



INTEGRATION OF THE DECISION TABLE FORMALISM WITH A RELATIONAL DATABASE ENVIRONMENT[†]

J. VANTHIENEN and G. WETS

Katholieke Universiteit Leuven, Department of Applied Economic Sciences, Naamsestraat 69, B-3000 Leuven, Belgium

(Received 25 January, in final revised form 3 April 1995)

Abstract— In this paper it is examined how the relational approach may be used, not only in storing, but also in constructing, filling in, validating and consulting decision tables. This integration of a decision table system and the relational concept is studied in the context of an existing decision table engineering workbench, PROLOGA (PROcedural LOGic Analyzer), an interactive rule-based design tool for decision table construction and manipulation.

Key words: Decision Tables, Relational Databases, Knowledge Based Systems, Verification and Validation, Knowledge Acquisition

1. INTRODUCTION

Recent years have shown a lot of research about the integration of knowledge based systems and databases, e.g. adding knowledge based facilities to traditional database management systems, or storing and manipulating knowledge using proven database techniques [5]. It has been reported earlier that knowledge, in the form of clauses [14], or in the form of decision tables [12], [15], can be stored in a relational database [2].

Storing and evaluating clauses (whether or not decision table based), however, is only one aspect. In this paper it is examined how the relational approach may be used, not only in storing, but also in constructing, filling in, validating and consulting decision tables. This integration of a decision table system and the relational concept is elaborated in the context of an existing decision table engineering workbench.

In section 2 the decision table representation is briefly described. Section 3 analyzes how knowledge in the form of rules or decision tables can be stored in a relational database system. In the main part of this paper (section 4), the major modelling issues of decision table construction including extended verification & validation and consultation are examined, based on the correspondence between a decision table and a table in a relational database system. A possible integration of a decision table tool and a relational database system is presented in section 5. Finally our major findings are summarized.

2. KNOWLEDGE REPRESENTATION USING DECISION TABLES

2.1 Definition

A decision table consists of four parts [1]:

1. The condition subjects are the criteria which are relevant to the decision making process. They represent the items about which information is needed to take the right decision. Condition subjects are found in the upper left part of the table.

[†] Recommended by Peri Loucopoulos

2. The condition states are logical expressions determining the relevant sets of values for a given condition. Every condition has its set of condition states. Condition states are found at the right hand side of the table.
3. The action subjects describe the results of the decision making process. They are found in the lower left part of the table.
4. The action values are the possible values a given action can take. They are found at the right hand side of the table.

These four parts can be defined more formally:

- $CS = \{CS_i\}$ ($i=1..cnum$) is the set of condition subjects;
- $CD = \{CD_i\}$ ($i=1..cnum$) is the set of condition domains, with CD_i the domain of condition i , i.e. the set of all possible values of condition subject CS_i ;
- $CT = \{CT_i\}$ ($i=1..cnum$) is the set of condition state sets, with $CT_i = \{S_{ik}\}$ ($k=1..n_i$) an ordered set of n_i condition states S_{ik} . Each condition state S_{ik} is a logical expression concerning the elements of CD_i , that determines a subset of CD_i , such that the set of all these subsets constitutes a partition of CD_i (completeness and exclusivity of the condition states);
- $AS = \{AS_j\}$ ($j=1..anum$) is the set of action subjects;
- $AV = \{AV_j\}$ ($j=1..anum$) is the set of action value sets, with $AV_j = \{\text{true (x), false (-), null (.)}\}$ the set of action values¹, which is, in first instance, null for every action subject, for reasons of consistency checking.

The decision table DT is a function from the Cartesian product of the condition states to the Cartesian product of the action values, by which every condition combination is mapped into one (completeness) and only one (exclusivity) action configuration.

$$DT: CT_1 \times CT_2 \times \dots \times CT_{cnum} \rightarrow AV_1 \times AV_2 \times \dots \times AV_{anum}$$

If each column only contains simple states (no contractions or irrelevant conditions, this is shown in the table as a '-'), the table is called an *expanded* decision table (*canonical form*), in the other case, the table is called a *contracted* decision table (*consolidated form*).

The condition subjects and action subjects can refer to other tables (*subtables*). Two types of subtables are possible: the action subtable, i.e. a further specification of a certain action, and the condition subtable, determining the value of a condition.

Figure 1 shows an example of a system of decision tables. A condition or an action which is worked out in detail in a subtable is preceded by the subtable sign '»'.

Types of decision tables

Many variations of the decision table concept exist which look similar at first sight [21]. In practice, one has to distinguish between some major kinds of tables, with the decision grid chart at one end of the spectrum and the real decision table at the other end.

The most important criterion when distinguishing tables, is the question whether all columns are mutually exclusive (*single hit* versus *multiple hit*). In a single hit table each possible combination of conditions can be found in exactly one (one and only one) column. This makes an unambiguous use of the table possible.

¹A '-' value in the condition part means irrelevant. In the action part it means don't execute. A null value means unknown.

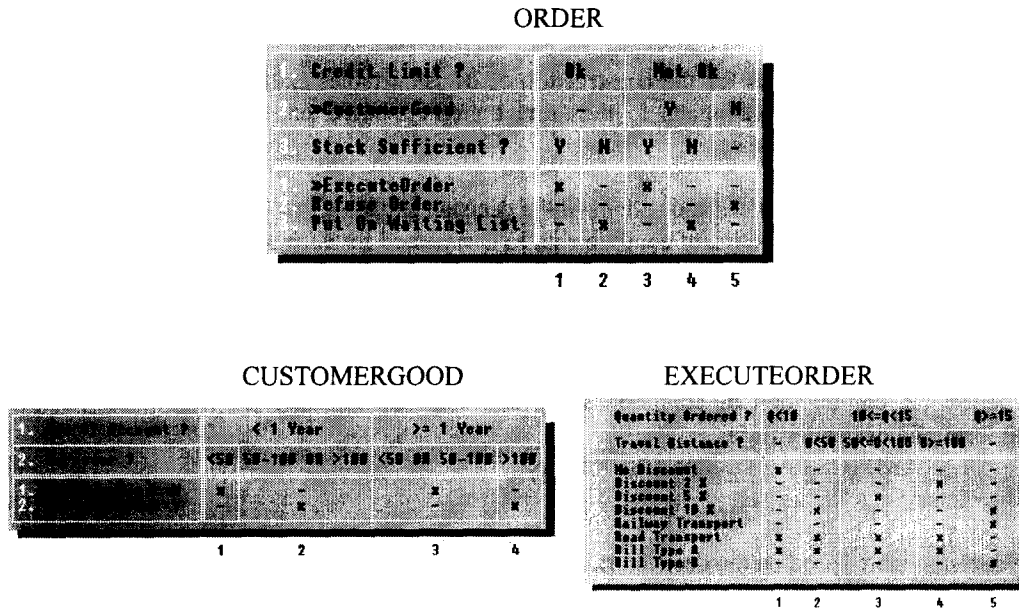


Fig. 1. Decision table representation

If the columns are not exclusive, some combination of conditions is present in more than one column, which may lead to inconsistency. When consulting the table, the first hit rule will often be used. This rule states that the *first hit* (from left to right) will determine which set of actions has to be executed, thus preventing contradictions.

