An Extended Relational Model for Managing Uncertain Information

Joseph Nassif Said

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

April 15, 1994

ISBN 0-315-97657-8

ABSTRACT

An Extended Relational Model
for
Managing Uncertain Information

Joseph Nassif Said

Representing uncertain information in database systems could have two approaches: a quantitative approach and a qualitative approach. In the quantitative approach a factor is associated with the information stored in the database representing its "degree of validity" (certainty, reliability, confidence, etc.). In the qualitative approach we track the sources of information by associating source vectors to the information stored into the database. In our new framework, we will consider the qualitative approach to manage uncertain information and we will extend the concept of *The Information Source Tracking Method* introduced by Sadri. To achieve this goal, we seek a correct semantics for the uncertain database model and correct methods to manipulate the uncertain information to provide the users with answers to queries given to such systems as well as the contributing sources and their roles with respect to each answer.

The Information Source Tracking Method (IST) allows every information source (observer) to contribute to the whole tuple in the extended relations. The goal in our new approach is to allow the information sources to contribute to any attribute value of any tuple or to the whole tuple in the extended relations.

Acknowledgement

This thesis would have been impossible without the encouragement and support of My two supervisors Dr. F. Sadri and Dr. V. S. Alagar. I would like to thank them for their helpful discussion and guidance during this work.

I wish to acknowledge the support and help of the SOFTEKS Reasearch Group since the implementation of this thesis was carried out in the SOFTEKS Research Laboratory at Concordia University. Moreover, I would like to point out that the theoretical model in this thesis is an out growth of a term paper presented to Dr. V. S. Lakshmanan in connection with the course COMP 659 - Introduction to Knowledge Base Systems (Fall 1993).

I'd like to thank my parents, my father Nassif, my mother Salma, and my beautiful sister Sandra for always believing in me and encouraging me to achieve high goals. A very special thanks to my brother Fady for many great discussions and for providing me a place in his great heart and in his great soul and mind.

Dedication

This thesis is dedicated

to

My Father

Nassif

My Mother

Salma

My Brother

Fady

and

My Sister

Sandra

Contents

List of Figures

Chapter 1

Introduction

Description of the problem, its significance,
the novelty of the approach, and thesis organization.

1.1 The Problem

Information in a relational database system could come from different sources, where every source has a certain reliability associated with it. In this context, the IST model proposed by Sadri [18, 19, 20, 21] assumes that the information is confirmed by information sources (observers). The reliability of the information provided by these sources is based on the reliability of the contributing sources of information. Each tuple in an extended relation is associated with an information source vector(s) indicating the contributing sources to every tuple. Answers to queries identify the sources contributing to the answer as well as their nature of contribution. Reliability calculation algorithms [18, 19] could be used to calculate the reliability of the answers presented to the query.

Recently, there has been a great deal of interest in implementing the IST method in several strategic areas of importance such as defense and medicine. In such systems, observers might not contribute to a whole tuple stored in an extended relation; however, an observer might contribute to some attribute values that constitute part of that tuple. In this context (IST approach), the observer will contribute to the whole tuple provided to the uncertain database instead of contributing to the individual attribute values that constitute that tuple. For this reason, we are in need for a new model that gives the observers the flexibility to contribute to any attribute value of any tuple in the extended relations.

An extended relational model is proposed in which each attribute value of each tuple in an extended relation is associated with an information source vector showing the sources (observers) that contribute to that attribute. Moreover, each tuple in an extended relation is associated with an information source vector showing the contributing sources and indicating the condition under which each tuple exists in an extended relation. We will discuss how the relational algebra operations can be extended and implemented using information source vectors to trace the information source(s) that corresponds to each attribute/tuple in the answer to a query provided to the extended relational model. This helps us to identify the contributing information source(s) to each attribute/tuple in the answer to a query. The reliability of answers to queries could be calculated based on the reliability of the information sources that contributed to the information stored in the extended relational database model. The reliability calculation algorithms were introduced by Sadri [18, 19] and were proven to be correct. In our work, we will not discuss how to give reliabilities to the contributing information sources; however, we assume that information sources have predefined reliabilities so that we can calculate the reliability of answers to user's queries.

The rest of this thesis is organized as follows. Chapter 2 introduces a survey of similar works in the same area and highlights two active research areas: *Qualitative* and *Quantitative*. Chapter 3 introduces our new formal model and defines the extended relational algebra operations Selection, Projection, Union, Cartesian Product, Join, Intersection, and Set Difference. Chapter 4 presents the semantics of our model and proves that the extended relational algebra operations are "precise" (correct). Chapter 5 introduces and proves the reliability calculation algorithms that calculate the reliability of answers to users queries. Chapter 6 discusses implementation issues of our framework and presents a prototype that implements the new model. Chapter 7 presents the conclusion and our future research directions.

Chapter 2

A Survey of Similar Works and Their Relationship to This Work

Review of some other approaches for uncertainty such as "Quantitative Approach" (van Emden), "Annotated Logic Programs", (Subrahmanian), "Quantitative Logic Programs" (Kifer and Li), "The Management of Probabilistic Data" , (D. Barbara, H. Garcia-Molina, and D. Porter), "Information Source Tracking (IST)" (Sadri), and "IST-Based Deductive Database" (Lakshmanan and Sadri) and a comparison study to Our model.

2.1 Quantitative Approach (van Emden)

van Emden [26] developed a framework for quantitative deduction based on positive *Horn clauses.* In this context, instead of associating the usual truth values $\{0,1\}$ with a rule, an entire continuum of "uncertainties" will be used, where each clause receives a numerical *attenuation* factor from the interval $[0,1]$. In other words. in the new extended model, each element in $[0,1]$ is viewed as a degree of belief, a certainty value, or a confidence factor. Moreover, Herbrand interpretations that are subsets of the Herbrand base are generalized to fuzzy subsets. On the semantic level, van Emden proved that the result of the fixed point method in this quantitative approach is an extension of that in the conventional model.

2.1.1 Syntax of the Language

In the quantitative approach, a rule is of the form:

$$A \overset{f}{\leftarrow} B_1, \ldots, B_n$$

where $A, B_1, \ldots, B_n$ are first order logical formulas, and $n \geq 0$.

The attenuation factor f of a rule is associated with the implication itself rather than with the components $A, B_1, \ldots, B_n$. The purpose behind attaching a factor f to each implication " $\leftarrow$ " indicates the existence of uncertainty in the propagation of the truth values from the conditions, described as the rule body, $B_1, \ldots, B_n$, to the conclusion A presented in the head of the rule. Such a rule is called *Quantitative rule.*

In the new framework, the factor f associated with each quantitative rule in a given program (viewed as a set of annotated rules) is equal to a real value in $(0, 1]$. As in the conventional case, an attenuation factor $f = 0$ is associated with each rule not present in the program. In [26], the propagation of truth values from the body to the head of each rule is defined as follows. The truth value assigned to the head A of a quantitative rule with factor f is $f \times m$, where m is the minimum of the truth values of the atoms in the rule body. In the case where the rule body is empty, m is defined to be 1.

Programs written in van Emden's language are called *rule sets.* A rule set R is a finite set of quantitative rules. As in the conventional case, a Herbrand interpretation I for a rule R is a fuzzy subset[1] of the Herbrand base B_R of R, i.e., $I : B_R \rightarrow [0, 1]$. In other words, if A is a ground atom that appears in the head of a quantitative rule, then The interpretation of A denoted as $I(A)$ is the value of the membership function I at the argument A. The value $I(A)$ can be regarded as the *minimum* certainty with which A, being the head of the quantitative rule, is known to be true.

[1]A fuzzy subset S of a set U is a set where each element $X \in U$ is associated with a value $\mu_S(x)$, where $\mu_S(x)$ indicates the degree or the level of the membership of x in S. The function $\mu_S : U \rightarrow M$ is called a *membership function*, where M is a totally ordered set [9, 28]. In van Emden's language M is extended to be $M = [0, 1]$. Note that when $M = \{0, 1\}$, the "fuzzy subset" becomes as an "ordinary subset".

As a result of extending the truth values from {0,1} to [0,1] the notion of "implications" and "interpretations" should be generalized in the new model.

Definition 1 Let R be a rule, and let I be an interpretation.

1. R is true in I if and only if every rule in R is true in I.
2. A rule is true in I if and only if every one of its ground instances is true in I.
3. A ground instance $A \xleftarrow{f} B_1, \ldots, B_n$ of a rule is true in I if and only if $I(A) \geq f \times m$, where m is equal to $min\{I(B_1), \ldots, I(B_n)\}$. If the rule body is empty, then define $m = 1$.

Note that the above definition reduces to the qualitative case if for all rules in R we have $f = 1$, and $I(A) \in \{0,1\}, \forall A \in B_R$.

2.1.2 Semantics

A *Herbrand model* of a rule R is a Herbrand interpretation I in which R is true. For all rule sets R, $\forall A \in B_R$, and $\forall f \in (0,1]$ we have that $R \models \{A \xleftarrow{f}\}$ if and only if A is true in every Herbrand model of R.

In the qualitative case, the classical result that the intersection of all Herbrand models is also a model, is extended by Emden. In [26], this definition is extended as follows. If M_R is a set of all Herbrand models of a rule set R and S is a family of Herbrand interpretations for R, then $\cap S$ is an interpretation for R such that $\forall A \in B_R$, it assigns the value $inf\{S'(A) \mid S' \in S\}$, where inf is the *greatest lower bound*[2]. It was shown in [26] that the value assigned to each element $A \in B_R$ in the least model $\cap M_R$ is $sup\{f \mid R \models \{A \xleftarrow{f}\}\}$, where sup is *the least upper bound.*

In the interval [0,1], from which an attenuation factor is used, an ordering " $\leq$ ", defined on real numbers, could be defined. As a direct consequence of this, it is obvious that a partial order could be defined among the interpretations as follows. If H_R is a set of all Herbrand interpretations for a rule set R and I_1 and I_2 are elements in H_R. Then $I_1 \subseteq I_2$ if and only if $(\forall A \in B_R), I_1(A) \leq I_2(A)$.

[2]This definition is adopted from a rule for "intersection" for fuzzy sets [28].

Associated with each program R is an operator $T_R : H_R \rightarrow H_R$, such that $T_R(I)(A)$ is defined to be:

$sup\{f \times min\{I(B_1), \ldots, I(B_n)\} \mid A \overset{f}{\leftarrow} B_1, \ldots, B_n$ is a ground instance of a rule in $R\}$.

Note that $min\emptyset = 1$. In [26], it was proven that T_R is *monotonic* and *continuous*. Moreover, in [26] when no functions symbols are not allowed, it was shown that for any finite set of rules, and for all $A \in B_R$,

$$\cap M_R(A) = T_R{}^n(\emptyset)(A)$$

for some integer number n, where M_R is the set of all Herbrand models of a program R, and $\emptyset : B_R \rightarrow \{0\}$. This shows that the truth value of an atom A in the least model of a program R can be obtained in a finite number of iterations of the T_R operator.

2.1.3 Soundness

As for the soundness results of rule sets van Emden [26] pointed out that the proof should be viewed as a two-person game. In this approach, the value of the game turns out to be the truth value of the atomic formulas to be proved, evaluated in the minimal fixpoint of the rule set. Moreover, The analog of the PROLOG interpreter for quantitative deduction becomes a search of the game tree using the alpha-beta heuristic known in game theory. In other words, for every rule set R with a finite and/or tree $T(R, A)$ and every $A \in B_R$, the value of the root in the tree is not greater than $M_R(A)$, where $M_R(A)$ is the value of A in the least model of R.

Formally, for every rule set R with a finite and/or tree and for every $A \in B_R$, the value of the root in the and/or tree $T(R, A)$ is at least $M_R(A)$. This theorem indicates that the proof procedure works when the following conditions are true:

- the and/or tree associated with R is finite

- the query G is ground, i.e., $G \in B_R$.

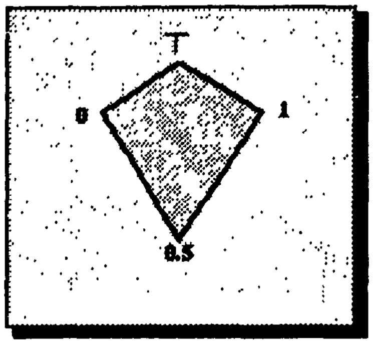


Figure 2.1: Ordering of truth values in $\mathcal{T}$.

In van Emden's work we realize a clear connection to fuzzy logic, resulting in what he called as the "fuzzy" logic "programming". van Emden pointed out that when the existing framework is generalized, in the sense that a weight is assigned for each atom in a rule body, the corresponding results hold true.

2.2 Annotated Logic Programs

The notion of annotated programs was introduced in [24] to extend van Emden's approach [26] since, according to [24], there are two problems in the quantitative approach introduced by van Emden in [26]. The first problem is that negation in the rule bodies and rule heads is not allowed in the quantitative approach. Subrahmanian [24] proposed what he called the *annotated logic*, where negated atoms are allowed to appear in a rule head as well as in the rule body. The second problem observed by Subrahmanian [24] is the total ordering of truth values $[0, 1]$. As a solution, annotated logic used the truth values illustrated in Figure 2.1. $\mathcal{T} = [0, 1] \cup \{\top\}$ on which the ordering $\ll$ is defined as follows:

- for all $x \in \mathcal{T}$, we have $x \ll \top$ and $0.5 \ll x$;
- for all $x, y \in \mathcal{T}$, if $x, y \in [0, 0.5)$ and $x \leq y$, then $y \ll x$;
- for all $x, y \in \mathcal{T}$, if $x, y \in (0.5, 1]$ and $x \leq y$, then $x \ll y$.

