ISTITUTO DI ELABORAZIONE DELL'INFORMAZIONE DEL C.N.R. - PISA

A PROPOSED BUSINESS APPLICATIONS GENERATING SYSTEM

M.L. Brodie*, C. Thanos**, D. Tsichritzis*

Nota Interna B74-26

Settembre 1974

* Computer Science Department - University of Toronto

** Istituto di Elaborazione dell'Informazione del C.N.R. - Pisa

Stampato in proprio

I Introduction

A business applications system consists conceptually of three main parts, a data entry system, a data processing activity and a report generating facility. The data entry system enables the clerks to input data directly, or from forms, into the system for processing. The data processing activity consists mainly of applications programs, e.g., inventory control, billing, which operate on the data and produce logical images of reports. Finally, the report generating facility prints out the reports from ready data according to desired format. All these functions together provide most of the needed facilities, especially for a small business.

Currently used programming languages try to provide constructs which enable a programmer to program any part of a business system. As a result they are rather low level, rich and hard to master. Their use requires a high level of expertise which is hard to expect from the casual user. It is desirable, however, to generate the appropriate tools such that casual users and non-programmers can write small business applications. In this manner computer systems can be within reach of small businesses. Small businesses can now afford the cost of minicomputers. The only stumbling block is the high cost of producing or acquiring the right software. [Brodie & Tsichritzis]

At this point, many nice data entry systems are in the market [Feidlman and Bernstein 1973]. We can also assume that the output report generating facility can be packaged in the right manner. The data processing activity, however, will be hard to completely package in the right set of generalized models and parameters. Businesses differ substantially in their needs, assumptions, rules and requirements. Each one has a certain way for "doing business". It is very difficult to produce a general model which fits all businesses even in the same market sector. We propose to leave this part of an application system unspecified, but to provide the right tools for its specification.

Let's assume in our hypothetical environment that all data is entered into input files using a data entry system. Assume also that a report generating facility exists which enables us to print out data from output files in the right format. We will therefore concentrate on that part of the business system which operates on input files and master files and produces output files. We will refer to this activity as data manipulation.

We need two main features in the data manipulation activity. First, we need to obtain from the input and master files the right subset of data and update them accordingly. Second, we need to perform some operations on the data.

We propose to obtain the first feature through implementation of the relational model [Codd 1970]. By manipulating relations and qualifying domains and tuples we can isolate any data we want in our data base for processing. We will not elaborate on the facilities offered by a relational system, or on its advantages [Tsichritzis, Brodie, Schuster 1974, Date 1972]. We just point out that a relational system can be both complete [Codd 1972] and advantageous for the non-expert programmer [Codd & Date 1974].

The second feature of data manipulation is the ability to describe some computation 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 [SIGPLAN 1971] in data processing. We propose to build them in our language as its main feature.

In the following sections you will find a description of a proposed language which combines relational commands and decision tables. We believe that this language can be adequate for generating the right data manipulation programs in a business. If we assume that data entry and report generation are handled by generalized systems separately, this language can provide a nice environment for business application system generation.

There are many issues which are important in a real world business application system, e.g., backup, security. We will not discuss them. We are aware of their importance. Instead, we concentrate mainly on the features of the language.

II Language

1. Philosophy

The goal of the proposed language is to simplify application development while maintaning the clearest possible statement of the programmer's intention. To accomplish this, a methodology for application development is proposed. A main feature of the methodology is the division of responsibility. The application programmer uses a simplified language which attempts to avoid technical and programming language details allowing concentration on the problem to be solved. The technical aspects are handled by the Data Base Administrator (DBA). This division is made in the languages offered, both of which are oriented to the relational model of data. The system is based on a simple relational data base management system [Brodie 1973].

The DBA is responsible for technical aspects; data definition, input of data and the development of application primitives such as programs to produce reports. For data definition, input and output the DBA may use any combination of the self-contained language, a host language with embedded commands (see appendix I) and entry or report packages.