Another possibility is that *all hits* are used to determine the set of actions to be executed. In this case, each hit from left to right can add actions (not mentioned by previous columns) or delete actions (overwriting previous columns) to the set. An interesting concept of this latter form is the so called decision grid chart (a tabular (action by action) representation of a set of decision rules. The decision grid chart corresponds with the rule base of a knowledge base system [15].

In both multiple hit cases (first hit versus all hits) the same combination of conditions can occur in different columns. As a result the overview over the columns is lost, and with it, the simplicity of inspection. For these reasons we do not consider these tables to be real decision tables.

2.2. Decision Tables in Knowledge Based Systems

Knowledge acquisition is one of the main problems in building knowledge based systems, and usually, after the knowledge acquisition process is finished, a lot of contradictions and insufficiencies remain to be detected and solved. Also, maintaining the knowledge base is not a trivial task which often introduces unnoticed inconsistencies or contradictions. Therefore verification and validation of knowledge based systems are receiving increased attention [6]. The emerging problems of validation and verification have led to the occasional use of schemes, tables or similar techniques in knowledge representation and validation. It has been reported earlier [4], [15], that, in a vast majority of cases, the decision table technique is able to provide for extensive validation and verification assistance.

Besides verification and validation there are two other domains where decision tables are used. In the knowledge acquisition stage, knowledge can be modelled by a system of decision tables, [13], [16]. In the implementation phase knowledge bases can be transformed to decision tables to obtain faster execution of the knowledge base [3].

2.3. Construction of Decision Tables

A number of *manual* methods for the construction of decision tables exist [22]. In this paper the construction process proceeds analogous to the "direct method based on simple rules" [16]. The following *stages* can be distinguished:

1. obtain conditions, condition states and actions of the decision situation;
When conditions are defined, a list of condition states is expected for each condition. Actions have a standard set of action values (cf. *supra*), only the action subjects must be defined.
2. specify the problem in terms of decision rules;
During this phase, the problem is described as a series of production rules where the connection is made between a combination of condition states and some actions that must be executed. The general form of a decision rule is:
IF $CS_1 = S_{1k}$ AND ($CS_2 = S_{2m}$ OR $CS_3 = S_{3m}$)
THEN action AS_j AND ...
3. fill the decision table based on the decision rules;
First, the empty expanded table is drawn, then the condition entries are filled, finally based on the decision rules the action entries are filled.
4. check for completeness, consistency, and correctness;
After the decision tables are constructed, they will be checked on:
completeness (for each combination of conditions states there is at least one action configuration);
consistency (for each combination of condition states there is only one action configuration);
correctness (does the decision table express what was meant by the designer. If not, the decision rules need to be adapted).
5. simplify the decision table and display it.
In this step decision tables can be simplified. In this process columns with the same action configuration will be contracted [15].

Manually constructing decision tables can be time-consuming, therefore in section 4 of this paper it will be explained how the construction of decision tables can be automated. Also it will be illustrated how this construction process can be supported by the relational approach.

3. MANAGING DECISION TABLE KNOWLEDGE IN A RELATIONAL DATABASE SYSTEM

When integrating decision tables with relational databases, one has two options. The first option is to use the relational environment only as a reliable means to store the decision tables, which have already been constructed either manually or by some form of computer aided construction. In the second option, the relational environment is not only used as a means to store decision tables, but also as an aid to the construction process of the decision tables. In this section the first option is elaborated, while the second option will be explained in the next section.

When it was decided to choose the relational environment as an efficient storing mechanism, one has to decide what to store in the relational database? Will we store the decision rules (which are used to construct the decision table, cf. *supra*), the resulting decision table or both?

Irrespective of the answer to this question, we need a technique to store production rules. If we look at a constructed decision table column by column, a decision table is equal to a set of production rules and these rules are non overlapping. Thus, an approach used to store production rules can also be used to store decision tables.

In the next paragraphs two techniques for storing production rules in a relational database system will be described. Then these techniques will be used to store decision tables.

3.1 Storing Production Rules

The basic form of a rule expressed by production rules is: IF *premise* \rightarrow *conclusion*

Both the premise part as the conclusion part could have been connected with the logical operators AND or OR. If this is the case, those rules have to be converted to their disjunctive normal form. In this form each rule consists of a disjunction of terms, while each terms consists of a conjunction of subjects. Next, two techniques to store production rules are explained. These techniques will be illustrated with the following two rules, which are already in disjunctive normal form:

if $(P1 = 1 \wedge P2 = 3) \vee (P1 = 1 \wedge P3 = 4) \rightarrow C1 = 6 \wedge C2 = 5$

if $P1 = 2 \wedge P3 = 5 \rightarrow C1 = 2 \wedge C3 = 3$

Technique 1

The first technique to store production rules has been presented by numerous authors e.g. [7], [14]. In this technique a relational table is created, with as columns the different subjects which occur in a rule. Each rule is stored as a different tuple. A production rule with an OR-operator will be decomposed in two or more tuples. The primary key of each tuple consists of all the subjects occurring in the rule. It is not enough to include only the subjects which occur in the IF-part of the rule in the primary key since the THEN-part of the rule may determine different conclusions.

In te remainder of this paper the primary key of the relation is indicated by a solid background in the heading of the table. A foreign key of the relation is denoted by an up diagonal shading.

1	P2	P3	C1	C2	C3
1	3	-	6	5	-
1	-	4	6	5	-
2	-	5	2	-	3

 Primary Key

Table 1. Example rules stored in relational database

Technique 2

In the previous technique, the number of columns of the relational table in which the rules are stored is equal to the total number of subjects which can occur in a rule. If most of the rules use only a few subjects, a lot of space is wasted. Therefore a more problem independent approach to store production rules may be chosen.

