsability for execution and general operation of the dbms facilities of that system.
Consequently, in this paper we discuss the design and implementation of a minicomputer relational data base management system: the DEcision TAble RELational system (DETAREL).

There are three major approaches to data base management:
1) The hierchical approach
2) The network approach
3) The relational approach
We believe that the relational approach is best suited to our goals and requirements. The advantages of a relational model for data base management systems have been extensively discussed in the literature [Codd 70, Codd 74, Date 74] and hardly require further elaboration. In choosing the relational model, we were particularly motivated by 1) the high degree of data independence that such a model affords and 2) the possibility of providing a high level and entirely procedure-free facility for data definition, retrieval, update, access control, support of views and integrity verification.

The users of our system can vary including: clerks, secreteries, managers, application programmers and Data base Administrators. It is desirable for the users of a dbms to interact with it in a way most suitable to their taste. Codd identifies different types of languages which have been developed for the query and modification of relational data bases [Codd 1974]. They serve two different types of users:
casual users (nonprogrammers) and
application programmers.
Clerks and managers access the data base via the nonprogrammer interface. Application programmers and Data base Administrators access the data base via the programmer interface, but may also call on the nonprogrammer interface. In the current version of DETAREL the programmer interface is not implemented.

THE NONPROGRAMMER INTERFACE

The nonprogrammer interface is provided to allow casual users access to the data base without the aid of a programmer. The casual user is presented with a simple relational view of the data and a simple decision table language to operate on the data. The decision tables control the flow of his actions on the data. We believe that complicated or unrestricted control structures are not needed. Most business oriented programming logic is rather simple. We propose, therefore, decision tables as the main vehicle for representing such logic. Decision tables have been used extensively in the past in the data processing [SIGPLAN 1971]. We propose to build them in our language as its main feature. The decision table language is a non procedural language which frees the user from concern for how data structures are implemented and what algorithms are operating on stored data. As such it facilitates a considerable degree of data independence. Absent from the language are: cursors, explicit interaction, control structures, variables, quantifiers e.t.c. We believe that this language can provide a nice environment for business applications. The version of the language, which has been implemented, includes facilities for query, insertion, deletion and update of relations [Thanos e Fraccalini 1976]. The system also provides a set of utilities to aid the user, e.g. data entry, report generating and system functions.
A DETAREL program consists of one decision table as illustrated in figure 1.

The FOR statement is used to define the scope of interaction with the data base. We have two types of FOR statement:

a) FOR ALL TUPLES OF ⟨relation name⟩
b) FOR EACH TUPLE OF ⟨relation name⟩ WITH ALL TUPLES OF ⟨relation name 2⟩

MIMI 77- ISMM Third Annual Symposium (Zurich, 1977)

The first type of FOR statement involves only one relation and implies that the following decision table contains only simple conditions i.e., conditions that involve a comparison on a single value. The second type of FOR statement involves two relations and implies that the following decision table contains compound conditions as well as simple conditions, with the restriction that the simple conditions are refering to <relation name 1>. By compound conditions we mean conditions that involve a restriction of a domain of <relation name 2> to be a set of values selected from <relation name 1>. The conditions are the basic qualification facility. It enables the user to choose a subset of a relation based on Boolean combinations of conditions on domain values. As we have seen the conditions may be simple or compound. Conditions are used to determine which rule applies. The condition is stated in the left hand column and its values are given in the rule columns. The conditions are predicates with limited entries Y,N, on blank. Actions are performed within rules in the integer order given. The actions with blank entries are ignored. The actions are data manipulation i.e., invocations of system functions, whose syntax is presented in the Appendix A.
On condition statements are used to detect special situations that may arise at any time during execution. When a condition is detected the associated actions are performed. On conditions have the form:
<On condition> : <action>
Conditions are restricted to those detectable by the hardware-software support of the interpreter plus an end condition to terminate execution.

Each column on the right hand side of the decision table corresponds to a rule which consists of a set of conditions and a set of actions. The rule holds if and only if each condition holds according to its entry in the column (E_{ij} may be N for 'no', Y for 'yes' or blank for 'don't care').Only one rule may hold at a time. When a rule holds the actions indicated and ordered by the integer entries in the action column, O_{kj}, are performed.