The value 0.5 represents the least information (i.e., unknown, undefined, or under-specified), while $\top$ denotes contradiction or *over-specified.* The values 0 and 1 are false and true, in the conventional sense.

2.2.1 Syntax of the Language

The syntax of the language is quickly reviewed next. $A : \alpha$ is an *annotated literal* (atom), where A is a literal (atom) and $\alpha \in [0,1]$. α is called the annotation of A. The result of applying a substitution θ to an annotated literal $A : \alpha$ is $A\theta$. A quantitative clause (q-clause, for short) is defined as:

$$A_0 : \alpha_0 \leftarrow A_1 : \alpha_1, \ldots, A_n : \alpha_n,$$

where A_0 is an atom, $A_1, \ldots, A_n$ are literals, and $\alpha_0, \ldots, \alpha_n$ are all in $[0,1]$.

A *quantitative logic program* (QLP) is a finite set of q-clauses. We refer to a QLP Q as a program Q.

If Q is a program, and B_Q is its Herbrand base. An interpretation I for Q is defined as a function from B_Q to the truth values in $\mathcal{T}$. In other words, to each ground atom in B_Q, I assigns a truth value in $\mathcal{T}$.

Let us give an example. Consider the QLP Q_1 shown below:

$$p(a) \leftarrow 1$$

$$p(a) \leftarrow 0.0001$$

The truth values assigned to $p(a)$ in the least model of Q_1 is a value from $\mathcal{T}$. Assume that each rule in Q_1 is provided by an observer or an expert. Thus, the first expert says that $p(a)$ is *true*, giving $p(a)$ an annotation of 100%, while the observer assumes that $p(a)$ is almost *false*, giving $p(a)$ an annotation of 0.0001%. In this case, Q_1 contains contradictory information about $p(a)$. To model such contradictory situations, the truth value $\top$ is introduced in $\mathcal{T}$, giving a result, based on the ordering in $\mathcal{T}$, $lub\{1, 0.0001\} = \top$.

A generalization of quantitative rules [26] is viewed as follows. Associated with each rule is a computable function g, called *certainty function*, whose range is in (0,1]. In this case, the quantitative rule

$$A \xleftarrow{f} B_1, \ldots, B_n$$

In van Emden's language [26], where the certainty of B_i is $k_i \in [0,1]$, could be expressed as a generalized annotation rule:

$$A : r \times g(k_1, \ldots, k_n) \leftarrow: B_1 : k_1, \ldots, B_n : k_n.$$

where $g = min$.

2.2.2 Summary

In a q-clause, each component of an implication has a truth value associated with it; however, in [26] van Emden defined the rule sets where an attenuation factor is attached to the " $\leftarrow$ ". To compare the two approaches, note that the truth assignment in the latter is more intuitive, since it is conceivable that it deals with uncertainty rules. However, the former is close to the framework of standard logic, since the meaning of "implication" is unchanged.

In some situations we observe that the choice of $\mathcal{T}$ as the truth values and their ordering structure, as represented in Figure 2.1, is not applicable for some applications. Such a problem implies that the semantics is not capable to capture such situations. For instance, if p is known with a high degree of certainty and at the same time p is known with a low degree of certainty then according to [24], this should result in a contradiction -an unlikely course of action in most problem domains like medicine and military. We will see later in section 2.3 Kifer and Li proposed a different solution based on what they called *support logic* [10].

To illustrate, let us consider the following QLP q_2.

$$Students(12345, Mike) : 1 \leftarrow$$

$$Students(12345, Mike) : 0.49 \leftarrow$$

We assume that the predicate *Students* records a name for each student Id. We record such information in a database where each fact comes from an observer or expert. In this example, an expert states that he is 100% sure that "*Mike*", a student, has an Id number of 12345. However, another expert states that he is 49% sure that "*Mike*" has an Id number of 12345. If we observe this fact clearly, we realize that the QLP program, based on the ordering presented in $\mathcal{T}$, gives in the least model the following answer:

$$Students(12345, Mike) : lub\{1, 0.49\},$$

where $lub\{1, 0.49\} = \top$. The least model of QLP informs us that the fact that "*Mike*" has an Id number 12345 is considered as contradiction, which is counterintuitive. Moreover, we observe that the semantics of the QLP does not take into consideration the possibility that the observer who gave us the information $Students(12345, Mike)$: $1 \leftarrow$ might be 100% knowledgable or reliable, while the observer who gave us the information $Students(12345, Mike) : 0.49 \leftarrow$ might be 0.0001% reliable. In this case, we obtain in the least model the fact $Students(12345, Mike) : \top$. Such situations could arise in medicine, military, and many other applications that collect data obtained from various observers and serves the users by answering their queries based on the semantics of QLP, in which case such answers (as the one presented in the previous example) should be tolerated.

Kifer and Li in [10], extended annotations to allow variables and function symbols in the language. Semantics of programs in the extended annotated programs is carefully studied. As to the certainty function such as $g = min$ in the previous example, they defined several properties that a certainty function should have.

In the next section we will present the framework introduced by Kifer and Li in [10].

2.3 Quantitative Logic Programming

Kifer and Li in [10] presented a formal semantics for rule-based systems with uncertainty called "quantitative logic programming". They provided a rigorous treatment of the issue of evidential independence, and its impact on the semantics. Negation and conflicting evidence based on, the so called, *support logic* is given in the last part of their work.

2.3.1 Syntax of the Language

Literals are of the form $p : \sigma$, where p is a literal in the usual predicate calculus , and σ is the certainty information (*certainty term* for short) about p. The usual first order literals are called the *d−literals*. D-literals are of the form $p(x, y, \ldots)$ or $\neg p(x, y, \ldots)$, where p is a predicate symbol and $x, y, \ldots$ are variables or constants. In such literals, function symbols are not allowed. Ground d-literals will be called *d-facts.*

A finite collection of interpreted *certainty functions* is also assumed. A *k-ary* certainty function maps $[0, 1]^k$ into [0,1]. Certainty terms are built in a usual way using these functions, *certainty variables*, and *certainty constants* which will be taken from the domain of real numbers in the interval [0,1].

Definitely true facts are d-literals annotated with certainty 1. A fact with certainty value 0 has no supporting evidence whatsoever. Certainty values between 0 and 1 mean that there is some inconclusive evidence that the fact, to which the certainty value is associated, is true. Note that this does not imply anything about the falsehood of the fact. Dealing with negative information, and representing falsehood of facts was discussed in support logic by Kifer and Li.

A *Horn rule* is a statement of the form:

$$p : \sigma_0 \longleftarrow q_1 : \sigma_1, \ldots, q_n : \sigma_n,$$

where p, q_1, and q_n are positive literals, and $\sigma_0, \sigma_1, \ldots, \sigma_n$ are certainty terms. Variables that appear in the head of the rule (including certainty variables) also appear in the body of that rule. In their work [10], Kifer and Li, assumed that rules are of the form:

$$p : f(\alpha, \beta, \ldots, \gamma) \leftarrow q_1 : \alpha, q_2 : \beta, \ldots, q_n : \gamma,$$

where $\alpha, \beta, \ldots, \gamma$ are certainty variables or constants and f is a certainty function associated with this rule.

An *uncertain database* is a collection of Horn rules P and a set of facts, D. Keeping with the database tradition P is called the *intentional database* (IDB) and D is called the *extentional database* (EDB).

A certainty function associated with a rule is a measure of strength of the causality link between the rule premises and the consequent. Certainty functions have the following restrictions:

- *Monotonicity*: which means that the higher certainty of premises should yield a higher certainty of the consequent. Formally, $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ if $x_i \leq y_i$ for $i = 1, \ldots, n$.
- *Boundedness:* This states that conclusion of a rule can be only as good as its premises. Formally, $f(x_1, \ldots, x_n) \leq x_i$ for $i = 1, \ldots, n$.
- *Continuity:* f is continues w.r.t. to all of its arguments. As pointed out in [10], this technical assumption is needed in order to ensure the agreement between the model-theoretic and the fixpoint semantics introduced later.

Since an evidence could be supported by different (dependent or independent) sources in an uncertain databases, then strengths of such evidences are combined to determine the strength of the overall support for the same fact. Combination function could be different in different problem domains and for different types of facts. Moreover, the order in which facts are combined is not significant in some problem domains.

With a predicate symbol, p, there is a unique *combination function* used to calculate strength of combined evidences for p. Combination functions accept a single multi-valued[3] argument, which implies that the combined strength of evidences for the fact is independent from the order in which these evidences are obtained. Note that it could be argued that it should not be that way, i.e., evidences obtained prior to some other evidences may increase the significance of the latter evidences [8]. Combination functions should have the following properties:

[3]which could have many occurrences of the same element.

- *Commutativity.*
- *Monotonicity:* to ensure that stronger evidences yield stronger overall support for a fact. Formally, $S \leq S'$ implies $F(S) \leq F(S')$. The order on the multisets is defined in the usual way: $S \leq S'$ if there is a $1-1$ mapping f from S into S' s.t. for every $s \in S$, $s \leq f(s)$.
- *Associativity:* to ensure that there is no need to wait until all evidences for that fact are obtained to calculate the overall strength of the fact. Formally, $F(S \cup S') = F(S \cup F(S'))$. Here $\cup$ is a union of multisets which retains duplicate occurrences of the same element.
- *No Information Rule:* to ensure that a non-evidence cannot change the overall support of a certain fact. Formally, $F(S \cup \{0\}) = F(S)$.
- *Correctness:* This means that the combined support provided by a single evidence is exactly as strong as the evidence itself. Formally, $F(\{\alpha\}) = \alpha$.
- *Continuity:* F is continuous w.r.t. $\geq$ on multisets. Continuity of of combination functions is needed for the same reason as in the case of certainty functions.

2.3.2 Interpretations and Models

Although general interpretations can be used, Kifer and Li considered *Herbrand interpretation.* Given an uncertain database $E = P \cup D$, *the domain* D_E of any Herbrand interpretation is a collection of constants mentioned in E. The *Herbrand base* of E is a collection of all ground facts of the form $p(a_1, \ldots, a_n) : \alpha$, where p is a n-ary predicate symbol in P, a_is all belong to D_E, and $\alpha \in [0,1]$ is a certainty factor of $p(a_1, \ldots, a_n)$. A *Herbrand interpretation* I of E is a subset of the Herbrand base of E. Assume that for every d-fact there is at most one fact $p : \alpha \in I$. If $p : \alpha \notin I$ for all $\alpha > 0$ then we assume that $p : 0 \in I$.

If I is an interpretation and $p(a_1,\ldots,a_k) : \alpha$ is a fact, then $p(a_1,\ldots,a_k) : \alpha$ is *true* under I whenever $p(a_1,\ldots,a_k)$ is known with a higher certainty, β. Formally, $p(a_1,\ldots,a_k) : \alpha$ is *true* under I if there is $p(a_1,\ldots,a_n) : \beta \in I$ such that $\beta > \alpha$.

A ground rule $p(\vec{a}) : f(\alpha_1,\ldots,\alpha_k) \leftarrow q_1(\vec{e_1}) : \alpha_1,\ldots,q_k(\vec{e_k}) : \alpha_k$ is true in I if and only if whenever all the $q_i(\vec{e_i}) : \alpha_i$ are true in I, the head of rule, $p(\vec{a}) : f(\alpha_1,\ldots,\alpha_n)$ is also true. A (nonground) rule is true in I if and only if all its ground instances are true. In this framework, ground rules are used as evidences to the facts in their heads. We say that the above rule *supports* $p(\vec{a}) : f(\alpha_1,\ldots,\alpha_k)$ in I if all its literals (head and body) are true in I. We will also refer to such rules as *evidences.*

A program, P, is true if all its rules are true, *and* in addition, the following *combination requirement* is satisfied:

For every set of independent (explained later) ground instances of the rules of P with the same head d-literal

$p(\vec{a}) : f_1(\alpha_{11},\ldots,\alpha_{k_1 1}) \longleftarrow q_{11}(\vec{e_{11}}) : \alpha_{11},\ldots,q_{k_1 1}(\vec{e_{k_1 1}}) : \alpha_{k_1 1}$

...

$p(\vec{a}) : f_m(\alpha_{1m},\ldots,\alpha_{k_1 m}) \longleftarrow q_{1m}(\vec{e_{1m}}) : \alpha_{1m},\ldots,q_{k_1 m}(\vec{e_{k_1 m}}) : \alpha_{k_1 m}$

such that each individual rule supports its head literal in I, the literal

$$p(\vec{a}) : F_p(\{f_1(\alpha_{11},\ldots,\alpha_{k_1 1}),\ldots,f_m(\alpha_{1m},\ldots,\alpha_{k_1 m})\})$$

should be true in I. This will ensure that independently obtained evidences for the same fact are combined to obtain a possibly stronger evidence.

An unusual consequence of this definition is that although each individual rule might be true, the program might not be true, the program itself might not be satisfied by the interpretation, because some of the facts may not be known with sufficient certainty to satisfy the combination requirement (since $F(S) \geq \alpha$ for any combination function F and for each $\alpha \in S$).

Supporting evidences should be independent so that they could be combined together. Kifer and Li introduced two kinds of evidential independence.