To use this approach three relational tables (SUBJECT, RULE, RULEPART) have to be created: In the table SUBJECT all possible subjects which can occur in a rule are stored together with their type (premise (P) or conclusion (C)). In table RULE some characteristics of the rules are stored. E.g. when was the rule created, which expert proposed the rule, Finally in the table RULEPART, each rule is stored using one row for each term in the rule. In this row, the number of the rule is stored together with the subject name and the value of the subject.

Using this technique, the example rules will be stored as follows in a relational table:

SubjectName	Type
P1	P
P2	P
P3	P
C1	C
C2	C
C3	C

Table 2. Table SUBJECT

RuleN
1
2
3

Table 3. Table RULE

RuleNo	SubjectName	State
1	P1	1
1	P2	3
1	C1	6
1	C2	5
2	P1	1
2	P3	4
2	C1	6
2	C2	5
3	P1	2
3	P3	5
3	C1	2
3	C3	3

Table 4. Table RULEPART

3.2. Storing a Decision Table

As already mentioned, a decision table is equal to a set of non overlapping production rules. Therefore the two techniques to store production rules, which were explained above are now used to store decision tables.

In the first technique the decision table is considered as a relation and therefore stored as a relational table. Each column of a decision table will be represented as a row in a relational table. In this case the decision table format can be preserved. However, the catalog does not stay fixed because each time a decision table is added to the system, a new relational table has to be created.

When the decision table is stored using technique 2, a new tuple is used for each non irrelevant condition entry in a decision column. Storing the decision tables this way has the advantage that it is not necessary to create a new relational table each time a decision table is added.

Notice that the names of the decision tables, the subjects (conditions or actions) and the states are meta-data when the first technique is used. However they are data when the second technique is used.

Technique 1

The decision table can be seen as a set of ordered n-tuples $(ct_1, \dots, ct_{cnum}, av_1, \dots, av_{anum})$, with $ct_i \in CT_i$ and $av_j \in AV_j$, that can be represented as a relational table. This relational table, as representation of a decision table, has the following characteristics:

- each row represents a column of the decision table;
- the rows do not have any particular order (but some orderings are more useful);
- all rows are distinct (exclusivity);
- the order of the columns (conditions and actions) is not important to the description of the problem at the logical level (unless a certain order has to be respected at execution time because of side effects, for instance an ordering of the actions);
- the meaning of each column is defined through a named domain as heading (condition or action subject);
- on each row position of the table, an attribute value (a condition state or an action value, possibly "null") is found, and not a set of values.

It is clear that such relational table is identical to the transposed expanded decision table, so that the rows correspond with the columns of the decision table and vice versa. Since every condition combination occurs precisely once in the relation, the condition attribute values uniquely identify the n-tuples in the relation (candidate key). It is, indeed, the intention of the decision table to indicate which actions should be executed for a given combination of conditions. So the set of condition attributes is defined as primary key. The action attributes can then be indicated as the non-key attributes. Notice that it is sufficient that the primary key consists of the condition attributes, while this was not the case when storing general production rules.

By analogy with dependencies in relational tables, the relationship between conditions and actions (or possibly the interrelationship between conditions or actions) in decision tables can be expressed as a cause-effect relation. Such logical if...then...-relation corresponds with the "implicational statement" in propositional logic, which is equivalent to the "dependency statement" for functional dependencies. The decision table, being a set of implications, can therefore be described in terms of functional dependencies. In the first instance, especially the dependency between conditions and actions is important, so that functional dependency can be defined as:

Definition 1 Given a decision table DT with conditions C_i ($i=1..cnum$) and actions A_j ($j=1..anum$) and X, Y subsets of resp. the condition set and action set of DT: Y is *functional dependent* on X if with every combination of X-values in DT corresponds one and only one configuration of Y-values.

Since every combination of condition states occurs at most once, each action is functional dependent on the complete condition set (primary key). For an elaborated treatment of this topic see [20]. The formal correspondence between the decision table and the relational table is given in table 5.

Decision table	Relational table
condition (row)	key attribute (column)
condition states	key domain
action	non-key attribute
action value	non-key domain
stub	heading
number of rows	degree
entry	attribute value
column	n-tuple
number of columns	cardinality

Table 5. Terminology of the decision table and the relational table

In [12], [15] the technique 1 to store rules (cf. supra) was used to store a decision table. How the decision tables of figure 1 are stored, using this technique, is shown in tables 6-8.

Credit_Limit_?	»CustomerGood	Stock_Sufficient_?	»ExecuteOrder	Refuse_Order	Put_On_Waiting_List
ok	y	y	x	-	-
ok	y	n	-	-	x
ok	n	y	x	-	-
ok	n	n	-	-	x
not ok	y	y	x	-	-
not ok	y	n	-	-	x
not ok	n	y	-	x	-
not ok	n	n	-	x	-

Table 6. Table ORDER

Age_of_Account	Turnover_?	CustomerGood:=N	CustomerGood:=Y
<1	<50	x	-
<1	50-100	-	x
<1	>100	-	x
>=1	<50	x	-
>=1	50-100	x	-
>=1	>100	-	x

Table 7. Table CUSTOMERGOOD

Quantity_Ordered_?	Travel_Distance_?	0%	2%	5%	10%	Railway	Road	Bill_A	Bill_B
<10	<50	x	-	-	-	-	x	x	-
<10	50-100	x	-	-	-	-	x	x	-
<10	>100	x	-	-	-	-	x	x	-
10-15	<50	-	-	-	x	-	x	x	-
10-15	50-100	-	-	x	-	-	x	x	-
10-15	>100	-	x	-	-	-	x	x	-
>15	<50	-	-	-	x	x	-	-	x
>15	50-100	-	-	-	x	x	-	-	x
>15	>100	-	-	-	x	x	-	-	x

Table 8. Table EXECUTEORDER

Technique 2

The relations needed to store the system of decision tables, applying technique 2, cannot be readily deduced, therefore an EER-model of a system of decision tables was designed. When designing the EER-model for a decision table system, the following assumptions were made:

- A subject is a condition or an action, but not both;
- a subject which occurs with the same name in different decision tables is the same subject;
- the states for a subject are always the same;
- each subject can call one subtable.