The programmer is presented with a simple view of the relational data for the application and a simple decision table language to operate on the data base. The language consists of conditions and actions including four data base manipulation commands. The actions are either simple statements or invocations of application primitives developed by the DBA at the programmer's request. Absent from the language are: cursors, explicit iteration, control structures, data I/O and unstructured branching. The programmer may deal directly with the data base, greatly reducing the need to introduce new variables.

A program consists of one main decision table which may be supported by several secondary decision tables. A program is developed in the following way: A For statement is used to define the scope of interaction with the data base by selecting a specific set of tuples. Following this there is a decision table with conditions and actions which refer to one tuple only, by its data base names. The table is implicitly repeated for each tuple in the selected set.

For a set of given situations PL/1 - like on conditions may be used to interupt normal execution of the table for some alternative actions.

A decision table may involve other decision tables to process on conditions or to perform other well defined actions in the same way procedures are used in a well structured program. These tables must be defined as being local to a program which may Invoke it. The invoked table may use Stop to halt all processing, Invoke to invoke a new decision table or Resume to return control to the decision table that invoked it.

An application consists of one or more sequentially combined programs. Flow of control goes through each program (main and secondary decision tables) and then sequentially to the next program starting with its main decision table. Flow of control can be altered by the control actions Stop, Invoke and Resume.

To complete an application the DBA must provide all the application primitives requested. In order to run an application the DBA must input all the required data into the appropriate relations. At the end of each execution a program report is produced. It includes: user and system messages, job and data base status, program name and run time as well as other operational statistics.

The above methodology simplifies the task of the business application programmer by providing an uncomplicated framework for program development. However, the job of the DBA is not simple; it is the crucial part of the entire system.

2. Decision Table Language

This language deals with relations or subrelations, a tuple at a time. Within a subrelation,different sets of conditions (rules) apply causing selected actions to be performed on the appropriate set of tuples.

The features of the language are now illustrated and then described individually.

```
<Table name> [local to <Table name>*]
  [<declaration>*]
  [For [each          ] <relation name> [<where clause>]]
       [all           ]
       [at most n of  ]
       [one of        ]
```

	rule .... rule
<condition> ⋮ <condition>	E_{ij}
<action> ⋮ <action>	O_{kj}

[<on condition>*]

Figure I Diagram of a Decision Table

2.1 Table name

Each decision table must be named uniquely within an application. A table may be declared as local to another table which may in turn Invoke the former table.

2.2 For statement

The For statement selects the tuples that relate to the program. First the source relation is given. This may be qualified by a where clause [Schuster 1974, Boyce et.al. 1973] and the result may be restricted in number by the phrases each, at most n or one. Implicit in the

For statement is that the following decision table is executed once for each tuple in the selected set. When the For statement is omitted the table is executed once. Example: For each Receipt_card where Quantity_ordered>0.

2.3 Declarations and Data Base Names

Declarations are required to introduce variables, not in the data base, but into the decision table. These variables must be defined to be of a type of one of the data base domains. Although not generally needed, these variables may be used as temporaries or to improve readability. Variables introduced by declarations may be initialized. example: Declare Part-total integer initially (0).

Data base names may be used directly in decision tables. They must be qualified, using PL/1 dot notation, only when ambiguity would otherwise occur.

e.g. Project . Quantity
Current . Quantity

2.4 On Conditions

On conditions are used to detect special conditions that may arise at any time during execution. When a condition is detected, the associated actions are performed (as in PL/1 on conditions).

Syntax On <condition> : <action> *

Conditions are restricted to those detectable by the hardware-software support of the interpreter plus a beginning condition and an end condition. Actions may be those from the remainder of the cycle plus Stop which halts all execution and Resume which returns control to the decision table which invoked the current one.

When control returns to a table all its on conditions are re-evaluated and then processing of tuples resumes where it left off.

Examples: On beginning: Order Employee by Empl_#

On end : Print Status_report
Stop

On end : Resume Payroll

On Zerodivide: Print message 'Zero divide'
Stop

2.5 Rules

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 (Eij 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.