- *Independence-1:* A pair of ground rules $p : \alpha \leftarrow q_1 : \beta_1, \ldots, q_n : \beta_n$ and $p : \gamma \leftarrow r_1 : \delta_1, \ldots, r_m : \delta_m$ provide independent evidences for the fact p if and only if the sets of ground atoms $\{q_1, \ldots, q_n\}$ and $\{r_1, \ldots, r_m\}$ are different.

- *Independence-2:* A pair of ground rules $p : \alpha \leftarrow q_1 : \beta_1, \ldots, q_n : \beta_n$ and $p : \gamma \leftarrow r_1 : \delta_1, \ldots, r_m : \delta_m$ provide independent evidences for p if and only if the sets of $\{q_1, \ldots, q_n\}$ and $\{r_1, \ldots, r_m\}$ are incomparable w.r.t. $\subseteq$.

I is a model of $E = P \cup D$ if and only if P and each fact in D are true in I. We also write $E \models p(a_1, \ldots, a_n) : \alpha$ if $p(a_1, \ldots, a_n) : \alpha$ is true in every model of E. A ***partial order*** on the interpretations is defined as follows: $I \subseteq J$ if and only if for every $p(a_1, \ldots, a_n) : \alpha \in I$ there is $p(b_1, \ldots, b_n) : \beta$ such that $\alpha \leq \beta$. Clearly, $I = J$ if and only if $I \subseteq J$ and $J \subseteq I$.

Intersection of a collection of interpretations, $\{I_k\}_{k \in K}$, is the interpretation:

$$\bigcap_{k \in K} = \{p : \alpha \mid \alpha = \inf_{k \in K}(\alpha_k), \text{ where } p : \alpha_k \in I_k, \text{ for all } k \in K\}.$$

Similarly, the *union* of the above collection of interpretations is defined to be

$$\bigcup_{k \in K} = \{p : \alpha \mid \alpha = \sup_{k \in K}(\alpha_k), \text{ where } p : \alpha_k \in I_k, \text{ for all } k \in K\}.$$

In this context, Kifer and Li, pointed out that it could be shown that the intersection of any number of models of $E = P \cup D$ is also a model. Therefore, E has a least model, which is called the *intended* model of E.

2.3.3 The Fixpoint Semantics

The *immediate consequence operator* T_E of E is another interpretation of E, where I is an interpretation of E, defined as follows. Let $a(\vec{q})$ be an arbitrary d-fact. If $a(\vec{q})$ is a base predicate then $a(\vec{q}) : \alpha \in T_E(I)$ if and only if $a(\vec{q}) : \alpha \in D$. if $a(\vec{q})$ is a derived predicate then and for no certainty $\alpha > 0$, $a(\vec{q}) : \alpha$ is supported by a rule in P, then $a(\vec{q}) : 0$ is in $T_P(I)$. Otherwise, suppose that $a(\vec{q}) : \alpha$ is supported for some $\alpha > 0$. Consider any set of *independent ground* rules of P defining $a(\vec{q})$ whose bodies are true in I: $a(\vec{q}) : \beta_1 \leftarrow body_1, \ldots, a(\vec{q}) : \beta_k \leftarrow body_k$. Then $a(\vec{q}) : \gamma$ is in $T_E(I)$, where

$$\gamma = \max_{\beta}\{F_a(\{\beta_1, \ldots, \beta_k\})\},$$

and *max* ranges over the collection of all multisets of certainty factors $\beta = \{\beta_1, \ldots, \beta_k\}$ produced by all possible independent ground rules defined before.

The definition of the T_E operator is somewhat more involved than that defined by van Emden [26] because with general combination functions it is possible that $F(S) > \max(S)$. In this case, one should be careful not to combine dependent evidences. In contrast, in [23, 26, 24] $F(S) = \max(S)$, in which case evidence independence does not have any significance.

Lemma 1 The immediate consequence operator T_E is monotonic, i.e. $I \subseteq J$ implies $T_E(I) \subseteq T_E(J)$, and continuous, i.e. for any monotonically increasing sequence of interpretations $I_1 \subseteq I_2 \subseteq I_3 \subseteq \ldots \subseteq T_E(\bigcup_j I_j) = \bigcup_j T_E(I_j)$.

Theorem 1 Th · least fix point of T_E is equal to $\bigcup_j T_E{}^i(\emptyset)$ [4] , where $\emptyset$ is the interpretation in which every fact is assigned certainty 0.

Theorem 2 The least fixed point of T_E is the least (w.r.t $\subseteq$) model of E. in other words, the least fixed point of T_E coincides with the intended model of E.

2.3.4 Semantics of Conflicting Evidences and Negation

To allow negative literals to appear in rule premises as well as rule consequences, Kifer and Li in [10] extended the framework introduced in section 2.3.1 . When dealing with incomplete knowledge, the latter is particularly useful, where different evidences may contradict one another. *Support logic* is used to cope with such situations. In support logic, each fact, p, has a measure of *belief*, MB, or certainty, and a measure of *disbelief*, MD, viewed as the measure of the belief in $\neg p$. Thus, each fact, p, is assigned an interval $[low, high]$, meaning that the strength of belief in p is somewhere

[4]$T_E{}^0(I) = I$ and $T_E{}^n(I) = T_E{}^{n-1}(I)(T_E(I))$ for $n > 0$.

between the value *low* and the value *high*. The difference *high* − *low* is the ***knowledge gap*** about p.

In support logic, literals are of the form $p : [\alpha, \beta]$ or $\neg p : [\alpha, \beta]$, where p is a d-literal, and α, β are certainty terms. A negative literal, $\neg p : [\alpha, \beta]$, should be perceived as $(\neg p) : [\alpha, \beta]$ rather than $\neg(p : [\alpha, \beta])$, which will be clear from their semantics. Rules are as in section 2.3.2, except for the following difference:

- Certainty terms are replaced by intervals of certainty terms.
- Negative literals can appear in rule bodies as well as their heads.

Certainty intervals are ordered according to $\leq^p$. Given two certainty intervals $[\alpha, \beta]$ and $[\gamma, \delta]$ the partial ordering on these two intervals is defined as: $[\alpha, \beta] \leq^p [\gamma, \delta]$ if and only if $[\gamma, \delta] \subseteq [\alpha, \beta]$. The intuition behind this definition is that a bigger (w.r.t. $\leq^p$) certainty interval means that stronger positive and negative evidences are available for the associated fact. The domain of all certainty intervals is augmented by adding the element $\top$ that represents all *inconsistent* intervals, i.e., intervals $[\alpha, \beta]$ such that $\alpha > \beta$. Thus, $[\gamma, \delta] \leq^p \top$. Interval $[0, 1]$ is, obviously, the smallest certainty interval. Certainty intervals form a complete lattice: glb (greatest lower bound) of a set of intervals is the smallest (w.r.t. $\subseteq$) interval containing each of the intervals in the set. The least upper bound, lub, of that collection is the largest interval contained in each of the intervals in the set, if it exists; it is $\top$, otherwise. This partial order can be naturally extended to literals so that for the same d-literal p, $p : [\alpha, \beta] \leq^p p : [\gamma, \delta]$ if and only if $[\alpha, \beta] \leq^p [\gamma, \delta]$. Moreover, $p : [\alpha, \beta] <^p p : [\gamma, \delta]$ if $[\alpha, \beta] \leq^p [\gamma, \delta]$, but $[\alpha, \beta] \neq [\gamma, \delta]$.

Given that $E = P \cup D$, the Herbrand base of E is now a collection of all positive ground facts $p(\vec{a}) : [\alpha, \beta]$, where $[\alpha, \beta]$ is a certainty interval (*or*$\top$), p is a predicate symbol from P, and $\vec{a}$ is a vector of values from the domain D_E. A fact $p : \top$ is called an *inconsistent fact*. Interpretations are as before, subsets of the Herbrand base. Without loss of generality every d-fact, p, may appear in I in conjunction with at most one certainty interval. For convenience, if p does not appear in I and no certainty interval is associated with it, then $p : [0,1] \in I$, meaning that no information is available about p and $\neg p$ (i.e., the truth of p and $\neg p$ is undefined). Intervals $[0,0]$ and $[1,1]$ represent the usual *false* and *true*, respectively.

If I is an interpretation, then a ground positive fact, $p : [\alpha, \beta]$, is true in I, if there is $p : [\gamma, \delta] \in I$ such that $[\alpha, \beta] \leq^p [\gamma, \delta]$. A negative fact, $\neg p : [\alpha, \beta]$, is true in I if and only if $p : [1 - \beta, 1 - \alpha]$ is true in I.

Certainty (resp. combination) functions now map sequences (resp. multisets) of intervals (including $\top$) into the set of all certainty intervals plus $\top$.

The partial order on interpretations and other definitions carry over from Section 2.3.2 without much change. Let I and J be a pair of interpretations of E We write $I \subseteq J$ if and only if for every positive fact $p : [\alpha, \beta] \in I$ there is a fact $p : [\gamma, \delta] \in I$ such that $[\alpha, \beta] \leq^p [\gamma, \delta]$. The notions of intersection and union of interpretations carry over directly with the exception that $\leq^p$ replaces the usual ordering on real numbers in the interval $[0,1]$ which was used in Section 2.3.2.

It is now easy to see that the framework of Section 2.3.2 is a special case of support logic once we replace each literal $p : \alpha$ of Section 2.3.2 by a support logic literal of the form $p : [\alpha, 1]$.

Because of negative information (either in the form of negative literals, or as disbelief measures), defining model-theoretic semantic is much more involved than it was in Section 2.3.2. Non-Horn programs may have no unique least model, which is the same case as in Logic Programming. Instead, Non-Horn programs usually have several minimal models, and it is not always clear which one should be preferred. Kifer and Li handled this situation by developing a theory of stratified programs in the framework of support logic.

To develop a theory of stratified programs in the framework of support logic an additional point should be considered, though. The semantics of [1] is a manifestation of the so called, *closed world assumption* (CWA), in which a fact is assumed false unless there is an evidence to prove that this fact is true. However, negation is already present in the current framework in the form of disbelief factors, and we do not always want to deduce a negative conclusion whenever there is a knowledge gap about some fact. Namely, if say, $p(a) : [0.4, 0.8]$, is known then as an answer to the query $p(a) : [\alpha, \beta])$? we expect the answer $p(a) : [0.4, 0.8]$, not $p(a) : [0.4, 0.4]$, as CWA would suggest. As in Logic Programming, Kifer and Li took the position that the intent of deducing a negative conclusion should be explicitly stated in the rule. For this, Kifer and Li viewed some negative body literals as such an explicit declaration, which is, again, consistent with the approach taken in logic programming and deductive databases.

Thus, (OWA) will, sometimes (not always), be used to treat some negative information either in the form of disbelief measure of positive literals or in the form of negative head literals. However, in some other situations the CWA assumption should be used to treat some specially annotated negative body literals. Treating negative literals using CWA has the following consequence. Suppose $p : [0.2, 0.6] \in E$. Then the answer to the query $p : [\alpha, \beta]$? would be $p : [0.2, 0.6]$, while the answer to $p : [1 - \alpha, 1 - \beta]$? is $p : [0.2, 0.2]$, even though the literals in these two queries are logically equivalent.[5] This effect is not a shortcoming of the proposed semantics, but merely a manifestation of different meanings attached to these queries.

It was argued that not all models capture equally well the causality aspect hidden in the syntactic structure of logic rules [16]. The appropriate semantics is given by the, so called, *perfect models* [16]. Kifer and Li extended this notion to accommodate uncertainty.

With the purpose of providing a suitable model-theoretic semantic, Kifer and Li assumed that, in the program, certain occurrences (not necessary all) of negative body literals are specially annotated with "^{o}" to indicate that they should be treated under CWA[6]. Then each occurrence of an annotated negative literal, $\neg q^o : [\alpha, \beta]$ is replaced, by a new positive literal $q_\neg[\alpha, \beta]$. In addition, rule $q(\vec{X}) : [\gamma, \delta] \leftarrow \neg q(\vec{X}) : [\gamma, \delta]$ is added for each symbol $q_\neg$ (and remove the annotation). From now it is assumed that the program P is modified in this way. Predicates of the form $q_\neg$ introduced by this

[5]More precisely, initially we get $\neg p[0.4, 0.8]$. as an answer, but after closing the knowledge gap under CWA the result is $\neg p[0.8, 0.8]$ or, equivalently, $p[0.2, 0.2]$.

[6]It is allowed to treat some occurrences of a literal under CWA, while others under OWA.

modification will be called *CWA-predicates*; the remaining predicates will be called OWA-predicates.

The *predicate dependency graph* is used in support logic to introduce stratification into P. For this, Consider P, its predicate dependency graph, G_P, has predicate symbols of P as its nodes. An unlabeled arc $< q, p >$ between a pair of nodes in the graph exists if and only if P has a rule $\mu\ p(\vec{x}) \leftarrow \ldots, \lambda q(\vec{y}), \ldots$, where μ and λ can be either "$\neg$" or blank (i.e. no negation). A *CWA-cycle* in G_P is a directed cycle passing through at least one CWA predicate. If the dependence graph has no CWA-cycles, then the program is stratified. $E = P \cup D$ is stratified if its intentional part, P, is stratified. A ***stratification ordering*** is defined (in fact, a partial quasi-order) on predicate symbols appearing in a stratified program as: $q \leq p$ if there is a path from q to p in G_P. If at least one node on the path is a CWA-node (including p or q) then we write $q < p$.