Taking into account these assumptions, the following EER-model can be constructed.

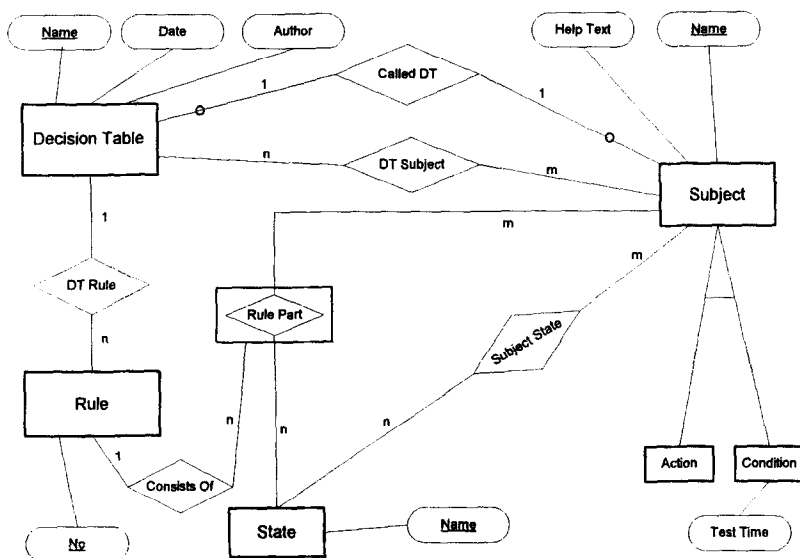


Fig. 2: An EER-model of a system of decision tables

The following relations can be deduced when transforming the conceptual model of figure 2. Attributes which were given a solid underline denote the primary key of the relation. Attributes which were given a dotted underline indicate a foreign key of the relation.

DECISION_TABLE (Name, Date, Author)

In this relational table some characteristics of the decision tables are stored such as creation date, author, etc. .

SUBJECT (SubjectName, Called_DT_Name, Type, HelpText)

In the table SUBJECT all possible subjects, which can occur in a decision table are stored together with their type (condition or action). In this table it is also stored which subtable ('-' if no subtable is called) the subject calls to determine its value. Finally some helptext is stored, which can help the user during the consultation.

ACTION (ActionName)**CONDITION (ConditionName, TestTime)**

These tables results from the specialisation in the EER-model. For the table ACTION no additional attributes were given in this model, so for the moment this table is redundant. But in view of future extensions to the model it might be interesting to have this table already. In the table CONDITION, the attribute TestTime is stored. The test time is the time needed to assign a value to a condition. This is interesting for optimization purposes. E.g. when generating a decision tree from a decision table, a goal might be to minimize the average time needed to answer a question.

DT_SUBJECT (DT_Name, SubjectName)

The table DT_SUBJECT keeps track of which subjects occur in which decision table.

STATE (StateName)

In this table all the different states of the subjects are stored.

SUBJECT_STATE (SubjectName, StateName)

This table specifies which states can be assigned to a particular condition

RULE (RuleNo, DT_Name)**RULEPART (RuleNo, SubjectName, StateName)**

In the table RULE, the number of the rule is stored together with the name of the decision table where the rule refers to. In the table RULEPART, the columns of a decision table are stored using technique 2 (cf. supra).

These tables are filled using INSERT statements. An example (the model will be demonstrated using the tables of figure 1) of such a statement for the table SUBJECT is the following:

INSERT INTO SUBJECT

VALUES ('Credit_Limit_?', -, C, 'Some helptext concerning credit limit');

Since the other INSERT statements are similar they will not be repeated. The result of those INSERT statements is given in the next tables.

Name	Date	Author
Order	10/07/1994	Baldwin
CustomerGood	10/07/1994	Reynolds
Stock_Sufficient_?	10/07/1994	Gaines

Table 9. Table DT

		Type	HelpText
Credit_Limit_?	-	C	Some helptext concerning credit
CustomerGood	CustomerGood	C	
Stock_Sufficient_?	-	C	
ExecuteOrder	ExecuteOrder	A	
Refuse_Order	-	A	
Put_On_Waiting_List	-	A	
Age_Of_Account_?	-	C	
Turnover_?	-	C	
CustomerGood:=N	-	A	
CustomerGood:=Y	-	A	
Quantity_Ordered_?	-	C	
Travel_Distance_?	-	C	
No_Discount	-	A	
Discount_2%	-	A	
Discount_5%	-	A	
Discount_10%	-	A	
Railway_Transport	-	A	
Road_Transport	-	A	
Bill_Type_A	-	A	
Bill_Type_B	-	A	



Primary key



Foreign key

Table 10. Table SUBJECT

ActionName
ExecuteOrder
Refuse_Order
Put_On_Waiting_List
CustomerGood:=N
CustomerGood:=Y
No_Discount
Discount_2%
Discount_5%
Discount_10%
Railway_Transport
Road_Transport
Bill_Type_A
Bill_Type_B

Table 11. Table ACTION

Condition Name	TestTime
Credit_Limit_?	5
CustomerGood	7
Stock_Sufficient_?	10
Age_Of_Account_?	3
Turnover_?	7
Quantity_Ordered_?	9
Travel_Distance_?	2

Table 12. Table CONDITION

DT_Name	SubjectName
Order	Credit_Limit_?
Order	CustomerGood
Order	Stock_Sufficient_?
Order	ExecuteOrder
Order	Refuse_Order
Order	Put_On_Waiting_List
CustomerGood	Age_Of_Account_?
CustomerGood	Turnover_?
CustomerGood	CustomerGood:=N
CustomerGood	CustomerGood:=Y
ExecuteOrder	Quantity_Ordered_?
ExecuteOrder	Travel_Distance_?
ExecuteOrder	No_Discount
ExecuteOrder	Discount_2%
ExecuteOrder	Discount_5%
ExecuteOrder	Discount_10%
ExecuteOrder	Railway_Transport
ExecuteOrder	Road_Transport
ExecuteOrder	Bill_Type_A
ExecuteOrder	Bill_Type_B

Table 13. Table DT_Subject

StateName
Ok
Not_Ok
Y
N
A<1_Year
A>=1_Year
T<50
50<=T<100
T>100
Q<10
10<=Q<15
Q>=15
D<50
50<=D<100
D>=100
x
-