Example: language construct and equivalent decision table

```
if C1 & C2 & C3 then
   do
      A1
      A4
      A3
   end
```

	rule
C1	Y
C2	N
C3	Y
C4	
A1	1
A2	
A3	3
A4	2
A5	

2.6 Conditions

Conditions are used to determine which rule applies for the current tuple being processed. The condition is stated in the left hand column and its values are given in the rule columns. The conditions may be Boolean with limited entries Y, N or blank. Syntax for limited entry conditions is taken from SP/K [Holt and Wortman

1973]. The conditions may also evaluate to values from a defined domain; these extended entry conditions may be expressions or calls to application primitives.

Example:

Credit > 500	Y	N	Y	N		Y
Balance - Debit ≤ 0	Y	N	N			
Adjustment_type	overdraft	deposit	withdrawl			credit
Overtime = 0	Y	Y	Y	N		N
Transaction_code	01	45	32	17	03	

2.6.1 Use of Conditions

The major programming tool in this language is selection which occurs in two forms. The For statement reduces the scope of interaction with the data base to a set of tuples. The rules are then used to factor those tuples into cases for processing. The choice of how to select sets and subsets of tuples must be made carefully. Each condition expressed in the where clause can be expressed as a condition within a rule; however, the reverse is not true (e.g. extended entry conditions must be expressed within rules). In business applications conditions in the where clause of the For statement are simple and few in number while those in the decision table, those used to compose rules, are complicated and numerous. The following example indicates the complicated cases that decision tables can handle.

Example:

For all Employees where Pay_period = weekly

	rule1	rule2	rule3	rule4	rule5	etc.
Salary_range	100	100	150	200	250	
Union_deduction	Y	N	Y	Y	N	
Pension_deduction	Y	N	Y	Y	Y	
Savings_bond_deduction	Y	Y	Y	Y	Y	
Credit_union_deduction	N	N	Y	N	N	
etc.						

2.7 Actions

Actions are performed within rules in the integer order given. Those actions with blank entries are ignored. The five types of actions are described below.

2.7.1 Assignment

Syntax: variable := expression

Syntax for the expression is that of SP/K [Holt and Wortman 1973.] Primarily, it allows arithmetic and the invocation of application primitives.

2.7.2 Invocation of Application Primitives

Syntax: [Invoke] <application primitive name> [(<parameter list>)]

An application primitive is any decision table or any function, procedure or program which is available in the library. They may be invoked as single actions or within an assignment as a part of an action. Invoke permits nesting of decision tables to any level. Resume (2.7.4) provides a return mechanism.

Examples:

Overtime-pay := Calculate_overtime (Overtime)

Print Status_report (format parameters)

Invoke Payroll_adjustment (Empl_#)

2.7.3 Data Manipulation Commands

Syntax: <command> <relation name> [with key <attribute name>*] where <command> is Get, Update, Put or Delete.

Each of these commands refers to one tuple only, whether in the data base or in the decision table. In the decision table there is one tuple from each relation. Implicitly, these tuples are the source of Put and Update and the destination of Get. Attribute names in the decision table refer to the tuples in the decision table. Update and Put may use parenthesized values which must be given in the order defined in the relation definition.

The power of the For statement makes these keyed retrievals sufficient for any data manipulation.

Examples:

Get Employee with key Empl_#

Employee.Bonus : = 100

Update Employee

Delete Employee with key Empl-#

Put Employee (073,'Smith, John',0,0,0)

2.7.4 Control Statements

Two statements Stop and Resume can be used only with On conditions. Stop halts execution of the entire application. Resume returns control to the decision table which invoked the current one.

Example: On end: Stop

On end: Resume Payroll

2.7.5 <u>Message Statement</u>

Syntax: <u>Print</u> <u>message</u> <parameter list>

The syntax for the parameter list is taken from SP/K.

This statement is used to print short messages on the program report. This facility is useful for testing, debugging and application development.

Example: On exception_Condition: <u>Print</u> <u>message</u> 'Exception Found:', Value

2.8 <u>Comments</u>

Comments may occur within /*, */ on complete lines only, anywhere in the decision table.