Next the notion of *perfect* models in [16] is extended to this framework. First, the order $\leq^p$ defined on OWA ground literals should be extended to include the CWA-literals which were added to replace negative literals in the rule bodies. This is done by assuming that for each CWA d-literal $q : [\alpha, \beta] \leq^p q_\neg[\gamma, \delta]$ if and only if $\alpha \geq \gamma$ and $\beta \geq \gamma$. Notice that the $\leq^p$ ordering on CWA literals differs from the case of OWA literals only in that the first inequality is reversed.

$\leq^p$ is extended further by writing $p(\vec{a}) : [\ldots] <^p q(\vec{b}) : [\ldots]$, where p and q are predicate symbols in P, if and only if $p <^p q$ in the stratification ordering. Of course, with this extension if P is stratified, then $\leq^p$ is no longer a partial order,in general, but it is a partial order if$P$ is stratified. An interpretation I of P is *preferable* to J ($I \ll J$) if and only if for every ground atom $p : [\alpha, \beta]$ which is true in I but false in J, there is a ground atom $q : [\gamma, \delta]$ which is true in J but false in I such that $q : [\gamma, \delta] <^p p : [\alpha, \beta]$. A model of P is said to be *perfect* if it is minimal with respect to the preference relation $\ll$.

In Logic Programming, stratified programs have a unique perfect model, which represents its intended semantics. Moreover, by applying the rules to the partial order imposed by stratification, perfect models is relatively inexpensive to compute.

In this case one never has to retract any fact previously derived in the computation. However, in the quantitative case, stratification alone does not ensure such behavior.

An assumption on the certainty and combination functions should be true and that is to satisfy all the requirements for these functions listed in Section 2.3.2, where, as before, scalar certainty are replaced by intervals of certainties, and the usual order $\leq$ on $[0,1]$ is replaced by $\leq^p$ on the certainty intervals. Again, it is intuitive to see that the requirements introduced on certainty and combination functions in Section 2.3.2 become a special case of the new requirements, once each scalar entity, α, is replaced by the interval $[\alpha, 1]$.

Theorem 3 Every stratified expert system has a unique perfect model.

We will call the unique perfect model of P the *intended model.* This model can be computed as in [1] by successively applying the fixpoint operators corresponding to different strata of the program (in the stratification ordering). We will refer to this process as the *fixpoint computation.* Notice that it may turn out during the computation that the intended model of P contains inconsistent facts of the form $p : [\alpha, \beta]$, where $\alpha > \beta$. However, as noted earlier, this problem is localized to the inconsistent facts themselves, and to the facts which are directly dependent on them.

In what follows we will list the theorems proven by Kifer and Li and present an example used in [10] to illustrate their approach.

Theorem 4 The fixedpoint computation yields the intended model of P.

Example 1 We will present an example taken from [10] that contains negative predicates in the rule bodies to illustrate the approach. Suppose the *EDB* contains a single fact *bird(tweety)* : $[0.7, 0.9]$, and consider the following rule:

$$flies(X) : [min(\alpha,\gamma), max(\beta,\delta)] \leftarrow bird(X) : [\alpha,\beta], \neg abnormal^{p}(X) : [\gamma,\delta].$$

Applying the rule yields $flies(tweety) : [0.7, 1]$. If, in addition, there would be an evidence that Tweety is abnormal, for example, $abnormal(tweety) : [0.4, 0.7]$, then we can only conclude $flies(tweety) : [0.3, 0.9]$, thereby decreasing our belief in Tweety's ability to fly. On the other hand, if we were told that $abnormal(tweety) : [0.7, 1]$ (say, because Tweety looks like a penguin), then the conclusion would be $flies(tweety) : [0, 0.9]$, eliminating our belief in Tweety being a flying creature, while leaving some small evidence to the contrary.

Suppose now that we had another rule, $\neg flies(X) : [\alpha, \beta] \leftarrow abnormal(X) : [\alpha, \beta]$, and the combination function for $flies(\ldots)$ were $F(\{[x, v], [y, w]\}) = [x + y - x * y, v * w]$. Here abnormal is treated under OWA. Assuming $abnormal(tweety) : [0, 0.27]$, strongly suggesting that Tweety cannot fly.

2.3.5 Summary

Kifer and Li presented a model-theoretic and fixpoint semantics for rule-based systems with uncertainty. Their approach to the problem is much more general than the earlier works on that issue [24, 26]. They have also considered some new aspects of the problem such as evidential independence and conflicting evidences. Their treatment of negation accommodates both the closed and the open world assumptions.

Other cases of evidential independence need to be studied. The two independence criteria presented in Section 2.3.2 do not always produce the desired results. For example, under both criteria, combination of the rule $p : \alpha \leftarrow p : \alpha$ and a fact $p : 0.0001$ would lead us to conclude $p : 1$. This conclusion may not be the desired one in many situations. Likewise, given the following three rules: $p \leftarrow b, p \leftarrow a$, and $a \leftarrow p$, we may not want to view the first pair of rules as being independent (because $a \leftarrow p$).

Rules are viewed as being either totally independent, or totally dependent on each other causing the concept of independence to be restrictive. In practice, however, one may want to think of a pair of rules $p \leftarrow a, b$ and $p \leftarrow a, c$ as being only *partially* dependent (because of the common premise a). This means that different evidences supplied by the two rules should not be combined to a full extent, as it would be in the case of totally independent rules. We are currently pursuing several possibilities for accommodating partial independence in the proposed framework.

The assumptions specified on the combination functions and the certainty functions makes it hard to find an adequate application to implement this model which is able to respect these assumptions. However, if there exists a system that is able to make these assumptions true, then in section 2.3.4 these assumptions are no longer that natural as they were in Section 2.3.2. For instance, it may be desirable to assume that, as the belief in $\neg p$ increase, the belief in p should decrease. The monotonicity assumption about the certainty and combination functions rules this possibility out. Extensions to include other important combination rules is a topic for future research.

2.4 Probabilistic Data Model (PDM)

Barbara, Garcia-Molina, and Porter [4] extended the conventional relational data model into a new model called the Probabilistic Data Model (PDM) in which probabilities are associated with the attribute values. Relations in this context are viewed as *probabilistic relations* and have *deterministic* keys, which is a central premise in this model. In other words, each tuple in the probabilistic model represents a known real entity. Nonkey attributes contain information describing the properties of the entities, where such nonkey attributes are deterministic or *stochastic* in nature (to be defined later).

In PDM, each stochastic attribute is handled as a discrete probability distribution function. In other words, the sum of probabilities associated with the attribute values of a given key of an entity in the probabilistic relation should always add up to 1. This requirement will introduce the *missing probability* concept since it is not always possible to know the probabilities of all possible domain values of a given non-key attribute value.

In fact, the model makes use of the missing probability problem by giving various interpretations to the missing probability. In other words, the missing probability allows the model to capture uncertainty in data values as well as in the probabilities. It facilitates the insertion of data into a probabilistic relation, i.e., it is not necessary to have all the information when displaying relations as answers to users queries. For instance a query might be interested in locating all tuples with probability greater than a given value $p \in [0,1]$. As noted in this paragraph, various interpretations could be assumed for missing probabilities. The *PDM* [4] assumed the so called ***no assumptions*** interpretation, for simplicity, to interpret missing probabilities. Another possible interpretation, however, may consider the missing probabilities to be distributed over the domain values not explicitly listed in the probabilistic relation.

In the following section we will present the formal model, semantics of probabilistic relations, some relational algebra operations for the *PDM*, the definition of lossy operations, probabilistic relational decomposition, and some useful new operators that are useful to answer probabilistic queries[7].

2.4.1 Probabilistic Relational Algebra

In this section, the formal model is formally defined as well as some formal probabilistic relational algebra operations. *Single group* relations[8] are defined and introduced first and at the end of this section the concept of a single group relation is extended to define multigroup relations (multirelations) which could be viewed as a collection of single-group relations.

Definition 2 *Probabilistic Relation:* Let K be a set of attributes $K_1, \ldots, K_n$ and A be a set of attributes $A_1, \ldots, A_m$. The domains are given by $dom(K_i)$ and $dom(A_i)$. A probabilistic relation r (or K, A) is a function (many to one) from $dom(K_1) \times \ldots \times dom(K_n)$ into PF. The set PF is a set of probability functions for A. Each $\beta \in PF$ is a function (many to one) from $dom(A_1) \cup \{*\} \times \ldots \times dom(A_m) \cup \{*\}$ into the real number interval $[0,1]$. The symbol $* \notin dom(A_i)$ for $1 \leq i \leq m$, represents instances in which the value of attribute A_i is unknown, i.e., acts as a wildcard. (The symbols $*$ is not allowed in any key domain $dom(K_i)$.) Every β must satisfy the following property:

[7]We define Probabilistic queries as queries that selects tuples/entities from probabilistic relations giving the users the power to specify probabilistic arguments as part of the query syntax.

[8]A single group relation is a probabilistic relation with a set of attributes $K = \{k_1, \ldots, K_n\}$ that works as a set of keys and a stochastic or deterministic set of attributes $A = \{A_1 \ldots, A_m\}$.

$$\sum_a \beta(a) = 1 \tag{2.1}$$

As in the conventional relational database model, a table form is used to represent probabilistic relations. Note that, since PDM shows tuples whose attribute values have non-zero probabilities associated with them, then tuples that have zero values for their attributes are not displayed in the table. Consider the following table that contains probabilistic information on students:

Probabilistic Relation for Students	
Stud – Id	*Address Phone*
12345	0.6 [St. Paul 345-3232] 0.2 [St. Paul 345-3231] 0.1 [St. Paul *] 0.1 [* *]
12346	1.0 [St. Dennis 345-2342]

This probabilistic relation is equivalent to the relation $r(k_1) = \beta_1$, $r(k_2) = \beta_2$, where β_1 and β_2 are defined as follows:

$$\beta_1([St.Paul\ 345 - 3232] = 0.6)$$
$$\beta_1([St.Paul\ 345 - 3231] = 0.2)$$
$$\beta_1([St.Paul\ *] = 0.1)$$
$$\beta_1([*\ *] = 0.1)$$
$$\beta_2([St.Dennis\ 345 - 2342] = 1.0)$$

and all other values are mapped by β_1 and β_2 to 0. We refer to the probabilistic function of a relation as *rows* since they correspond to tuples in the table representation. For instance we will define $r(12345) = \beta_1$ and $r(12346) = \beta_2$ as rows in the probabilistic table for students. Note the interpretation of such a table is as follows: In $r(12345)$ we are told that the probability that *Student-id* = 12345 has *Address* = "*St.Paul*" and *Phone* =345-3232 is exactly 60%, 20% is the probability that *Student-id* = 12345 has *Address* = "*St.Paul*" and *Phone* = 345-3231, and 10% is the probability that *Student-id* = 12345 has *Address* = "*St.Paul*" where the phone number could take any value because it is not known. Note that the last entry (0.1[* *]) in $r(12345)$ is called *the missing probability* and is introduced to indicate that 10% is the unknown probability. In other words, the missing probability is introduced to make sure that every β satisfy property 2.1.

Definition 3 $a' = (a'_1, \ldots, a'_m)$ covers $a = (a_1, \ldots, a_m)$ if for all $i, 1 \leq i \leq m$, either $a'_i = a_i$ or $a'_i = *$.

2.4.2 Semantics for Probabilistic Relations

A probabilistic relation r on K, A defines the probabilities of certain events. One can think of attributes K and A as random variables. If there are no missing probabilities then it is possible to say:

$$r(k)(a) = prob[A = a \mid K = k]$$

where $k = (k_1, \ldots, k_n)$, $a = (a_1, \ldots, a_m)$, $k_i \in dom(K_i)$ and $a_i \in dom(A_i)$. Note that the expression $A = a$ above is shorthand for the event "$A_1 = a_1, \ldots, A_m = a_m$". The event $K = k$ has similar meaning.

With missing probabilities, and a containing no wildcards:

$$r(k)(a) = Prob[A = a \mid K = k] \leq \sum_{a' covers\ a} r(k)(a')$$

That is, the function $r(k)(a)$ gives a lower bound of the probability assigned to a. The upper bound is found by adding probabilities assigned to all $a's$ that cover a.

There is an implicit independence assumption in every probabilistic relation. In other words, it is assumed that probabilities given in r are independent of any other possible database attributes. So, the existence of r in the database indirectly implies that:

$$Prob[A = a \mid K = k \text{ and } B = b] = Prob[A = a \mid K = k]$$

for any set of attributes B such that $B \cap K = B \cap A = \emptyset$.

2.4.3 Relational Algebra Operations for PDM

In this section, some relational algebra operations for the PDM are formally introduced, but some definitions are needed first.

Definition 4 A *Restricted* tuple is defined as follows: Let a be a tuple over a set of attributes A, and let B be a subset of these attributes. Then $a[B]$ is the subtuple that contains the corresponding B elements. For example, if $A = A_1, A_2, A_3$ and $B = A_2, A_3$, then $(a_1, a_2, a_3)[B] = (a_2, a_3)$.

Consider $r' = \prod_\alpha(r)$, where r is a probabilistic relation over K, A and α is a set of attributes $\alpha_1, \ldots, \alpha_m$. Let $A' = A \cap \alpha$, the nonkey attributes that remain in r'. In what follows it is assumed that $A' \neq \emptyset$ (the case where A' is empty is trivial; the key attribute will simply be projected out).

If no missing probabilities are assumed, then the interpretation of r' is that $r'(k)(a') = Prob[A' = a' \mid K = k]$. From basic probability theory, we know that this probability can be computed as follows:

$$Prob[A' = a' \mid K = k] = \sum_{all\ a\ s.t.\ a[A']=a'} Prob[A = a \mid K = k].$$

As a direct consequence from this equation we deduce how to define r', at least for the case with no missing probabilities.