EXAMPLES

The facilities of the language will be introduced by two examples against a data base which describes a small company. The data base contains the following tables:
EMPLOYEE (ENO, NAME, UNIT, JOB CODE, TITLE, PRIMARY SKILL, SECONDARY SKILL, SALARY, DEPT)
DEPARTMENT (DEPT, MGR, FLOOR, INVOICED)

Example 1 (Fig. 2)

- List NAME, JOB CODE, TITLE of all those working as electrical engineers or in a position where electrical engineer (skill code = 1130) is the job code of the position.
- List NAME, UNIT, JOB CODE, PRIMARY SKILL for all those whose primary skill does not correspond to the Job code required for the position they fill.
- List NAME of those who have the skill of electrical engineer (skill code = 1130).
- List NAME of those who have the skill of electrical engineer but are occupying a position requiring a different job code.
- Find and identify all those employees who are either skilled as electrical engineers and have an annual salary over 10.000 or are skilled as mechanical engineers (skill code = 1120) and have an annual salary of 11.000 or more.
- Print AVERAGE SALARY for all electrical engineers in the company.
- Find how many people are employed in the company.
- Increase SALARY by 5% to every one in the organization.

Example 2 (Fig. 3)

- List NAME, JOB CODE, SALARY of all those having as Manager ANDERSON
- List NAME of all those working for departments on the second floor.
- LIST NAME of all those working for departments with invoiced over 100.000

DATA DEFINITION FACILITY (DDF)

DETAREL offers facilities for data definition. Data definition in relational systems has three main aspects:
- Specification of the characteristics of data to be stored e.g. the attributes names and data types for each relation;
- changes of the data base after it has been created e.g. insertions of new relations or droppings of relations; and
- definition of alternative "views" which are derived from the stored data. A view is a dynamic "window" of a data base [D. Chamberlin 75].

The DETAREL data definition facility (DDF) covers the first two aspects and partially covers the third. A data base can be created based on its description and destroyed. In addition, after a data base has been created, it may be changed by inserting new permanent relations or by dropping existing stored relations.
An important observation to be made is that the definition of a view is simply a process of deriving a relation from the set of stored relations. A view may be a selected subset of a stored relation, or it may span more than one stored relation, as in the case of a join.

DETAREL offers the ability to define "limited views". These views can be thought of as derived relations (only projections) which cannot be operated on as can stored relations in a data base. Any change of the underlying relation is reflected in these limited views but the views cannot be modified directly, they only can be retrieved.
The syntax of the DDF is presented in the Appendix B

IMPLEMENTATION CONDITIONS

It was felt that a simple approach to the implementation of a mini oriented system should be followed [McLeod e Meldam 1975]. Some of the intricate implementation problems in large relational systems can be avoided in the environment of a mini computer.

Systems Catalogs. The data base description is given in the system catalogs. The system catalogs describe all units of logical data and the relationships that exist among them. In particular the USER DESCRIPTION catalog contains the user-id of all the data bases users and the access rights they have on every relation owned by them. The DATA BASE DESCRIPTION catalog contains a list of the data bases (by name); for every data base it contains a refno and the type (primary relation or view) of each relation. The RELATION DECRIPTION catalog provides information about the structure and content of each relation in the data base [name of the relation, number of tuples in relation, number of attributes in relation, width (in bytes) of a tuple, attribute names, attribute number (position) in relation, data type of attribute (integer or character string), length (in bytes) of attributes, key information (if an attribute is part of the key)

e.t.c]. An catalogs are stored in files provided by the DOS/3.

- Storage structure. Each data base contains two types of components: relations and datatypes. Relations are the most important part of the data base conceptually and contain the interdata relationship information. The data types contain the actual data items that are related by the relations (in the data base). A relation does not contain the data items themselves, but rather contains pointers to the data items in the data types. Thus, for example, the name "A. Rossi" is stored at most once in the data base (in the data type "name"). All occurances of "A. Rossi" are actually pointers to the string "A. Rossi" in the data type "name". Thus, for each relation there is a tuple file containing an entry for each tuple in the relation. Each entry is a list of pointers (possibly null), one for each attribute of the relation. These entries are stored sequentially as fixed - length blocks in the current implementation. The pointers in the tuple file point to the actual data items, which are stored in the attribute value files. Actually, if the data type of the data item is integer, then it is stored in place of the pointer.