2.9 <u>Relation To Other Language Features</u>

The language is complete with respect to the following control structures: the <u>case</u> statement, if then [<u>else</u>], <u>do</u> <u>while</u> or <u>cycle</u>, iteration, sequence and on conditions. These features are implicit in the decision table forcing the programmer to use them to develop the program in a structured way.

Recently E.W. Dijkstra proposed two guarded command structures for programming languages. Within two types of brackets: if....... <u>end</u>, and <u>do</u> ..... <u>end</u>, there are lists of statements which are executed if the guard condition preceding the statement list is true. This type of structure is very similar to decision tables. Sets of conditions are the same as guards and a statement list is the same as a set of ordered actions within a rule.

Example: guarded construct and equivalent decision table

```
do     guard0
   guard1  : actions1
   guard2  : actions2
     :
   guardn  : actionsn
od
```

For ... where guard0

guard1	Y	Y	...	N
guard2	N	Y		Y
:				
guardn	N	N		Y
actions1	1	1		
actions2		2		
:				
actionsn				1

3. DBA Language

The self-contained DBA language has four functions: data definition, input, output and data manipulation. These commands may also be used as external procedure calls from a host language.

3.1 Data Definition

Data definition facilitates the description of new data components and the alteration of existing ones. There are four commands in this regard.

3.1.1 Declare/Drop Domain

This command adds or deletes a domain to or from the schema. Domain values must be given explicitly in terms of domains that exist already. Five domains or user defined data types initially exist: numeric or float, integer, picture, string and character

3.1.2 Declare/Drop Relation

This command adds or deletes a relation description. For each relation the following is required: a unique name, attribute and domain names and the key must be given.

Example:

Relation Planning with attributes

Item_#	integer	key
QOH	integer	
ON_order	integer	
Order_point	string	

3.1.3 Declare/Drop Index

This command tells the data base which attributes are to be indexed, for performance reasons.

Example: Declare Index Planning.Item_#

3.1.4 Declare/Drop User

This command names a user and gives his capabilities. The command is part of the security and integrity features of the system.

3.2 Input and Output

The I/O commands are used to read data into relations and to print data from relations created by an application program or primitive. The commands involve reading data, deleting data and writing data. Packages for data entry and report generation already exist for these purposes.

3.3 Data Manipulation

All the data manipulation commands, listed in Appendix I are available to the DBA. They are used primarily to create the application primitives requested by the programmers. For example, Join, Order and Project can be used to create reports. Another function of the data manipulation commands is to test the data base in the development of the data base design and in the design of application.

4. An Inventory Control application

A common business application has been developed using the proposed system. Although the data manipulation syntax is used an alternative syntax to that of the DBA (section II.3) is offered. A balance forward inventory system is presented. The method has complete status information on master cards and inventory transactions on transaction cards. Five relations are used to represent the system in the relational data base (Fig. II).

Master card information has been seperated into two relations On_hand and Planning. On_hand contains current status of inventory items while Planning contains information regarding future transactions.

Transaction_card is a relation each tuple of which describes an inventory transaction. Outstanding conditions are stored in Transaction_Register. Finally, a Status_report relation contains all information for output purposes.

Let us assume that data has been entered into the relations (via a data entry program), so that the data base reflects the current state of the inventory. The required data manipulations, as described by the tuples of Transaction_card, may be performed by executing the application Inventory updating (Fig. III). For each Transaction_card tuple, one rule is selected by the code and type of the transaction and the selected actions are executed.

Each transaction type or rule involves a particular update of On-hand and/or Planning. As this is done other conditions such as: below order point and overshipped are noted in the Transaction_-register.