Table 14. Table STATE

SubjectName	StateName
Credit_Limit_?	Ok
Credit_Limit_?	Not_Ok
Credit_Limit_?	-
CustomerGood	Y
CustomerGood	N
CustomerGood	-
Stock_Sufficient_?	Y
Stock_Sufficient_?	N
Stock_Sufficient_?	-
ExecuteOrder	x
ExecuteOrder	-
Refuse_Order	x
Refuse_Order	-
Put_On_Waiting_List	x
Put_On_Waiting_List	-
Age_Of_Account_?	A<1_Year
Age_Of_Account_?	A>=1_Year
Age_Of_Account_?	-
Turnover_?	T<50
Turnover_?	50<=T<100
Turnover_?	T>100
Turnover_?	-
CustomerGood:=N	x
CustomerGood:=N	-
CustomerGood:=Y	x
CustomerGood:=N	-
Quantity_Ordered_?	Q<10
Quantity_Ordered_?	10<=Q<15
Quantity_Ordered_?	Q>=15
Quantity_Ordered_?	-
Travel_Distance_?	D<50
Travel_Distance_?	50<=D<100
Travel_Distance_?	D>=100
Travel_Distance_?	-
No_Discount	x
No_Discount	-
Discount_2%	x
Discount_2%	-
Discount_5%	x
Discount_5%	-
Discount_10%	x
Discount_10%	-
Railway_Transport	x
Railway_Transport	-
Road_Transport	x
Road_Transport	-
Bill_Type_A	x
Bill_Type_A	-
Bill_Type_B	x
Bill_Type_B	-

Table 15. Table SUBJECT_STATE

RuleNo	
1	Order
2	Order
3	Order
4	Order
5	Order

Table 16. Table RULE

RuleNo	StateName	SubjectName
1	Credit_Limit_?	Ok
1	Stock_Sufficient_?	Y
1	ExecuteOrder	x
2	Credit_Limit_?	Ok
2	Stock_Sufficient_?	N
2	Put_On_Waiting_List	x
3	Credit_Limit_?	Not_Ok
3	CustomerGood	Y
3	Stock_Sufficient_?	Y
3	ExecuteOrder	x
4	Credit_Limit_?	Not_Ok
4	CustomerGood	Y
4	Stock_Sufficient_?	N
4	Put_On_Waiting_List	x
5	Credit_Limit_?	Not_Ok
5	CustomerGood	N
5	Refuse_Order	x

Table 17. Table RULEPART

4. MODELLING DECISION TABLES IN A RELATIONAL DATABASE SYSTEM

In the previous section two techniques were explained to store decision tables in a relational database. Storing decision tables, however, is only one aspect. In this section, the construction, including verification and validation, of a decision table in a relational environment is illustrated based on the correspondence between a decision table and a relational database. The full construction process is formulated through the use of SQL-statements. The decision tables are stored using technique 1 of section 3, because this technique is the most intuitive way of storing a decision table in a relational database system. However, if needed it is also possible to store the decision tables using technique 2.

When the construction process is finished, the decision tables can be consulted using SQL-queries.

4.1. The Construction Process of Decision Tables Using SQL

This construction process proceeds analogously to the "direct method based on simple rules" [22] (cf. supra). This method will be illustrated with the decision table ORDER of figure 1.

4.1.1 Obtain Conditions, Condition States and Actions of the Decision Situation

The knowledge engineer obtains in an interactive way relevant conditions, condition states and actions from the expert. When the list of conditions, condition states and actions is obtained, a table is created for each condition. This table contains one column (the condition name) and as many rows as there are different states. Because of the demands of completeness and exclusivity, the states have to be unique and should not be irrelevant (-). Later on, these tables will give rise to the expanded decision table by executing a join operation.

```

CREATE TABLE C_CREDIT_LIMIT_? ( Credit_Limit_? CHARACTER (6) PRIMARY KEY);
INSERT INTO C_CREDIT_LIMIT_?
  VALUES ('Ok');
INSERT INTO C_CREDIT_LIMIT_?
  VALUES ('Not Ok');
CREATE TABLE C_CUSTOMERGOOD (»CustomerGood CHARACTER PRIMARY KEY);
INSERT INTO C_CUSTOMERGOOD
  VALUES ('Y');
INSERT INTO C_CUSTOMERGOOD
  VALUES ('N');
CREATE TABLE C_STOCK_SUFFICIENT_? ( Stock_Sufficient_? CHARACTER PRIMARY
KEY);
INSERT INTO C_STOCK_SUFFICIENT_?
  VALUES
INSERT INTO C_STOCK_SUFFICIENT_?
  VALUES ('N');

```

Next, the empty, expanded decision table can be created and the condition part can be filled. The condition part consists of the Cartesian product of the various condition states and can thus be formulated as a join of the already constructed condition tables (*C_CREDIT_LIMIT_?*, *C_CUSTOMERGOOD*, and *C_STOCK_SUFFICIENT_?*). According to the definition of the expanded decision table, the condition combinations must be unique and the condition states must not be irrelevant.

```

CREATE TABLE ORDER (
  Credit_Limit_? CHARACTER (6) PRIMARY KEY,
  »CustomerGood CHARACTER PRIMARY KEY,
  Stock_Sufficient_? CHARACTER PRIMARY KEY,
  »ExecuteOrder CHARACTER,
  Refuse_Order CHARACTER,
  Put_On_Waiting_List CHARACTER);

INSERT INTO ORDER (Credit_Limit_?, CustomerGood, Stock_Sufficient_?)
SELECT *
FROM C_CREDIT_LIMIT_?, C_CUSTOMERGOOD, C_STOCK_SUFFICIENT_?

```

The construction process for the tables *CUSTOMERGOOD* and *EXECUTEORDER* is analogous.