Definition 5 *Projection* ($\prod$). The project of r over α is the following probabilistic relation r' over K, A'. If $r(k)$ is not defined, then $r'(k)$ is not defined either. If $r(k)$ is defined, then $r'(k)$ is the function β defined by:

$$\beta(a') = \sum_{all\ a\ s.t.\ a[A']=a'} r(k)(a).$$

Note that in this case the function β is valid and we have:

$$S = \sum_{a} \beta(a') = \sum_{a'} \sum_{all\ a\ s.t.\ a[A']=a'} r(k)(a).$$

If an arbitrary tuple $z = (z_1, \ldots, z_m)$, (where $z_i \in dom(A_i) \cup \{\}$). It will be included in the sum S sum when $a' = z[A']$ and $a = z$ (and in no other case). So, since all z tuples are being considered once in the sum we have:

$$S = \sum_{a} r(k)(a) = 1$$

Example 2 Consider the probabilistic relation r for students presented in section 2.4.1 and let us project on the *Stud-Id* and *Address*. In this case we have $K = \{Stud-Id\}$ and $A = \{Address\}$. let $r' = \prod_{K,A}(r)$.

Probabilistic Relation for Students Addresses	
Stud − Id	*Address*
12345	0.9 [St. Paul] 0.1 [*]
12346	1.0 [St. Dennis]

This example illustrates that for the *stud-id* = 12345 there is 90% certainty that his address is "*St. Paul*" and 10% is the missing probability.

Definition 6 A selection condition $C(k,r)$ evaluates to true if row k of r satisfies the selection condition. A specified syntax is considered. A selection condition is of the form $\mathbf{V} = a$, $\mathbf{P}$ *op* p (or v= a, p *op* p), where a is a tuple with possible $*'s$, *op* is an operation like $<, >, \ldots$, and p is a real number in $[0,1]$. Recall that **V**,**P** are used for certainty conditions and **v**,**p** for possibility condition.

Definition 7 The conditions $C(k,r)$: $\mathbf{V} = a$, $\mathbf{P}$ *op* p when *op* is $>, \geq, =$; and $C(k,r)$: $\mathbf{v} = a$, $\mathbf{p}$ *op* p when *op* is $<, \leq$, are evaluated as follows. Let A' be the attribute of r where a has no $*'s$, and let $a' = a[A']$. Let $r_j = \prod_{A'}(r)$. Then $C(k,r)$ is true if $r_j(k)(a')$ *op* p is true.

Projection ensures that the probability referred to is at least the value obtained for a'. Therefore, when *op* is $>, \geq, =$, and $r_j(k)(a')$ *op* p is true, then it is certain that the same will be true for a. On the other hand, when *op* is $<, \leq$, and $r_j(k)(a')$ *op* p is true, then there exist the possibility that the value in a does not go beyond the one obtained for a'.

Definition 8 The condition $C(k,r)$: $\mathbf{v} = a$, $\mathbf{p}$ *op* p when *op* is $>, \geq, =$; and $C(k,r)$: $\mathbf{V} = a$, $\mathbf{P}$ *op* p when *op* is $<, \leq$ are evaluated as follows. Again, let A' be the attributes of r where a has no $*'s$, and let $A' = a[A']$. Let $r_j = \prod_{A'}(r)$. Then $C(k,r)$ is true if:

$$\sum_{\text{all } x \text{ s.t. } x \text{ covers } a'} r_j(k)(x)\ op\ p$$

is true.

Here, with respect to definition 7 the situation is reversed. When op is $\leq$ or $<$, and the sum for all tuples that covers a' evaluates to true, then this implies that we are certain that the value of the probability will never be greater, or greater or equal than the one obtained in the sum. If op is $>, \geq$ or $=$, there is the possibility that the value is greater, greater or equal or simply equal to the one obtained in the sum.

Definition 9 *Select* (σ). *Let K, A be a probabilistic relation defined as r. Moreover,* Let $r' = \sigma_C(r)$ be the relation obtained by the operation select over the probabilistic relation r. Probabilistic relation r' is also defined on K, A. If $r(k)$ is undefined or if $C(k, r)$ is false, then $r'(k)$ is undefined. Otherwise, $r'(k) = r(k)$.

Example 3 Consider the possibility relation r on students and suppose we would like to select all tuples from r that have $Address =$ "St. Paul" with a certainty greater than 0.8. In other words, the query is asking if we assume a certainty (not possibility) 0.8, can we find any tuples where $Address =$ "St. Paul". The result of this query is shown below:

$r' = \Pi_{K,A}(r)$	
Stud – Id	*Address Phone*
12345	0.6 [St. Paul 345-3232]
	0.2 [St. Paul 345-3231]
	0.1 [St. Paul *]
	0.1 [* *]

In this case K = *Stud-Id* and A = *Address, Phone.*

Definition 10 *Natural Join* ($\bowtie$). Consider two relations, r on K, A, and s on A, B. Let r' be the natural join of r and s. Relation r' is over K, AB. If $r(k)$ is not defined, then $r'(k)$ is not defined. Otherwise the function is defined as follows:

If $s(a)$ is defined then

$$r'(k)(ab) = r(k)(a) \times s(a)(b)$$

else if $s(a)$ is not defined then

$$r'(k)(a*) = r(k)(a)$$

$$r'(k)(ab) = 0 \text{ for all } b \text{ other than } *.$$

The second part of the definition covers the case in which a does not appear in the relation s. In this case, the resulting row contains an unknown value for attribute b, and the probability equals that of row $r(k)(a)$.

Note that if $a = (a_1, \dots, a_m)$ and $b = (b_1, \dots, b_o)$, then ab is the tuple $(a_1, \dots, a_m, b_1, \dots, b_o)$. The tuple $a*$ represents $(a_1, \dots, a_m, *, \dots, *)$.

If no missing probabilities are present, this definition is justified by basic probability theory. The value $r'(k)(ab)$ should be equal to $Prob[AB = ab \mid K = k]$. Using Bayes' theorem, we get:

$prob[AB = ab \mid K = k] = Prob[B = b \mid K = k \text{ and } A = a]\, prob[A = a \mid K = k]$.
Because of our independence assumption, the probability of $B = b$ only depends on A, not on K. Thus

$$Prob[AB = ab \mid K = k] = Prob[B = b \mid A = a] Prob[A = a \mid K = k]$$

This is the formula that is used in Definition 10.

Definition 11 *Multirelations:* A table with multiple groups is represented by a multirelation. A Multirelation R on attributes $K, A_1, \dots, A_q$ is a tuple $(r_1, \dots, r_q)$ where each r_i is a probabilistic relation on K, A_i. There is one condition (we call it the "same keys" constraint) that must be satisfied by the relations: if $r_i(k)$ is defined, then for all $j, 1 \leq j \leq q$, $r_j(k)$ must also be defined.

Relational operators on multirelations can be defined in terms of the operators on the component relations. Let $R = (r_1, \dots, r_q)$ be a multirelation on $K, A_1, \dots, A_q$.

Project (Π) : $\Pi_a(R)$ is another multirelation defined as $(r'_1, \dots, r'_q)$, where $r'_i = \Pi_a(r_1)$.

Select (σ) : $\sigma_{c_0,c_1,\ldots,c_q}(r_i)$. The selection condition c for R has $q+1$ components, $c_0, c_1, \ldots, c_q$. The condition c_i is intended to select within relation r_i. Condition c_0 selects keys. If no selection is desired for r_i, then c_i is set to "true".

$\sigma_{c_0,c_1,\ldots,c_q}(r_i) = \sigma_d(r_i)$. Condition $d(k, r_i)$ is true if and only if $c_0(k)$, $c_1(k, r_1)$, ..., $c_q(k, r_q)$ are all true which implements a logical AND of the conditions.

Natural Join $(\bowtie)$: Let $S = (s_1, \ldots, s_t)$ be a second multirelation on $A_i, B_1, \ldots, B_t$. $R \bowtie S$ is

$$(r'_1, \ldots, r'_{i-1}, r'_{i+1}, \ldots, r'_q, s'_1, \ldots, s'_t)$$

, where $r'_j = r'_j (j \neq i)$, and $s'_j = r_i \bowtie s_j$.

For each of the above definitions, it is relatively simple to show that the result is a valid multirelation, i.e., the same keys constraint holds. Deterministic attributes have the same concept as that introduced in an earlier section. In the case of multirelations, deterministic attributes are just like probabilistic ones except that the probability function always assigns probability 1.0 to a single value. When such attributes are displayed in a table the probabilities are stripped away.

Example 4 Consider a University's database and let the two probabilistic relations r and s be as follows. r is the probabilistic relation on students registration. s is the probabilistic relation on full time professors.

Probabilistic relation r represents the probability that a student will register in a course (not the actual or the current registration). For instance, The first tuple of r indicates that there is 60% chance that the student, whose *Id* is 12345, will take COMP 664 and 20% chance that he will take COMP772 etc. Notice that there are no missing probabilities in r.

Probabilistic relation s represents the probability that a professor will give a course for the coming term. For example, in the first tuple there is a probability of 40% that John will be giving COMP 664, while Mike have a higher probability to teach the same course. Note that there are no missing probabilities in s.

Student Registration (r)	
Stud-Id	*CrsNumber*
12345	0.6 [COMP 664]
	0.2 [COMP 772]
	0.1 [COMP 345]
	0.1 [COMP 777]
12346	0.5 [COMP 520]
	0.3 [COMP 575]
	0.1 [COMP 543]
	0.1 [COMP 745]

Full Time Professors (s)	
CrsNumber	*Prof*
COMP 664	0.4 [John]
COMP 664	0.6 [Mike]
COMP 345	0.9 [Peter]
COMP 345	0.1 [Mary]
COMP 777	1.0 [Mike]
COMP 745	1.0 [Mary]

Probabilistic relation $r \bowtie s$ contains missing probabilities because of the tuples that did not join. The first tuple in $r \bowtie s$ indicates that there is 26% chance that student, whose *Id* is 12345, will take COMP 664 with John and 36% chance that the same student will take COMP 664 with Mike. Similar interpretations hold for the other tuples.

($r \bowtie s$)	
Stud-Id	*CrsNumber Prof*
12345	0.6×0.4 [COMP 664 John]
	0.6×0.6 [COMP 664 Mike]
	0.1×0.9 [COMP 345 Peter]
	0.1×0.1 [COMP 345 Mary]
	0.1×1.0 [COMP 777 Mike]
	0.2 [* *]
12346	0.1×1.0 [COMP 745 Mary]

2.4.4 Lossy Operations

We realize from the previous definitions of project and join that these operators generate new probability distribution functions in the result. The operators join, select, and project were presented when no missing probabilities were involved. The project and join operators become more and more complicated when the probabilistic relations in use have missing probabilities. In this section, the issue of whether the new operators are still meaningful when missing probabilities are involved is addressed. Before we proceed any further we need some definitions.

Definition 12 *Potential Set:* Given a tuple with key k in a probabilistic relation r, the potential set of $r(k)$, or simply $PSET(r(k))$, is defined as the set of all functions (without missing probabilities) that can be obtained from $r(k)$, by each one assuming a particular distribution of the missing probabilities. More formally, $PSET(r(k))$ is the set of all valid probabilistic functions β such that

i) for all tuples a with $*'s$, $\beta(a) = 0$;

ii) for all tuples a with no $*'s$:

$$r(k)(a) \leq \beta(a) \leq \sum_{a'} r(k)(a').$$

By extension, the $PSET$ of a relation r is simply defined as:

$$PSET(r) = \{r' \mid r'(k) \in PSET(r(k)), \text{ for all } k\}.$$

Definition 13 *Probabilistic Lossless Operation.* Let $\oplus$ be an operation over relations $r_1, \ldots, r_n$ such that $r = \oplus(r_1, \ldots, r_n)$. The operation is probabilistically lossless if for all k $r(k)$ is defined as either

i) $r(k) = r_i(k)$ for some $i, 1 \leq i \leq n$ or

ii)

$$PSET(r(k)) = \bigcup_{all\ r'_i \in PSET(r_i)} PSET[\oplus(r'_1, \ldots, r'_n)(k)].$$

In part (ii) of definition 13, notice that even though $r'_1, \ldots, r'_n$ have no missing probabilities, $\oplus(r'_1, \ldots, r'_n)(k)$ may have missing probabilities (in particular if $\oplus$ is a join). This makes necessary for the application of the *PSET* function to $\oplus(r'_1, \ldots, r'_n)(k)$.

In *PDM* the following theorems were proven correct.

Theorem 5 Project is a probabilistically lossless operation.

Theorem 6 Select is a probabilistically lossless operation.

Theorem 7 If a relation r has property NMP[9], then $r_1 \bowtie r_2$ is probabilistically lossless, for any relation r_2.

2.4.5 Decomposing Relations

We introduced, so far, possible information loss at the level of probability distribution functions. It is also possible to study information loss in the conventional sense [27]. That is, assume we have a relation K, AB. It is always possible to decompose t into two relations r over K, A and s over A, B such that $t = r \bowtie s$. However, since t should be recovered after the decomposition is done to ensure the correctness of the decomposition, the decomposition should follow some non trivial steps before it is done. A condition should be satisfied before any decomposition is made. The condition under which decomposition is not lossless is stated below.

Definition 14 *Probabilistic Independence:* Given a probabilistic relation t over K, AB, we say that B is probabilistically independent from K if the following is true

$$Prob[B = b \mid A = a, K = k] = Prob[B = b \mid A = a].$$

Decomposition of t involves two steps. The first step is the projection over attributes K, A, to generate r. To generate s we need a new operator that extracts conditional probabilities and an extension of the project operator that allows projection over subsets of the key.