- The Decision Table Language Interpreter. The Decision Table Language Interpreter implements the Decision Table Language using an Access Method Interface (AMI) for storage and retrieval of data in the form of n-any relations. The Interpreter must translate the non-procedural Decision Table Language statements into an efficient sequence of tuple-at-a-time low level commands. The Decision Table Language Interpreter is a FORTRAN IV program. Its main sections are a Syntactic Analyzer, a Rule Mark Algorithm and a Translator [Thanos and Fraccalini 1976].

- The Syntactic Analyzer checks syntax and generates a clean form of the Decision Table Language commands for the Translator, as well as data structures for the Rule Mask Algorithm. The data structures created are: a condition matrix containing an entry for every condition in the condition part in the decision table, an action matrix containing an entry for every action in the action part in the decision table, a matrix representing the action entry of the decision table. Moreover there are created two other matrices, the mask matrix and the table matrix. We prefer to create matrix data structures instead of binary trees because they are easier to manipulate.

- The Rule Mask Algorithm. We have selected the Rule Mask Technique for processing decision tables. We will give a short description of this algorithm for a limited-entry decision table.The entries in a limited-entry decision table are ternary. If we want to represent the decision table in binary logic, we have to use two bits for each entry. One way is to use one bit to indicate whether the entry is a dash (0) or not (1), the other bit to indicate wether the entry is a y. Thus we get a mask matrix and a table matrix. The actual processing of the decision table is done in the following way. A binary transaction Vector is built by placing a "1" in each true condition position and a "0" in all other positions. The conjunction of this vector with the first rule column of the mask matrix is built and compared with the first column of the table matrix. The conjunction filters out all non-pertinent entries. The comparison with the table matrix shows whether the pertinent y-entries match. If they do not match, the conjunction and comparison are repeated for the second column and so on. If the decision table is complete we will certainly find a rule which corresponds to the actual conditions. If the decision table is redundant or inconsistent, we might even find more than one rule. When selecting a method for processing decision tables, the objectives are the minimization of the storage requirements for the object program, or the minimization of the average execution time. In terms of storage requirements the rule mask technique is very efficient. In our environment the storage requirement is a critical factor, so we have selected the rule mask technique.

- The Translator makes the semantics checks and generates the data structures for the Access Method Interface: i.e.,it translates the Decision Table Language commands represented by the matrix data structures, as has been mentioned above, into low level commands represented by command control blocks. The system catalogs are used to check that relation and attribute names, and so on, are specified appropriately. These data structures, describing the user and the command type, are built by the Translator and are passed as parameters in a procedure call to the Access Method Interface.

- Access Method Interface (AMI). The AMI handles all actual processing of data from relations. The AMI language is implemented as a set of functions whose calling conventions are indicated below.

The eight implemented calls are as follows:

1) START (data_base_name, user_id)
   The purpose of the START command is to prepare a data base for processing. It enables the user ⟨user _ id⟩ to access the data base ⟨data_base_name⟩.
2) STOP (data_base_name, user _ id)
   The STOP command cleans up processing when access to a data base is no longer desired.
3) OPEN (relation _ name, decriptor, access-mode)
   Before a relation may be accessed it must be opened. This function opens the DOS/3 file for the relation and fills in a descriptor with information about the relation from the RELATION DESCRIPTION catalog. The descriptor is used in subsequent calls on AMI routines as an input parameter to indicate what relation is involved. Consequently the AMI data accessing routines need not themselves check the system catalogs for the description of a relation. Access-mode specifies whether the relation is being opened for update or for retrieval only.
4) CLOSE (relation_name, descriptor)
   This function closes the relation's DOS/3 file and rewrites the information in the descriptor back into the system catalog if there has been any change.
5) GET (descriptor, tuple_id, work_area)
   This function retrieves into "work_area" a single tuple from the relation indicated by "descriptor". Tuple_id is the tuple identifier. The GET function is intended to be called successively to retrieve all tuples of the relation indicated by "descriptor". Each time GET is called the tuple identifier of the next tuple is placed into"tuple_id"in readiness for the next call. Reaching of the last tuple is indicated by a special return code.
6) INSERT (descriptor, work_area)
   This function inserts a new tuple in the relation.
7) REPLACE (descriptor, tuple id, work area)
   This function replaces by new values the tuple indicated by "tuple_id". The tuple identifier of the affected tuple will have been obtained by a previous GET.
8) DELETE (descriptor, tuple-id)