The DBA has used primitives such as JOIN to construct the Status report relation which is the final action of the application.

elation	Planning	with attributes		
		Item_#	numeric	key
		QOH	numeric	
		On_order	numeric	
		available	numeric	
		Order_point	numeric	
elation	On_hand	with attributes		
		Item_#	numeric	key
		Item_description	string	
		Opening_balance	numeric	
		Receipts	numeric	
		Issues	numeric	
		Adjustments	numeric	
lation	Transaction_card	with attributes		
		Item_#	numeric	key
		Transaction-code	numeric	
		Quantity	numeric	
		Type	string	
lation	Transaction_register	with attributes		
		Item_#	numeric	key
		Quantity_overshipped	numeric	
		Below_order_point	numeric	
lation	Status_report	with attributes		
		Item_#	numeric	key
		Item_description	string	
		QOH	numeric	
		On_order	numeric	
		Quantity_received	numeric	
		Quantity_issued	numeric	
		Quantity_adjusted	numeric	
		Quantity_overshipped	numeric	
		Below_order_point	numeric	

Figure II Inventory Control Data Base Description

Domains	Adjustments	numeric
	Available	numeric
	Below-order-point	numeric
	Issues	numeric
	Item-description	string
	Item-#	numeric
	On-Order	numeric
	Opening-balance	numeric
	Order-point	numeric
	QOH	numeric
	Quality-adjusted	numeric
	Quantity-issued	numeric
	Quantity-overshipped	numeric
	Quantity-received	numeric
	Receipts	numeric
	Transaction-code	numeric
	Type	string

Relations

Planning (Item-#, QOH, On-Order, Available, Order-point)
Key (Item-#)

On-hand (Item-#, Item-description, Opening-balance, Receipts, Issues, Adjustments)
Key (Item-#)

Transaction-card (Item-#, Transaction-code, Quantity, Type)
Key (Item-#)

Transaction-register (Item-#, Quantity-overshilied, Below-order-point)
Key (Item-#)

Status-report (Item-#, Item-description, QOH,
On-Order, Quantity-received, Quantity-
-issued, Quantity-adjusted, Quantity-
-overshipped, Below-order-point)
Key (Item-#)

Figure II-a Inventory Control Data Base Description

(An alternative syntax)

```
Inventory_updating :

    /* This application processes the following transactions on the inventory data base:  */
    /*         1 put an item on order,                                                     */
    /*         2 item received, raise stock level,                                         */
    /*         3 itel sold, lower stock level,                                             */
    /*         4 adjustment                                                                */
    /* Exceptions requiring other actions; below order point and overshipped are noted.    */
    /*Finally, the total volume of goods is printed and a clean up table is invoked.       */
    /* Volume is an example of a decision table variable.                                  */
    Declare Volume numeric initially (0)

For each Transaction_card
```

Transaction_code Type	1	2	3	4 damaged goods	4 cancel	4 return	ELSE
Get Planning with Key Item_#	1	1	1	1	1	1	
Get On_hand with Key Item_#		2	2	2		2	
Planning.On_order : = Planning.On_order + Quantity	2						
Planning.On_order : = Planning.On_order - Quantity		3			2		
Planning.Available : = Planning.Available + Quantity	3					3	
Planning.Available : = Planning.Available - Quantity			3	3	3		
Planning.QOH : = Planning.QOH + Quantity		4				4	
Planning.QOH : = Planning.QOH - Quantity			4	4			
On_hand.Receipts : = On_hand.Receipts + Quantity		5					
On_hand.Issues : = On_hand.Issues + Quantity			5				
On_hand.Adjustments: = On_hand.Adjustment + Quantity				5		5	
Put Transaction_register (Item_#, Quantity, 0)						6	
Update Planning with Key Item_#	4	6	6	6	4	7	
Update On_hand with Key Item_#		7	7	7		8	
Volume : = Volume + Quantity	5	8	8	8	5	9	
Print message 'Data error in the following tuple'							1
Write Transaction_card with Key Item_#							2

```
On end : Print message 'Total volume was:', Volume
         Invoke Inventory_control
```

Figure III <u>Data Manipulation Decision Table for Inventory Control</u>

(continued next page)

Inventory_control local to Inventory_updating :

```
/* This decision table examines the data base after the transactions of Inventory_updating. */
/* Planning is examined for conditions that require action.                                  */
/* These conditions are noted in the Transaction_register or on Transaction_cards.          */
/* Finally, a Status report is printed.                                                      */
```

For all of planning

On_order < 0 Available - Order_point ≤ 0	YES	NO YES	NO NO
Put Transaction_register (Item_#, On_order, 0)	1		
On_order := 0	2		
Update Planning with Key Item_#	3		
Put Transaction_card (Item_#, 2, On_order, ' ')			1
Put Transaction_register (Item_#, 0, Available - Order_point)		1	

On end : Print Status_report (format parameters)
Stop

igure III (cont.) Data Manipulation Decisione Table for Inventory Control

III Conclusions

We have proposed a self contained language which can provide a nice environment for business applications. Many features of the language are tentative. We may want to change the details in another iteration. An interpreter for such a language will not be difficult to implement, provided there are available a reasonable able host language and a relational system.