[9]A relation r has property NMP (No Missing Probability) if no row in r is a function with missing probabilities.

Definition 15 *Conditional Operator* (*COND*): Given a relation r with key K and attributes AB, the conditional operator converts r into r' with key KA and attribute B. The value $r'(ka)(b)$ is equal to $Prob[B = b \mid K = k \text{ and } A = a]$. This value is computed as (Bayes Rule)

$$r'(ka)(b) = Prob[B = b \mid K = k \text{ and } A = a]$$
$$\frac{Prob[B=b \text{ and } A=a|K=k]}{\sum_{b'} Prob[B=b' \text{and} A=a|K=k]}$$
$$\frac{r(k)(ab)}{\sum_{b'} r(k)(ab')}$$

The syntax of the condition operator is denoted as $COND\ r\ OVER\ \alpha$, where α is the set of attributes that are pulled into the key of the new relation.

Definition 16 *Strip.* Given a probabilistic relation r on K, A, the operation $Strip(r)$ results in a probabilistic relation in which all the no information rows have been deleted.

It is obvious now that the significant use of the $Strip(r)$ operation to a relation r allows us to get rid of the useless rows. However, before using this operator, we should be careful so that stripping useless tuples does not somehow affect our results. The following theorem illustrate that performing a *Strip* operation on Join of two relations is exactly the same as performing the *Strip* on separate relations and then performing the Join between them. The proof of this theorem is not included here, but it was proven in the PDM that this theorem is valid.

Lemma 2 The operation Join and Strip are distributive.

$$Strip(r_1 \bowtie r_2) = Strip(r_1) \bowtie Strip(r_2)$$

We need to introduce the notion of projecting out a subset of the key, in a probabilistic relation K, A, before providing how to decompose a relation. In other words, given a relation r with key attributes KA and attribute B, we want to create a relation with key A and attribute B. The value $r'(a)(b)$ for all possible values of k. In the case of probabilistic independence between K and B, $r(ka)(b)$ is the same for all values k, hence, $r'(a)(b) = r(ka)(b)$ for any k.

Definition 17 *Projecting Out a Subset of the Key:* Consider a relation r with key attributes KA and attribute B. The projection of r onto $\alpha = AB$ is a relation r' with key attributes A and attributes B. If K and B are probabilistically independent, $r(ka)(b)$ is the same for any k, hence, $r'(a)(b) = r(ka)(b)$ for any k. If K and B are not independent, the project operation is not defined.

Lemma 3 Given a relation t over K, AB, where K, B are probabilistically independent, it is possible to decompose t in two relations r over K, A and s over A, B such that $r = \Pi_{K,A}(t)$; $s = \Pi_{A,B}(t')$, where $t' = COND\ t\ OVER\ A$ and $t = r \bowtie s$.

2.4.6 New Operators

So far we presented the formal PDM, the semantics, some operators (join, select, and project), and the method to decompose probabilistic relations specially when we have missing probabilities in the probabilistic relations. In some application domains it is important that the user is able to specify queries where probabilistic arguments are passed to the query processor. In this case, the query processor will use some probabilistic theories like the *Total Variation Distance Between Two Probability Functions* introduced next. In the coming sections we will introduce some operators that can model these situations.

Definition 18 *Total Variation Distance Between Two Probability Functions:* The total variation distance between two probability functions β_1 and β_2 is computed as follows:

$$d(\beta_1, \beta_2) =: \frac{1}{2} \sum_{a} \mid \beta_1(a) - \beta_2(a) \mid$$

Now, since some probabilistic relations contain missing probabilities, definition 18 is no more efficient. In this model, however, every relation $r(k)$ with missing probabilities could be used to compute $PSET(r(k))$, which is the set of distribution (no missing probabilities) that corresponds to a distribution $r(k)$ (in relation r) with no missing probabilities. Using the definition of the $PSET$, the maximum distance which is the largest value for total variation distance over all the combinations of the elements in the potential set of distributions involved will be considered instead of the total variation distance between two probability functions. In the same sense, finding the minimum distance should be obvious.

Definition 19 Maximum and Minimum Distances. Given two distributions β_1 and β_2, the maximum between them is computed as:

$$maxd(\beta_1, \beta_2) = max(d(\beta'_1, \beta'_2))$$

where $\beta'_1 \in PSET(\beta_1)$ and $\beta'_2 \in PSET(\beta_2)$. Conversely, the minimum distance between them is computed as:

$$mind(\beta_1, \beta_2) = min(d(\beta'_1, \beta'_2))$$

where $\beta'_1 \in PSET(\beta_1)$ and $\beta'_2 \in PSET(\beta_2)$.

Having defined the maximum and minimum distances between two probabilistic entities we now concentrate on the syntax of some useful operators that are able to accept probabilistic arguments and use the probabilistic relations with the purpose of increasing the power of expressing more meaningful queries.

Definition 20 *ϵ-SELECT and E-SELECT* : Given a relation r on K, A, and a parameter ϵ (or E) and a target distribution β, the relation $r' = \epsilon\text{-}SELECT\ r\ USING\ \beta$ is the relation formed by rows $r(k)$ such that $mind(\beta, r(k)) \leq \epsilon$. The relation $r' = E\text{-}SELECT\ r\ USING\ \beta$ is the one formed by rows $r(k)$ such that $maxd(\beta, r(k)) \leq E$.

Here the distinction between the *possibility* and the *certainty* should be cleared. In other words, the condition $min\ d \leq \epsilon$ yields the possibility that the actual distance between the two distributions is less than or equal to the target value. On the other hand, the condition $max\ d \leq E$ yields the certainty that the actual distance is less than or equal to the target value.

Definition 21 *ϵ-JOIN and E-JOIN:* Given two relations, r on K, B and s on A, B, and a parameter ϵ, let r' be the ϵ-*Join* over B. Relation r' is over KA, B. If

$r(k)$ is not defined, then $r'(k)$ is not defined. Otherwise, the function is defined as follows:

$r'(ka) = r(k)$ if $s(a)$ is defined and $mind(r(k), s(a)) \leq \epsilon$.

$r'(ka) = 0$, otherwise.

Conversely, the *E-Join* is defined in the same way replacing *min d* by *max d*, and ϵ by E.

Given a deterministic relation and a probabilistic schema, the *STOCHASTIC* operator takes as input the deterministic relation and the probabilistic schema returning a probabilistic relation based on that schema. The probabilistic schema describes what attributes comprise the KEY, and which of the dependent attributes are jointly distributed or independent. The relative frequencies of the attribute values in the original relation will determine the probabilities of the new relation. Hence, one deterministic relation can generate many different probabilistic relations.

Operator *DISCRETE* goes the opposite direction as *STOCHASTIC* operator. *DISCRETE* takes as input a probabilistic relation and yields a deterministic relation as output.

The last operator to mention is the *GROUP* operator. The *GROUP* operator combines two or more attributes in a relation into a single group. In doing this, it computes the join probability distribution for the new group. Operator *GROUP* exemplifies a multirelation operator, i.e., one for manipulating multirelations. Other operators in this context include *CUT*, which breaks up a multirelation into its components, and *GLUE*, which produces a multirelation out of a collection of simple relation.

2.4.7 Summary

A Probabilistic Data Model *PDM* is a model in which tuples have deterministic keys and both probabilistic and deterministic attribute groups. Some problem domains that deals with incomplete information could be modeled using the *PDM*. A null value, for instance, could be represented by 1.0[**] distribution. Missing probabilities is another important concept that was introduced in the *PDM*. This concept is important because based on this method we would like to distribute the missing probability, we can have different relational algebra operators that are able to manipulate the probabilistic relations. In the *PDM* the missing probability was interpreted as distributed over all domain values. As another interpretation, missing probabilities could be distributed over values not explicitly listed in the relation. Probability distribution function could also be used to assign probabilities to attribute values instead of assigning probability to each attribute value separately.

Sometimes it is desirable to have the same set of data where different distribution functions are associated to them. The *PDM* could further be extended to capture this idea. An operator could be introduced that is able to take two distribution functions that corresponds to the same set of data and combine the distribution functions yielding one distribution function associated with the same set of data. Of course, this requires a clean and correct semantics as well as a meaningful interpretation in providing answers to queries.

As an interesting application that could be a consequence of the *PDM* a restricted model in which only distributions with one probabilistic value are allowed. For instance, one may model evidences like the probability of having a storm at a particular time is p and hence the probability of not having a storm at the same particular time is $1 - p$. In some real life applications, the restricted model is useful and simple enough to be implemented.

2.5 The Information Source Tracking Model (IST)

In this section, we review the basic concepts of the IST proposed by Sadri [18]. An IST database is a set of *extended relation schemes*. An *extended relation scheme (relation scheme,* for short) $R = \{A_1, \ldots, A_n, A_I\}$ is the set of attributes, where $A_1, \ldots, A_n$ are called *regular attributes* as in the regular relational model, and A_I is a special attribute called the *information source attribute, (source attribute* for short). Let $S = \{s_1, \ldots, s_k\}$ be the set of all information sources in an IST database. We will refer to an information source $s_i \in S$ as a source. The domain D_I of the attribute A_I is the set of all vectors of length k with entries $\{+1, -1, 0, \top\}$. That is,

$$D_I = \{(a_1, \ldots, a_k) \mid a_i \in \{+1, -1, 0, \top\}, i = 1, \ldots, k\}$$

where k is the number of source vectors.

Let $R = \{A_1, \ldots, A_n, A_I\}$ be an *extended relation scheme.* An *extended relation (instance)* r of R is a finite subset of $D_1 \times \ldots \times D_n \times D_I$, where $D_1, \ldots, D_n$ are domains corresponding to the normal attributes $A_1, \ldots, A_n$, and D_I is the domain of A_I. A *database scheme* is a set of relation schemes $R_1, \ldots, R_m$. An IST database $\mathbf{D}$ is a set of extended relations $r_1, \ldots, r_m$, where r_i is defined over the relation scheme $R_i \in R'$, for $i = 1, \ldots, m$. Moreover, each relation $r_i \in \mathbf{D}$ is called a *base relation*, as usual.

Each tuple in a base relation r_i is of the form $t@u$, where t is the component of the tuple corresponding to the normal attributes $A_1, \ldots, An$, and $u \in D_I$ is the component corresponding to the source attribute A_I. If $t@u$ is a tuple in r_i, then we refer to t as the *pure tuple*, we refer to u as the source vector associated with t. The user may not know that the tuples in the database are associated with source vectors. In the IST model, it is allowed that a tuple in an extended relation to be associated with a set of source vectors, rather than a single source vector. For instance, an extended relation r may contain a tuple $t@x$, where $x = \{u_1, \ldots, u_l\}$ and $l \geq 1$. In this case, $t@x$ is a tuple associated with a set of source vectors in the extended relation r. This gives the flexibility to record the same information (evidence) that is contributed by different observers at different times an issue that could not be modeled in other approaches.

2.5.1 Information Source Vectors

Let r be an extended relation and let $t@u \in r$. The source vectors that are contributing to the pure tuple t in r are listed in u. If source $s_i = (a_1, \ldots, a_k)$, where k is the number of source vectors in the model, confirmed the information for the pure tuple t then the i^{th} position of the source vector u is set to one and all the other elements are set to zero indicating that they did not contribute to the information in the pure tuple t. An element $a_i = -1$ indicates that the source vector s_i denies the information in the pure tuple t. An element $a_i = 0$ indicates that the source s_i did not contribute to the information in the pure tuple t in the extended relation r. There are two special source vectors T and F, representing that the pure tuple t is *true* and *false* respectively.

A source vector $u = (a_1, a_2, .., a_k)$, associated with a pure tuple t in r specifies the conjunction of the meaning of each information source s_i as given by a_i. A set of source vectors $x = \{ u_1, \dots, u_m \}$ specifies the disjunction of the specification of the vectors $u_1, \dots, u_m$. More precisely, the meaning of information sources is captured by the expression corresponding to a set of source vectors defined as follows:

A boolean variable f_i is associated with each information source $s_i, i = 1, \dots, k$. For a source vector $u = (a_1, a_2, \dots, a_k)$, the set of information sources that are contributing positively to u is $S^+ = \{s_i \mid a_i = +1\}$. Similarly, the set of information sources that are contributing negatively to v is $S^- = \{s_i \mid a_i = -1\}$.

Having classified the behavior of the contributing sources we associate a boolean expression $e(u)$ corresponding to the source vector u, associated with the pure tuple t in r as:

$$e(u) = \bigwedge_{s_i \in S^+} f_i \bigwedge_{s_j \in S^-} \neg f_j.$$

This definition can be extended to a set x of source vectors associated with a pure tuple t in r as:

$$e(x) = \bigvee_{u \in x} e(u),$$

The expression corresponding to the source vectors T and F is *true* and *false* respectively. The special source vector $\top$ denotes a contradiction. If there exists a source s_i that supplied contradictory information about the pure tuple t in r, we define $e(\top) = \mathit{false}$.

The expression corresponding to the pure tuple t could be used to identify the contributing information sources as well as to derive the reliability of the information stored in the pure tuple t. In other words, given the probabilities (reliability of being correct) of sources $s_1, \ldots, s_k$, we can calculate the probability of the validity of pure tuple t. Calculating the reliabilities of answers to queries, which could be optional, could be done as the last step by the query processor.