4.1.2 Specify the Problem in Terms of Decision Rules

The next step is adding the decision rules (these rules can be obtained from a text, interview with the user, ...). These decision rules correspond with the rule base in knowledge based systems. The decision rules which were obtained for the system of decision tables of figure 1 are:

Decision table ORDER:

if Stock Sufficient ? = y \wedge (Credit limit ? = Ok \vee \gg CustomerGood = y) \rightarrow \gg ExecuteOrder = 'x'
 if Credit Limit ? = Not Ok \wedge \gg CustomerGood = n \rightarrow Refuse order = 'x'
 if Stock Sufficient = n \wedge (Credit Limit ? = Ok \vee \gg CustomerGood = y) \rightarrow Put On Waiting List = 'x'

Decision table CUSTOMERGOOD:

if Age Of Account ? = \geq 1 Year \wedge Turnover ? = 50-100 \vee Turnover ? = $<$ 50 \rightarrow CustomerGood := N = 'x'
 if Age Of Account ? = $<$ 1 Year \wedge Turnover ? = 50-100 \vee Turnover ? = $>$ 100 \rightarrow CustomerGood := Y = 'x'

Decision table EXECUTEORDER:

if Quantity Ordered ? = Q $<$ 10 \rightarrow No Discount = 'x'
 if Quantity Ordered ? = 10 \leq Q $<$ 15 \wedge Travel Distance ? = D \geq 100 \rightarrow Discount 2% = 'x'
 if Quantity Ordered ? = 10 \leq Q $<$ 15 \wedge Travel Distance ? = 50 \leq D $<$ 100 \rightarrow Discount 5% = 'x'
 if Quantity Ordered ? = 10 \leq Q $<$ 15 \wedge Travel Distance ? = D $<$ 50 \vee Quantity Ordered ? = Q \geq 15 \rightarrow Discount 5% = 'x'
 if Quantity Ordered ? = Q \geq 15 \rightarrow Railway Transport = 'x' \wedge Bill Type B = 'x'
 if Quantity Ordered ? = Q $<$ 10 \vee Quantity Ordered ? = 10 \leq Q $<$ 15 \rightarrow Road Transport = 'x' \wedge Bill Type A = 'x'

4.1.3 Fill the Decision Table Based on the Decision Rules

The implementation of those decision rules can be realized by a number of update statements, with in the where clause the logical expression selecting the condition combinations for which the action(s) must be executed.

UPDATE ORDER

SET \gg ExecuteOrder = 'x'
 WHERE Stock_Sufficient ? = 'Y' AND (Credit_limit ? = 'Ok' OR \gg CustomerGood = 'Y');

UPDATE ORDER

SET Refuse_Order = 'x'
 WHERE Credit_Limit ? = 'Not_Ok' AND \gg CustomerGood = 'N';

UPDATE ORDER

SET Put_On_Waiting_List = 'x'
 WHERE Stock_Sufficient ? = 'N' AND (Credit_Limit ? = 'Ok' OR \gg CustomerGood = 'Y');

Finally, this results in the completed decision table that can be checked and adapted, see table 6.

An alternative technique: fill the decision table using the grid chart

Formulating the decision rules as update statements, has the disadvantage that these rules are not preserved in the decision description, although they constitute an essential part of the decision situation. Therefore it is advised to construct a separate table corresponding with the so called decision grid chart (cf. supra) and to fill up the decision table using this added table. The grid table is constructed in a similar way as the decision table. However, a combination of condition values may occur more than once in the grid chart, thus the primary key of a tuple consists of all the subjects taking part in that tuple. Next, the decision table will be created using the GRID table.

*Adding the decision rules to the grid***INSERT INTO GRID**

VALUES ('OK', -, 'Y', 'x', -, -);

```
INSERT INTO GRID
VALUES ('OK', -, 'N', -, -, 'x');
```

```
INSERT INTO GRID
VALUES ('NotOK', 'Y', Y, 'x', -, -);
```

```
INSERT INTO GRID
VALUES ('NotOK', 'Y', 'N', -, -, 'x');
```

```
INSERT INTO GRID
VALUES ('NotOK', N, '-', -, 'x', -);
```

Filling the decision table from the grid chart

```
UPDATE ORDER T
SET »ExecuteOrder = 'x'
WHERE EXISTS
  (SELECT *
   FROM GRID
   WHERE (T.Credit_Limit_? = Credit_Limit_? OR Credit_Limit_? = '-')
        AND (T.»CustomerGood = »CustomerGood OR CustomerGood = '-')
        AND (T.Stock_Sufficient_? = Stock_Sufficient_? OR Stock_Sufficient_? = '-')
        AND (»ExecuteOrder = 'x'));
```

The UPDATE-statement is analogous for the other actions. The construction of the tables CUSTOMERGOOD and EXECUTEORDER is analogous.

These update statements are independent of the decision logic and remain unchanged when the decision logic is changed. Whenever the grid is updated, they have to be triggered.

In this simple case it is not necessary to reduce these rules. However in more complicated situations it might be useful to minimize the decision grid chart [9]. Minimizing the decision grid chart minimizes the number of columns in the grid chart. As a consequence it will improve efficiency.

4.1.4. Check for Completeness, Consistency, and Correctness

In a vast majority of cases, the decision table technique is able to provide for extensive validation and verification assistance. Most of the common verification and validation problems in rule based systems [10] can be solved using decision tables as will be shown next.

1. Redundant rules

Redundant Rules may be decomposed into four subcategories: identity, subsumption, unfireability and reducibility.

Identical rules: rules with the same premises and equal conclusions. Because in the decision table every possible case is included in only one column (exclusivity), identical rules will not occur. When the decision tables are stored in a relational database, this is enforced by including all the condition subjects in the primary key.

Subsumption: rules with the same conclusions but with one of them containing additional premises (and therefore being less general). Subsumption will not occur in the decision table, because columns do not overlap.

Unfireable rules: these are rules that will never be executed. E.g. if $a = y \wedge a = n \rightarrow b$. This type of error is not possible in a decision table since the condition a can take only one value at a time.

Reducible rules: complementary rules with equal conclusions, which can be combined. In a compressed decision table, two or more complementary rules with equal action configurations are combined, leading to irrelevant or partly irrelevant conditions. The number of distinct columns is thereby minimized.

2. Inconsistency

Inconsistency can occur in two ways: ambiguous rules and conflict rules.

Ambiguous rules: rules with the same premises and different, not contradictory conclusions. Because in the decision table every possible case is included in only one column (exclusivity), ambiguous rules will not occur.

Conflict rules: Rules with the same premises (or containing overlapping combinations), but leading to contradictory conclusions. If those different conclusions are different states of the same action this will be called contradiction. In a decision table all columns are non-overlapping and each column refers to exactly one configuration of conclusions, thus conflict is easily detected.

Of course it is possible that during the building process the designer wants to override previous configurations by adding a decision rule to the decision grid chart. But each time a decision rule is added, a trigger will be fired to check consistency. This trigger will then warn the developer if a conflict has been recognized.