The tuple indicated by "tuple-id" is deleted from the relation altogether. The tuple identifier of the affected tuple will have been obtained by a previous GET.

CONCLUDING REMARKS

In this paper we presented the DEcision TAble RELational system. It is implemented on HP 2100F with DOS/3. It has been shown that a relational data base management system is practical in the mini computer environment. Although some compromises in power and generality were necessary, a use full and cost-effective implementation has been produced. Only a single user program may execute at -a-time. We have tested the system with a number of small data bases. We are currently generating a data base for the loans in the Library of the I- stituto Elaborazione dell'Informazione. At this point, it seems appropriate to mention what appear to be some of the most significant limitations of the DETAREL system:

- Retrieval involving more than two relations is not possible.
- No great optimizing strategies were employed for storage allocation and retrieval. In designing a dbms for such a small computer the main difficulties are presented by the machine limitations.
- This first version of DETAREL is relatively slow.
- It provides an elementary protection.

However, it should be kept in mind that this is a prototype system. It was built as a feasibility study of a mini dbms. In our Institute we build prototypes mainly for research purposes. Other people can use the ideas in commercially oriented systems.
Improvements and additions to the implemented version of the DETAREL will be made, in particular we hope to introduce extended entry decision tables, a macro facility, and to implement inverted files as the basic tool for minimizing tuple retrieval operations in interpreting a query. In addition we plan to implement a version of DETAREL to operate on distributed data bases.

ACKNOWLEDGMENT

We are greatly indebted to Prof. Dennis Tsichritzis of Computer Science Department University of Toronto for his valuable guidance and help.

REFERENCES

1. Thanos, C. and Fraccalini, O. "DETAREL: un linguaggio relazionale per utenti non programmatori", Nota Interna B76-9 Istituto di Elaborazione dell'Informazione del C.N.R., Pisa, Italy, March 1976.

2. Thanos, C. and Fraccalini, O. "Il prototipo di un interprete per il linguaggio DETAREL" Nota Interna B76-6, Istituto Elaborazione dell'Informazione del C.N.R., Pisa, Italy, July 1976.

3. Chamberlin, D.D. and Boyce, R.F. "SEQUEL: a structured english query language" Proc. ACM-SIGMOD Work shop on Data Description, Access and Control, Ann Arbor, Mich., May 1-3, 1974.

4. Astrahan, M.M. and Chamberlin, D.D. "Implementation of a structured english query language" Research Report RJ 1464, IBM Research Laboratory, San Jose, Calif, October 1974.

5. Codd, E.F. "A relational model of data for large shared data banks", Com. ACM, vol. 13 No. 6, June 1970.

6. Codd, E.F. "Recent investigations in data base systems" Information Processing 74 North-Holland, Amsterdam, 1974.

7. Codd, E.F. and Date, C.J. "Interactive support for non programmers: the relational and network approaches", Proc. ACM-SIGMOD Debate on data models: data structures set versus relational, Ann Arbor, Mich, May 1-3, 1974.

8. Date, C.J. and Codd, E.F. "The relational and network approaches: comparison of the application programming interface" Proc. ACM-SIGMOD Debate on data models: data structures set versus relational, Ann Arbor, Mich., May 1-3, 1974.

9. Held, G.D., Stonebraker, M. and Wong. E. "INGRES A Relational data base Management System" Proc. 1975 NCC AFIPS Press, 1975.