2.5.2 Source Vector Operations

The extended relational algebra operations are defined in terms of the source vector operations *conjunction* $\wedge$, *disjunction* $\vee$, and *negation* $\neg$. In what follows we will explicitly define these operations.

Note that the domain of the source attribute $A_I = \{+1, -1, 0, \top\}$ constitutes a lattice, where $\top$ is the top element and 0 is the bottom element. The partial order $\preceq$ among the elements of A_I is defined as $0 \preceq 1 \preceq \top$ and $0 \preceq -1 \preceq \top$. Given two source vectors $v = (a_1, a_2, \ldots, a_k)$ and $w = (b_1, b_2, \ldots, b_k)$, their conjunction $u = v \wedge w$ is a source vector $u = (c_1, c_2, \ldots, c_k)$, where $c_i = lub(a_i, b_i)$ is calculated based on the table below:

a_i	b_i	c_i	a_i	b_i	c_i	a_i	b_i	c_i	a_i	b_i	c_i
0	0	0	+1	0	+1	−1	0	−1	⊤	0	⊤
0	−1	−1	+1	−1	⊤	−1	−1	−1	⊤	−1	⊤
0	+1	+1	+1	+1	+1	−1	+1	⊤	⊤	+1	⊤
0	⊤	⊤	+1	⊤	⊤	−1	⊤	⊤	⊤	⊤	⊤

The source vector conjunction operation of two sets of source vectors x and y is defined as:

$$x \wedge y = \{u \wedge v \mid u \in x \text{ and } v \in y\}.$$

The disjunction operation of two source vectors x and y, denoted by $x \vee y$, is their union, i.e., $x \vee y = x \cup y$.

The negation of a source vector u is defined as follows: Let $u = (a_1, \ldots, a_k)$ be a source vector, and let $a_{i1}, \ldots, a_{im}$ be the non-zero elements of u. The negation of u, written $\neg(u)$, is a set of source vectors $\{v_{i1}, \ldots, v_{iq}\}$ constructed as follows: All the elements of v_{ij} are zero except the element corresponding to the position ij, which is $1(-1)$ if $a_{ij} = -1(1)$.

The negation of a source vector u could be extended to the negation of a set of source vectors x, where $x = \{u_1, \ldots, u_p\}$ and p is the number of times pure tuple t appears in r, is as follows:

$$\neg(x) = \neg(u_1) \wedge \ldots \wedge (u_p).$$

The following theorems were proven in [18], where x, y, and z are sets of source vectors and $e(x), e(y)$, and $e(z)$ are their corresponding expressions, respectively.

Theorem 8 Let $z = x \wedge y$. Then $e(z) = e(x) \wedge e(y)$.

Theorem 9 Let $z = x \vee y$. Then $e(z) = e(x) \vee e(y)$.

Theorem 10 Let $z = \neg(x)$. Then $e(z) = \neg(e(x))$.

2.5.3 Extended Relational Algebra Operations for IST

In this section, we review the definition of the extended relational algebra operations in the IST model. These include σ (Selection), $\prod$ (Projection), $\cup$ (Union), $\times$ (Cartesian Product), $\bowtie$ (Join), and $-$ (Difference).

In the following definitions, r_1 and r_2 are two extended relation instances on the extended schemes R_1 and R_2 respectively. C is the selection condition in $\sigma_C(r_1)$ and can include attributes from $R_1 - \{A_I\}$. X in $\prod_X(p)$ denotes the set of joining attributes where $X \subseteq R_1 - \{A_I\}$.

$\sigma_C(r_1) = \{t@u \mid t@u \in r$, and t satisfies condition $C\}$

$\prod_X(r_1) = \{t[X]@u \mid t@u \in r\}$

$r_1 \cup r_2 = \{t@u \mid t@u \in r_1$ or $t@u \in r_2\}$

$r_1 \cap r_2 = \{t@w \mid t@u \in r_1, t@v \in r_2$, and $w = u \wedge v\}$

$r_1 \times r_2 = \{t_3@w \mid t_1@u \in r_1, t_2@v \in r_2, t_3 = t_1.t_2$, and $w = u \wedge v\}$

$r_1 \bowtie r_2 = \{t_3@w \mid t_1@u \in r_1, t_2@v \in r_2, t_3 = t_1 o t_2, w = u \wedge v$, and t_1 and t_2 join$\}$

$r_1 - r_2 = \{t@x \mid t@x \in r$, and pure tuple t does not appear in $r_2\} \cup$
$\{t@x \mid t@y \in r_1$ and $t@z \in r_2$ and $x = y \wedge (\neg z)\}$

where $t_1.t_2$ indicates the concatenation of t_1 and t_2, and $t_1 o t_2$ indicates the join of t_1 and t_2, i.e., the concatenation of t_1 and t_2 with the removal of duplicate values of common attributes. Two tuples t_1 and t_2 join if they have the same values for the common attributes. Note that tuples t_1 and t_2 are pure tuples and do not contain values for information source attribute A_I.

2.5.4 Reliability Calculation

knowing the reliability of information sources that contributed to a pure tuple t in an extended relational database system, we can calculate the reliability of answers to a query to the extended relational database.

Definition 22 The reliability of a source is defined as the probability that a tuple coming from that source is valid. We designate the reliability of source j by $re(j)$. Moreover, We assume that different information sources are independent.

Let r be an extended relation which is the result of some query and let $R = \{A_1, \ldots, A_n, I\}$ be the extended relational scheme of r. Consider a tuple $t@u \in r$ and let $u = (a_0, a_1, \ldots, a_n)$, where $a_i \in \{1, 0, -1, \top\}$ corresponds to the nature of contribution, to the pure tuple t, of the information source s_i.

Given the reliability of the contributing information sources, we would like to calculate the reliability or the probability that tuple t is in the answer to the query. Since u is the source vector associated with the pure tuple t in r, we consider the sources $s_{i_1}, \ldots, s_{i_p}$ corresponding to the elements of u with a value of 1. Formally, $a_{i_1} = \ldots a_{i_p} = 1$, where $\forall l \in \{i_1, \ldots i_p\}$, u_l represents the l^{th} element of the source vector u, associated with t, s.t. s_l is contributing positively to $t \in r$. Similarly, consider the sources $s_{j_1}, \ldots, s_{j_q}$ corresponding to the elements of u with a value of -1. Formally, $a_{j_1} = \ldots = a_{j_q} = -1$, where $\forall j \in \{j_1, \ldots j_q\}$, u_j represents the j^{th} element of the source vector u associated with t. Given the reliability of source vectors $s_1, \ldots, s_k$, the reliability of the pure tuple t denoted as $rel(t)$ is:

$$rel(t@u) = \prod_{l \in \{i_1, \ldots i_p\}} re(l) \prod_{k \in \{j_1, \ldots j_q\}} re(j) \tag{2.2}$$

$rel(t@u)$ indicates the probability or the reliability under which pure tuple t exists in r. In other words, pure tuple t exists in r provided that the sources $s_{i_1}, \ldots, s_{i_p}$ are reliable (or correct) and the sources $s_{j_1}, \ldots, s_{j_q}$ are not reliable (or not correct).

Definition 23 Two source vectors u and v associated with $t \in r$ are said to be *independent* if for no source j both of them have a nonzero entry. This definition could be extended to define independence between sets of source vectors. A set of source vectors is *independent* if the source vectors are pairwise independent.

Let $t@x \in r$, where $x = \{u_1, \ldots, u_p\}$, where $\{u_1, \ldots, u_p\}$ is the set of independent source vectors associated with pure tuple $t \in r$. The reliability of t is calculated as follows:

$$rel(t) = 1 - \prod_{i=1}^{p}(1 - rel(t@u_i)) \tag{2.3}$$

2.5.5 Reliability Calculation Algorithms

In the previous Equation 2.3, it is assumed that the source vectors contributing to the pure tuple t are independent. However, in some application domains, sources might be dependent one on the other. The following two Algorithms were introduced by Sadri in [18, 19] to calculate efficiently the reliability associated with the answers to queries when source vectors are dependent.

Algorithm 1 Assume $t@x \in r$, $x = \{u_1, \ldots, u_p\}$. Let

$$K_1 = \sum_{i=1}^{p} re(u_i)$$

$$K_2 = \sum_{i=1}^{p} \sum_{j>i}^{p} re(u_i \wedge u_j)$$

$$K_3 = \sum_{i=1}^{p} \sum_{j>i}^{p} \sum_{k>j}^{p} re(u_i \wedge u_j \wedge u_k)$$

$$\ldots$$

Then

$$re(t) = K_1 - K_2 + K_3 - \ldots + (-1)^{p-1}K_p \tag{2.4}$$

Before we proceed to the second Algorithm we need some definitions.

Definition 24 Let r be an extended relation and let $t@u \in r$, where $u = (a_1, \ldots, a_k)$ and k is the number of information sources in the model (assumed to be a constant). We define the set of contributing information sources that contributes $t@u \in r$ as:

$CON = \{s_i \mid a_i \neq 0$, where a_i corresponds to the contribution of source $s_i\}$

This definition could be extended to find C, the set of contributing information sources that contributes to $t@x$, where $x = \{u_1, \ldots, u_p\}$. In this case we define C as:

$$C = \{s_i \mid \exists u_i, i = 1, \ldots, p,\ where\ a_i \neq 0\}$$

Algorithm 2 Consider the expression for $t@x$, where $x = \{u_1, \ldots, u_p\}$ is a set of dependent source vectors associated with t in r. As pointed out by Sadri in [18, 19], the idea is to find an equivalent set V corresponding to a disjunctive normal form, and then calculate the sum of their reliabilities. For obtaining the set V we proceed as follows. Let C be the set of contributing information sources that contributes to $t@x \in r$. We define a vector v to be in standard form if all entries in v corresponding to sources in C have values from the set $\{1, -1\}$.

Based on this, we define $V = \{v_1, \ldots, v_l\}$, where all v_i's are in standard form, obtained by replacing each $u_i \in x, i = 1, \ldots, p$, by its equivalent standard form set, and eliminating duplicates (this is the important point: by eliminating duplicates we are making sure that the calculated reliability is correct).

$$e(t@x) = \bigvee_{i=1}^{p} e(t@u_i)$$

This expression is in disjunctive form, (also called "sum-of-products") where each u_i is a conjunct. Convert this expression into *disjunctive normal form*, i.e.

$$e(t@x) = \bigvee_{i=1}^{q} e(t@v_i)$$

where each $e(t@v_i)$ is a conjunct in which all variables $f_1, \ldots, f_k$ (maybe negated) appear. Note that the disjunctive form of a Boolean expression is unique (up to a permutation of conjuncts). Then

$$re(t) = re(t@v_1) + \ldots + re(t@v_q) \tag{2.5}$$

Example 5 Assume $t@\{(10-1),(110)\} \in r$. Let the reliability of information sources be 60%, 80%, and 90%, respectively. We calculate the reliability of t using the two algorithms. Using Algorithm 1:

$K_1 = re(t@(10-1) + re(t@(110) = 0.06 + 0.48 = 0.54$

$K_2 = re(t@(10-1) \wedge (110) = re(t@(11-1)) = 0.048$

$re(t) = K_1 - K_2 = 0.492$

Using Algorithm 2, first we convert $e(t)$ to disjunctive normal form:

$$e(t) = (f_1 \wedge \neg f_3) \vee (f_1 \wedge f_2) = (f_1 \wedge f_2 \wedge f_3) \vee (f_1 \wedge f_2 \wedge \neg f_3) \vee (f_1 \wedge \neg f_2 \wedge \neg f_3)$$

obtaining $t@(\{(111),(11-1),(1-1-1)\})$. Now

$$re(t) = re(t@(111)) + re(t@(11-1)) + re(t@(1-1-1))$$

$$re(t) = 0.432 + 0.048 + 0.012 = 0.492$$

2.5.6 Semantics for IST

In this section, we will introduce the semantic of the IST model introduced by Sadri in [19]. As discussed in [19], an extended relation represents a set of regular relations. Once we identify this set, we can process a query on the extended relations by processing it against the (regular) relations represented by the extended relations. Of course, this approach is not computationally possible in general. What we need is a query processing methodology that is applied to extended relations directly, is efficient, and generates the same answers that would be obtained by processing the query against represented (regular) relations. A query processing methodology that satisfies the latter condition is called "precise."

In what follows, the formal definition of the set of (regular) relations that represents an extended relation is quickly reviewed. Given a relation r on the scheme R, $r = \{t@x_1, \ldots, t@x_n\}$, where $x_1, \ldots, x_n$ are sets of source vectors, we define r^* as a function from the set of subsets of information sources $S = \{s_1, \ldots, s_k\}$ to be the set of (regular) relations *Rel* on the schemes $R - \{I\}$, that is,

$$r^* : 2^S \rightarrow Rel$$

Let $Q \subseteq S$ be a set of information sources. Assign truth values "true" to sources in Q, and 'false" to other sources. (We will denote this truth assignment by *truth(Q)* .) Then,

$$r^*(Q) = \{t_i \mid e(t_i) = true \text{ under truth(Q), }\}$$

where $e(t_i)$ is the expression corresponding to t_i in r.

Definition 25 An extended relation r *represents* the function r^*. We write this as $rep(r) = r^*$. The set of (regular) relations $r^*(Q)$, $Q \subseteq S$, is called the *alternate world* of r.

Informally, an extended relation r represents the set of (regular) relations consisting of those tuples that would be valid if the sources in Q were correct and all other sources were incorrect, for all $Q \subseteq S$.