3. Cyclical rules

Cyclical rules are a set of rules where a conclusion occurs somewhere as one of the premises. In a decision table context, conclusions occurring as premises lead to separate tables. The forward character of the decision table structure eliminates the problem of cyclical references.

4. Completeness of knowledge

Two types of incompleteness of knowledge are examined: unused attribute values or combinations and unreachable conclusions.

Unused attribute values or combinations

When possible attribute values (or combinations) never occur as premises, a number of rules is missing. The nature of the decision table easily allows to check for completeness: the number of simple columns should equal the product of the number of states for every condition. The join instruction guarantees to generate all possible combinations of condition states. This guaranty of completeness of condition combinations is one of the main advantages of decision tables.

Unreachable conclusions

This are conclusions which are never deduced and cannot be asked. If the result of the following query is empty, »Execute Order is an unreachable conclusion.

```
SELECT *
FROM ORDER
WHERE »ExecuteOrder = 'x' ;
```

Empty columns

To check if at least one action corresponds with each combination of condition values, the following query can be executed. If at least one row is selected, the table is not complete.

```

SELECT Credit_Limit_?, »CustomerGood, Stock_Sufficient_?
FROM ORDER
WHERE »ExecuteOrder IS NULL AND Refuse_Order IS NULL AND Put_On_Waiting_List IS
NULL ;

```

5. Correctness

After the decision tables have been designed, the knowledge engineer has to check with the expert the correctness of the decision tables. In this process the checks which need to be performed are not syntactic anymore, but semantic, therefore this process cannot be entirely automatized, but some computer support can be given.

E.g. if it is not allowed for the actions Bill Type A and Bill Type B to occur together, then the following query will discover this error:

```

SELECT Quantity_Ordered_?, Travel_Distance_?
FROM EXECUTEORDER
WHERE Bill_Type_A = 'x' AND Bill_Type_B <> '-' OR
      Bill_Type_B = 'x' AND Bill_Type_A <> '-' ;

```

4.1.5. Simplify the Decision Table and Display It

When the decision tables are verified and validated, they can be contracted. This is important because good overview and compactness support the acquisition process. It ameliorates the dialogue with the expert. Now the expert can concentrate on less cases because table contraction provides a more global view when looking at the problem. The possibility that the expert gets lost in details will be considerably reduced.

The relational environment does not provide facilities to support the contraction process. Therefore a tool, Prologa has been developed which handles advanced features of decision table construction. This will be elaborated in the next section.

4.2. Consulting the Decision Tables

Next some queries are proposed. The first query shows an example of the usual consultation of decision tables. Various other queries are also possible, as illustrated in the examples below.

1. Which action need to be taken when the credit limit of customer is OK, the customer is a good customer and the stock is sufficient ?

```

SELECT »ExecuteOrder, Refuse_Order, Put_On_Waiting_List
FROM ORDER
WHERE Credit_Limit_? = 'Ok' AND »CustomerGood = 'Y' AND Stock_Sufficient_? = 'Y';

```

2. What are the possible results if it is only known that their is not enough stock ?

```

SELECT »ExecuteOrder, Refuse_Order, Put_On_Waiting_List
FROM ORDER
WHERE Stock_Sufficient_? = 'N';

```

3. In which cases does a customer get a discount of 10% ?

```
SELECT O.Credit_Limit_?, C.Age_of_Account, C.Turnover, O.Stock_Sufficient_?,
       E.Quantity_Ordered_?, E.Travel_Distance_?
FROM   Order O, CustomerGood C, ExecuteOrder E
WHERE  O.ExecuteOrder = 'x' AND ((O.»CustomerGood = 'Y' AND C.CustomerGood:=Y = 'x') OR
                                  (O.»CustomerGood = 'N' AND C.CustomerGood:=N = 'x')) AND E.10% = 'x';
```

4. When will an order be put on a waiting list ?

```
SELECT O.Credit_Limit_?, O.Stock_Sufficient_?, C.Age_of_Account, C.Turnover
FROM   Order O, CustomerGood C
WHERE  O.Put_On_Waiting_List = 'x' AND
       ((O.»CustomerGood = 'Y' AND C.CustomerGood:=Y = 'x') OR
        (O.»CustomerGood = 'N' AND C.CustomerGood:=N = 'x'));
```

Although a decision table can be constructed in a relational environment, some advanced features of decision table construction are not possible, e.g. decision table contraction. In addition, the query facilities offered by SQL are rather basic with regard to explanations, user interface,

Those advanced features together with a flexible consultation manager are offered by Prologa (PROcedural LOGic Analyzer), an interactive rule-based design tool for decision table construction and manipulation.

5. INTEGRATING THE RELATIONAL FEATURES IN A DECISION TABLE SYSTEM

5.1. Evaluation of the Construction Process with the Relational Approach

Besides its utility in the construction process of decision tables, the relational approach provides report generation, recovery and basic facilities for consulting the information contained in the table. However, the relational environment by itself does not offer enough facilities to support the construction and consultation process in a flexible and effective way. The following facilities for instance are not or hardly available:

- use of a powerful specification language;
- contraction of the decision table;
- automatic reconstruction after modifications;
- optimal conversion to computer programs;
- recursive queries;
- decomposition of a large decision table in a hierarchy of smaller decision tables;
- decision making using what-if analysis.

Therefore we recommend to provide these facilities in a decision table workbench, while the internal storage and manipulation of the data will be performed by the relational database system. This workbench could then generate the SQL-statements to store the tables and to perform extended verification and validation.

5.2. Adding Prologa Features to the Proposed Approach

A major drawback of the use of decision tables (and many other condition oriented representations) is the complexity of the manual building process. A lot of redrawing work results from small changes like adding a condition, a condition state or an action. Some manipulations like the reordering of conditions are quite impossible to perform manually. To this end, **PROLOGA** (PROcedural LOGic Analyzer) has been developed, an interactive rule-based design tool for computer-supported construction and manipulation of decision tables ([15], [16]). Prologa has e.g. been used to develop **HANDIPAK** [17], a fully operational knowledge based system with regard to financial benefits for the disabled in Belgium. Next we will explain some features of Prologa.

- A powerful specification language allows the designer to formulate the decision specification in a straightforward way (with provisions for expressing general rules, exceptions, preliminary results, restrictive causes and consequences). Some example skeletons of decision rules:

Actions [generally] if condition combinations

Not action definitely if condition combinations

Action only possible if condition combinations

Action definitely if and only if condition combination

The modelling process can be simplified considerably by the use of interactive possibilities such as automatic checking for consistency, correctness and completeness or recommendations for a specific construction method.