10. McLeod, D.J. and Meldam, M.J. "RISS - a generalized minicomputer relational data base management system" Proc. 1975 NCC, AFIPS Press, 1975.

11. Lorie, R.A. "XRM - an extended (n-ary) relational memory" IBM Scientif Center Report G320-2096, Cambridge, Mass, January 1974.

12. Chamberlin, D.D. Gray J.N. and Traigger I.L. "Views, Authorization and Locking in a Relational Data Base System" Proc. 1975 NCC, AFIPS Press. 1975.

13. Schumacher, H. "The synthesis of optimal decision trees from decision tables" M. Sc. Thesis, University of Toronto, Toronto, December 1974.

14. SIGPLAN notices special issue on Decision Tables, vol. 6, No. 8. September 1971.

APPENDIX A

Data Manipulation commands

⟨command⟩ :: {⟨UPDATE PHRASE⟩|⟨INSERT PHRASE⟩|⟨DELETE PHRASE⟩|⟨OUTPUT PHRASE⟩

⟨UPDATE PHRASE⟩ :: = UPDATE ⟨DOMAIN NAME⟩ { BY {⟨VALUE⟩|⟨ STRING⟩|⟨DOMAIN NAME⟩}|{ BY ADDING | BY SUBTRACTING | BY MULTIPLUING BY| BY DIVIDING BY}{⟨VALUE⟩|⟨ DOMAIN NAME⟩}}[, ⟨DOMAIN NAME⟩ { BY{ ⟨VALUE⟩|⟨STRING⟩| ⟨ DOMAIN NAME⟩}| {BY ADDING |BY SUBTRACTING BY MOLTIPLYING BY | BY DIVIDING BY}{ ⟨VALUE⟩|⟨DOMAIN NAME⟩}}]

⟨INSERT PHRASE⟩ :: = INSERT ⟨DOMAIN NAME⟩ = { ⟨VALUE⟩| ⟨STRING⟩}[,⟨DOMAIN NAME⟩ = {⟨VALUE⟩| ⟨STRING⟩}]

⟨DELETE PHRASE⟩ :: = DELETE

⟨OUTPUT PHRASE⟩ :: = OUTPUT [⟨FUNCTION⟩] ⟨DOMAIN NAME⟩ [,[⟨FUNCTION⟩]⟨DOMAIN NAME⟩]

APPENDIX B

Data Definition Facility

The syntax below is intended primarily to give the reader an intuitive understanding of the structure of the DDF

1. ⟨CREATE USER⟩ :: = CR US ⟨USER CODE⟩ ⟨DATA BASE NAME⟩

2. ⟨DELETE USER⟩ :: = D US ⟨USER CODE⟩

3. ⟨CREATE DATA BASE⟩ :: = CR DA ⟨DATA BASE NAME⟩

4. ⟨DELETE DATA BASE⟩ :: = D DA ⟨DATA BASE NAME⟩ [⟨USER CODE⟩]

5. ⟨ASSIGN RELATION⟩ :: = A RE ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩ ⟨USER CODE⟩ ⟨ACCESS MODE⟩

6. ⟨CREATE RELATION⟩:: = CR RE ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩ ⟨ATTRIBUTES NUMBER⟩ ⟨ATTRIBUTE NAME⟩ ⟨ATTRIBUTE TYPE⟩ ⟨ATTRIBUTE LENGTH⟩ ⟨KEY⟩| ⟨INDEXED⟩ [,⟨ATTRIBUTE NAME⟩ ⟨ATTRIBUTE TYPE⟩ ⟨ATTRIBUTE LENGTH⟩ ⟨KEY⟩| ⟨INDEXED⟩]

7. ⟨DELETE RELATION⟩ :: = D D RE ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩

8. ⟨CREATE PROJECTION⟩:: = CR PR ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩ ⟨RELATION NAME⟩ ⟨ATTRIBUTES NUMBER⟩ ⟨ATTRIBUTE NAME⟩ ⟨KEY⟩|⟨INDEXED⟩ [,⟨ATTRIBUTE NAME⟩ ⟨KEY⟩| ⟨INDEXED⟩]