A business applications generating system can be based on such a language. The concept can be applicable on any size of machines. However, we believe it will be especially appropriate for small computers and small business systems. An implementation of the language on a machine like a PDP-11/45 or HP 3000 can be a very useful vehicle for future experimentation.

IV References

[Brodie 1973]

Brodie, M.L.,A Relational Data Base Management System,M.Sc thesis,Department of Computer Science, University of Toronto, October 1973.

[Brodie and Tsichritzis 1973]

Brodie, M.L., D. Tsichritzis "Small Business Systems" Proceeding of CIPS '73 Conference, Edmonton Alberta, Canada.

[Codd 1970, 1971, 1972]

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", Comm. of the ACM vol. 13, No. 6 (June 1970) pp. 377-387.

Codd, E.F., "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia 6, Data Base Systems, New York City,May 24-25 (To be published Prentice-Hall)

[Codd and Date 1974]

Codd, E.F., C.J. Date,"Interactive Support for Non-Programmers: The Relational and Network Approaches", Proceedings SIGFIDET 1974 workshop, Ann Arbor Michigan.

[Date 1972]

Date, C.J., "Relational Data Base Systems: a tutorial", Proc. COINS-72, Miami Florida, December 1972.

[Feidelman 1973]

Feidelman, L., C.B. Bernstein, "Advances In Data Entry". Datamation Vol. 19, No. 3 March 1973.

[Holt and Wortman 1973]

Holt R.C., D.B., Wortman Structured Subsets of the PL/1 Language, TR-27, Computer Systems Research Group, University of Toronto.

[Schuster 1973]

Schuster, S.A., A Relational Specification Sublanguage ZETA project working paper, University of Toronto, December 1973.

[SIGPLAN 1971]

SIGPLAN notices special issue on Decision Tables, volume 6, number 8 (September 1971).

[Tsichritzis, Brodie and Schuster 1974]

Tsichritzis, D., M.L. Brodie, S.A. Schuster "A Case For the Relational Data Base", To appear CIPS magazine, Canadian Information Processing Society, Toronto, Canada (August 1974).

[Moris 1972]

Bracchi, G., A.Fedeli and P. Paoline, "A Relational Data Base Management System", Proc, ACM 1972 Annual Conference 1972.

[Boyce et al 1973]

Boyce, R.F., D.D. Chamberlin, M.M. Hammer, W.F. King, "Specifying Queries as Relational Expressions", Proc. of ACM SIGPLAN-SIGIR interface meeting, Gaithersbury, Maryland, November 1973.

[Gamma- 0 1973]

Bjorner, D., E.F. Codd, K.L. Deckert, I.L. Traiger, "The Gamma-0 n-any Relational Data Base: Interface Specifications of Objects and Operations", IBM Research report RJ 1200, San Jose California, April 1973.

Appendix I

Commands of the Relational Data Base Management System

The language is based on a simple relational data base management system which supports the following commands:

Data Manipulation Commands

1. GET
2. UPDATE
3. PUT
4. DELETE

(1–4: referring to a keyed tuple)

5. For [all of | at most n of | one of] <subrelation>
6. JOIN
7. ORDER
8. PROJECT

Data Definition Commands:

9. Define/Drop Domain
10. Define/Drop Relation
11. Define/Drop Index
12. Define/Drop User
13. Read Tuple
14. Write Tuple

Several systems which supplies these kinds of commands on a relational Data base have proposed [Brodie 1973,Moris 1972 GAMMA-0 1973].