- When the table is constructed and verified, it can be contracted in Prologa. Although it is possible to store a simple form of the contracted table in a relational database, the full advantages of decision table contraction are not available. Because, only if all states of a condition lead to the same action configuration, contraction is possible. If groups of states with the same action configuration occur, the states may be simply joined by the OR-connector in Prologa. Of course this type of contraction cannot be stored in a relational table unless a Non First Normal Form model is allowed. Preceding the contraction of a decision table, it is possible to perform row order optimization. This determines the condition order which results in the minimum number of columns. It is not the purpose of this paper to explain in detail how the contracted decision table can be obtained. For more information about these issues, see [8], [15].

- The system can be used for optimization purposes, such as optimal contraction, layout, decomposition into subtables or conversion into efficient program code.

- An important issue is how a large decision table can be split up in a hierarchy of smaller ones. This process is called factoring. Given the striking similarity between decision tables and relational databases the normalisation rules for database design provide an excellent guide-line for the factoring of decision tables. For an elaborated treatment of this topic see [20].

- Although SQL offers some basic consulting facilities, real decision making is hardly possible. Therefore the hierarchy of decision tables is translated into a question and answer interface. The user however, need not be aware of the existence of the decision tables or any relations between them. A full consultation environment is built together with the decision table application [19]. This includes the possibility to get a list of all variables with their values, to obtain a chronological survey of the questions asked, with the answer reached by the consultation environment and finally have the possibility to change one or more answers and restart the reasoning process.

Although extending the basic approach to additional PROLOGA features solves most of the problems mentioned in section 5.1., some problems remain and are targets for future research e.g. guaranteeing semantic exhaustivity.

6. CONCLUSION

In this paper it was shown how decision tables can be stored into a relational database system. First, two techniques to store production rules were presented. Then it was shown how these techniques could be used to store a system of decision tables. Next, it was explained how the relational approach could be used to construct, to verify and to validate decision tables. Finally, the Prologa system was introduced as a tool to support the construction of decision tables in a flexible and efficient way while the internal storage and manipulation of the data can be performed by the relational database system.

Acknowledgements — The authors wish to thank J. Wijsen for his valuable comments and suggestions. We also wish to acknowledge the anonymous referees for their many insightful comments. This research was conducted in the context of LIRIS (Leuven Institute for Research on Information Systems).

REFERENCES

- [1] A. Codasyl. A modern appraisal of decision tables. *Report of the Decision Table Task Group*, ACM, New York, pp. 230-232 (1982).
- [2] E. Codd. A relational model of data for large shared data banks. *Comm. of the ACM*, 13(6):377-387 (1970).
- [3] R. Colomb and C. Chung. Very fast decision table execution of propositional expert systems. *InProc. AAAI90*, pp. 671-676 (1990).
- [4] B. Cragun and H. Steudel. A decision-table based processor for checking completeness and consistency in rule-based expert systems. *Int. Journal of Man-Machine Studies* 5, 633-648 (1987).
- [5] J. Grant and J. Minker. The impact of logic programming on databases. *Comm. of the ACM*, 35(3):66-81 (1992).
- [6] T. Hoppe and P. Meseguer. VVT terminology: A proposal. *IEEE Expert*, 3:48-55 (1993).
- [7] H. Li. To use RDBMS as a building tool of data-knowledge base systems. *InProc. Avignon 86*, volume 2, pp. 1143-1163 (1986).
- [8] R. Maes. An algorithmic approach to the conversion of decision grid charts into compressed decision tables. *Comm. of the ACM*, 23(5):286-293 (1980).
- [9] R. Maes. On minimizing decision grid charts. *Angewandte Informatik*, 9:451-456 (1982).
- [10] T. Nguyen, W. Perkins, T. Laffey, D. Pecora. Knowledge Base Verification. *AI Magazine*, 8(2):69-75 (1987).
- [11] O. K. Ngwenyama and N. Bryson. A formal method for analyzing and integrating the rule-sets of multiple experts. *Information Systems*, 17(1):1-16 (1992).
- [12] A. Salah. *An integration of decision tables and a relational database system into a Prolog environment*, Birmingham, Doctoral dissertation (1986).
- [13] L. Santos-Gomez and M. Darnell. Empirical evaluation of decision tables for constructing and comprehending expert system rules. *Knowledge Acquisition*, 4:427-444 (1992).
- [14] G. Van Emde Boas and P. Van Emde Boas. Storing and evaluating Horn-clause rules in a relational database. *IBM J. Res. Develop*, 30(1):80-91 (1986).
- [15] J. Vanthienen. *Automatiseringsaspecten van de specificatie, constructie en manipulatie van beslissingstabellen*, K.U.Leuven Dept. Applied Econ. Doctoral Dissertation (1986).
- [16] J. Vanthienen. Knowledge acquisition and validation using a decision table engineering workbench. *InThe World Congress on Expert Systems*, Pergamon Press, Orlando, pp. 1861-1868 (1991).
- [17] J. Vanthienen and F. Robben. Developing legal knowledge based systems using decision tables. *In Proc. of The Fourth Int. Conf. on A.I. and Law*, Amsterdam, pp. 282-291 (1993).
- [18] J. Vanthienen and E. Dries. Illustration of a decision table tool for specifying and implementing knowledge based systems. *In Proc. of The Fifth Int. Conf. on Tools with Artificial Intelligence (TAI)*, Boston (Mass.), pp. 98-205 (1993).
- [19] J. Vanthienen and G. Wets. Building intelligent systems for management applications using decision tables. *InProc. of The Fifth Annual Conference on Intelligent Systems in Accounting, Finance and Management*, Stanford (1993).
- [20] J. Vanthienen and M. Snoeck. Knowledge factoring using normalisation theory. *International Symposium on the Management of Industrial and Corporate Knowledge (ISMICK'93)*, October 27-28, Compiègne, pp. 97-106 (1993).
- [21] J. Vanthienen and E. Dries. Decision Tables: Refining the Concept and a Proposed Standard, to appear in: *Comm. of the ACM*.
- [22] M. Verhelst. *De Praktijk van Beslissingstabellen*, Kluwer, Deventer/Antwerpen (1980).