9. ⟨DELETE PROJECTION⟩ :: = D PR ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩

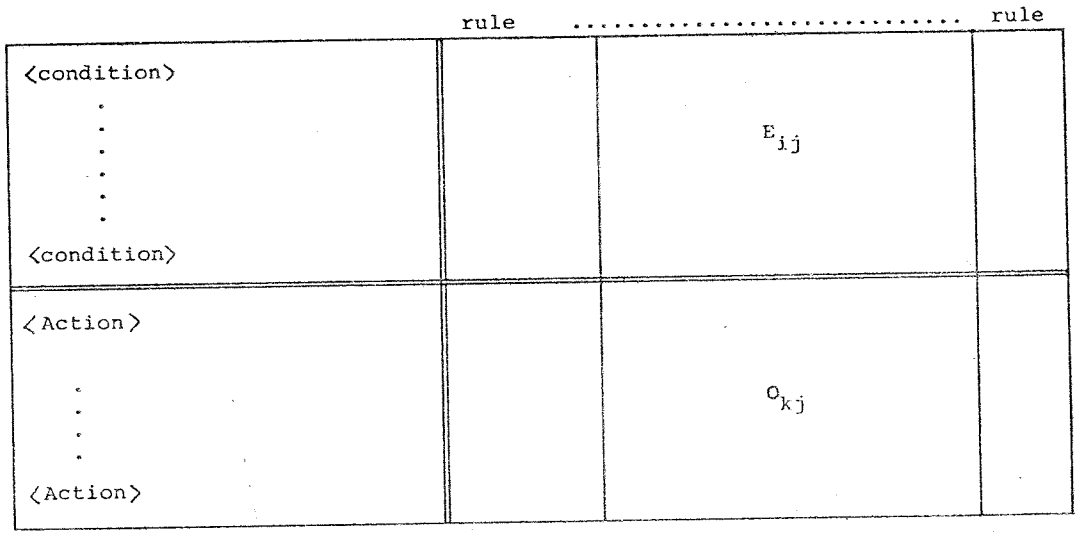
10. ⟨LIST USER⟩ :: = L US ⟨USER CODE⟩

11. ⟨LIST DATA BASE⟩ :: = L DA ⟨DATA BASE NAME⟩| ⟨BLANK⟩

12. ⟨LIST RELATION⟩ :: = L RE ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩

13. ⟨COPY RELATION⟩ :: = C RE ⟨RELATION NAME⟩ ⟨DATA BASE NAME⟩ ⟨RELATION NAME⟩

⟨DATA BASE NAME⟩ ⟨USER CODE⟩
⟨DECISION TABLE NAME⟩
⟨FOR STATEMENT⟩



⟨On condition⟩ ⟨Action⟩

Fig. 1

FOR ALL TUPLES OF EMPLOYEE

ENO ≠ 0	y											
TITLE = ELEC. ENGR.		y										
JOB CODE = 1130			y		N	N						
JOB CODE ≠ PRIMARY SKILL				y								
PRIMARY SKILL = 1130					y		y		y			
SECONDARY SKILL = 1130						y		y		y		
SAL > 10.000									y	y		
SAL ≥ 11.000											y	y
PRIMARY SKILL = 1220											y	
SECONDARY SKILL = 1220												y
LIST NAME,JOB CODE,TITLE		1	1									
LIST NAME,UNIT,JOB CODE, PRIMARY SKILL				1								
LIST TOTAL (ENO)	1											
LIST AVG (SAL)		2	2									
UPDATE SAL BY MULTIPLY BY 0.05	2											
LIST NAME					1	1	1	1				
LIST ENO									1	1	1	1

Fig. 2

FOR EACH TUPLE OF DEPARTMENT WITH ALL TUPLES OF EMPLOYEE

DEPARTMENT MGR=ANDERSON	y		
DEPARTMENT FLOOR = 2		y	
DEPARTMENT INVOICED > 100.000			y
EMPLOYEE.DEPT=DEPARTMENT.DEPT	y	y	y
LIST NAME, JOB CODE, SALARY	1		
LIST NAME		1	1

Fig. 3