Example: Consider the extended relation r_1 that represents the information gathered by sources s_1, s_2, and s_3 on students.

Relation r_1

Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}
John	St. Paul	10/07/1956	100
Sally	St. Paul	02/06/1954	001
Mary	St. Jack	01/01/1973	100
George	St. Dennis	02/03/1975	100
Peter	St. Paul	08/08/1975	101

We would like to find the alternate world representation of r_1. We will consider all subsets (Q) of { s_1, s_2, s_3 } and find their corresponding regular relations. Given 2^S = { {},{ s_1},{s_2},{s_3},{s_1, s_2},{s_1, s_3}, {s_2, s_3},{s_1, s_2, s_3} }, then the alternate world representation of r_1 consists of the empty regular relation corresponding to the empty set Q and the following:

$rep(\{s_1\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975

$rep(\{s_2\})$ is empty.

$rep(\{s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
Sally	St. Paul	02/06/1954

$rep(\{s_1, s_2\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975

$rep(\{s_1, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Sally	St. Paul	02/06/1954
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975
Peter	St. Paul	08/08/1975

$rep(\{s_2, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
Sally	St. Paul	02/06/1954

$rep(\{s_1, s_2, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Sally	St. Paul	02/06/1954
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975
Peter	St. Paul	08/08/1975

Definition: Let r be an extended relation, and Θ be the unary relational algebra operator. An extended operator Θ' is precise if

$$rep(\Theta'(r)) = \Theta(rep(r))$$

for all extended relations r, where $\Theta(rep(r))$ represents a function f such that $f(Q) = \Theta(r^*(Q))\ \forall Q \subseteq S$.

The graphical representation of this definition is depicted as follows:

$$\begin{tikzcd} r \arrow[r] \arrow[d] & \Theta'(r) \arrow[d] \\ rep(r) \arrow[r] & rep(\Theta'(r)) = \Theta(rep(r)) \end{tikzcd}$$

Definition: Let r_1 and r_2 be extended relations, and let Θ be a binary relational algebra operator. An extended operator Θ' is precise if

$$rep(r_1\Theta' r_2) = rep(r_1)\Theta rep(r_2)$$

for all extended relations r_1 and r_2, where $rep(r_1)\Theta rep(r_2)$ represents a function f such that $f(Q) = r_1{}^*(Q)\Theta r_2{}^*(Q)\ \forall Q \subseteq S$.

The graphical representation of this definition is depicted as follows:

$$\begin{tikzcd} r \arrow[d] & s \arrow[r] \arrow[d] & (r)\ \Theta'\ (s) \arrow[d] \\ rep(r) & rep(s) \arrow[r] & rep((r)\ \Theta'(s)) = (rep(r)\Theta(rep(s)) \end{tikzcd}$$

Correctness of extended relational algebra operations were proven by Sadri in [19]. We will list the result from [19].

Definition 26 The extended relational algebra operations selection (σ), Projection (Π), Union ($\cup$), Intersection ($\cap$), Cartesian Product ($\times$), and Set Difference ($-$) are precise.

2.5.7 Probabilistic Approach

In section 2.5.5 we had concentrated on the problem of determining information sources that contributed to an answer to a query. It is often possible to attach a quantitative *reliability* measure to each information source. In this section we will justify the correctness of these algorithms using the alternate worlds semantics.

Definition 27 The reliability of correctness of an information source s_i is called its *reliability* , and is denoted by P_i.

Recall that the information content of an extended relation r is a function r^*, where, for a given $Q \subseteq S$,

$r^*(Q) = \{t \mid t@x \in r$, and $e(t@x) = true$ under truth(Q) $\}$

With each $Q \subseteq S$, and with each $r^*(Q)$, we associate a probability $P(Q)$ as follows:

$$P(Q) = \prod_{s_i \in Q} P_i \prod_{s_i \in S-Q} (1 - P_i) \tag{2.6}$$

In this way, we have associated with each (regular) relation $r^*(Q)$ in the alternate world set or r a possibility $P(Q)$ which is the probability that r represents $r^*(Q)$ (when the sources in Q are true and all the other sources are not true). In other words, not only we know r represents $r^*(Q)$; but we also have a quantitative measure of the likelihood that r represents $r^*(Q)$.

Example 6 Consider the following extended relation. Assume the reliabilities of the three sources are as follows: $p_1 = 65\%, P_2 = 70\%$, and $P_3 = 90\%$. Then, the probabilities attached to the regular relations in the alternate world of the given extended relation is, corresponding to the set $2^S = \{\emptyset, \{s_1\}, \{s_2\}, \{s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, \{s_1, s_2, s_3\}\}$, are 0.0105%, 0.0195%, 0.0245%, 0.0945%, 0.0455%, 0.1755%, 0.2205%, 0.4095%, respectively.

Given an extended relation r (e.g. representing the answer to a query), and pure tuple t such that $t@x \in r$ for the set of source vectors $x = \{u_1, \ldots, u_p\}$, we would like to calculate the *reliability* of t. we define the reliability of t to be the degree to which r represents t. This, in turn, can be equated to the probability that t is present in the alternate world of r. We will make this notion precise below.

Definition 28 The *reliability* of a pure tuple t, ,represented by an extended relation r (i.e. $t@x \in r$ for a set of source vectors $x = \{u_1, \ldots, u_p\}$), is

$$re(t) = \sum_{t \in r^*(Q)} P(Q) \tag{2.7}$$

First, let us assume that $x = \{u\}$, e.g. t has a single source vector associated with it in r. Let $u = (a_1, \ldots, a_k)$. We can partition the set of information sources I into three sets: $S^+(u), S^-(u)$, and $S^0(u)$ as follows:

$S^+(u) = \{s_i \mid a_i = 1\}$, $S^-(u) = \{s_i \mid a_i = -1\}$, and $S^0(u) = \{s_i \mid a_i = 0\}$.

Intuitively, $S^+(u)$ is the set of information sources contributing positively to a tuple t; $S^-(u)$ is the set of information sources contributing negatively to t; and $S^0(u)$ is the set of information sources not contributing to t.

Lemma 4 If $t@u \in r$, then the pure tuple t appears in (regular) relations $r^*(Q)$ from the alternate world of r if and only if $S^+(u) \subseteq Q \subseteq S^+(u) \cup S^0(u)$. *Proof:* Recall that

$$e(t@u) = \bigwedge_{s_i \in S^+(u)} f \bigwedge_{s_i \in S^-(u)} \neg f$$

Obviously $e(t@u) = true$ under truth(Q) if and only if $S^+(u) \subseteq Q \subseteq S^+(u) \cup S^0(u)$.

Theorem 11 If $t@u \in r$ ($t@u$ is the only extended tuple in r corresponding to the pure tuple t), then

$$re(t) = \prod_{s_i \in S^+(u)} P_i \prod_{s_i \in S^-(u)} (1 - P_i) \tag{2.8}$$

where $re(t)$ is the reliability of t.

Proof: By the definition of the reliability of a tuple (Equation 2.7), and lemma 4, we have:

$$re(t) = \sum_{S^+(u) \subseteq Q \subseteq S^+(u) \cup S^0(u)} P(Q) \tag{2.9}$$

If we replace $P(Q)$ by its expression from the previous equation and reduce the above assumption, we obtain:

$$re(t) = \prod_{s_i \in S^+} P_i \prod_{s_i \in S^-} (1 - P_i)$$

Let us, now, concentrate on a more general case, where $t@u \in r$ has a set of source vectors associated with it. Reliability calculation algorithm were presented in section 2.5.5 to calculate the reliability of t. We will use the following definition:

$$re(t) = \sum_{t \in r^*(Q)} P(Q)$$

which is based on the alternate world model, to justify these algorithms.

Lemma 5 Assume $t@x \in r, x = \{u_1, \ldots, u_p\}$. Then $t \in r^*(Q)$ if and only if $S^+(u_i) \subseteq Q \subseteq S^+(u_i)$, for at least one $i, 1 \leq i \leq p$.

Proof: Obvious from lemma 4, and by the fact that

$$e(t@x) = \bigvee_{1=1}^{p} e(t@u_i)$$

Now we can derive an equation, similar to Equation 2.9 for the reliability of a tuple $t@x$, where x is the set of source vectors associated with t. Let

$$\mathcal{Q} = \{Q_i \mid S^+(u_i) \subseteq Q_i \subseteq S^+(u_i) \cup S^0(u_i), i = 1, \ldots, p\} \tag{2.10}$$

then

$$re(t) = \sum_{Q \in \mathcal{Q}} P(Q) \tag{2.11}$$

where $re(t@x)$ is the reliability of t when $t@x \in r$, and $re(t@u_i)$ is the reliability of t when *only* $t@u_i \in r$. This is because some of $P(Q)$'s in Equation 2.11 may be added more than once when we sum up $re(t@u_i)$'s. This observation led to the reliability calculation algorithm 1 and the following theorem.

Lemma 6 Let $\mathcal{Q} = \{Q \mid S^+(u_i) \subseteq Q_i \subseteq S^+(u_i) \cup S^0(u_i)\}$, $i = 1, 2$. Then

$$\sum_{Q \in \mathcal{Q}_1 \cup \mathcal{Q}_2} P(Q) = \sum_{Q \in \mathcal{Q}_1} P(Q) + \sum_{Q \in \mathcal{Q}_2} P(Q) - \sum_{Q \in \mathcal{Q}_1 \cap \mathcal{Q}_2} P(Q)$$

Proof: This lemma is based on the principle of inclusion and exclusion [14]. Some of the Q's may be repeated in the first and second summation on the right hand side, which are deduced in the third summation on the right hand side.

This lemma can be generalized to more than two sets. The generalization will alternate adding and subtracting the effect of elements appearing in only one set, in two sets, in three sets, etc ...

In [19], Sadri proved the reliability calculation algorithms are correct. We list these results below.

Theorem 12 Algorithm 1 correctly computes the reliability of a pure tuple t according to Equation 2.7.

Theorem 13 Algorithm 2 correctly computes the reliability of a tuple t according to Equation 2.7.

2.5.8 Summary

In the IST model, information that contributed to data are recorded along with data. The accuracy of data is represented by the reliability of the contributing source(s). The query processing algorithms manipulate data, as well as contributing sources of information. For each answer, information sources that contributed to the answer, and their nature of contribution are identified. A measure of validity (certainty) can be calculated for the answer as a function of the reliability of information sources.

The approach taken in IST differs from most other approaches to uncertainty management, in which rules and data are associated with "certainty" measures directly. IST provides a clean and effective way to model dependent and independent data, an issue which creates substantial difficulties with other techniques. The calculation of certainties to answers to users queries (or validities) are left as an (optional) last step in query processing.

2.6 IST-based Deductive Approach

Lakshmanan and Sadri [11] extended the IST method [18, 19] to deductive databases. They show that the *positive uncertain database*, i.e., IST-based deductive database with positive literals in the head and bodies of rules, has a least fixed point semantics. Query processing in this framework for answering queries was studied. The top-down and the bottom-up query processing techniques of logic programming and deductive databases are extended in their approach to provide reliabilities to query answers. Negation was studied for uncertain databases, concentrating on *stratified uncertain databases.*

In this approach, IST-based deductive database associates certainty, showing the contributing information sources, with the information stored in the deductive database. In other words, the information is supplied or confirmed by the information sources, and the reliability of the contributing information sources determines the certainty of the information stored in the deductive database. Query processing algorithms handle data as well as the information sources associated with the data to keep track of which source contributed to which data. An answer to a query also identifies sources contributing to the answer as well as their nature of contribution. Based on the nature of contribution, the reliability of data is calculated as an optional last step of query processing. Moreover, IST-based deductive database provides a clean and effective framework to deal with the issues of dependent and independent data.

2.6.1 Uncertain Database

The first order language for the IST-based deductive database is used. A deductive database, in this approach, consists of a set of annotated rules. The annotation of a rule identifies the information sources contributing to the rules and their nature of contribution. The facts, i.e. raw database information, are special cases of the rules. We will make these notions precise in the following definitions.

Variables, constants, and predicate symbols, as well as the set of *information source constants*, $\{0, -1, +1, \top\}$ constitute the alphabet of the language. As in the classical case, a term is a variable or a constant; a literal is of the form $p(x_1, \ldots, x_n)$, or $\neg p(x_1, \ldots, x_n)$, where p is an n-ary predicate symbol and $x_1, \ldots, x_n$ are terms. The information source constants are used to form vectors to annotate logical rules. In other words, a set of source vectors is associated with the head of a rule showing the contributing information to that rule. A rule r is defined as follows:

$$(p \leftarrow q_1, \ldots, q_n)@x, \text{ where}$$

p is a positive literal, $q_1, \ldots, q_n$ are literals, and x is a set of information source vectors. p is called the head of the rule and $q_1, \ldots, q_n$ are called the body of the rule r. As usual to produce a finite set of answers to a query, we require the rule to be *range restricted*, that is, all the variables appearing in the head of the rule must appear in the body of the rule. An uncertain *IST-based deductive database (uncertain database,* for short) is a set of such annotated rules.

As a special case, a *fact* is a rule with an empty body, and no variables in the head. We usually write a fact as follows:

$$(p \leftarrow) \text{ or } p@x, \text{ where}$$

x is a set of source vectors associated with the fact p.

As in the regular deductive database system, the positive uncertain database is divided into the *extentional* and the *intentional* predicates, *EDB* and *IDB*, where the *EDB* and the *IDB* predicates are disjoined.

2.6.2 Minimal Model Fixed Point

Let D be an uncertain database. The set of ground atoms obtained using the symbols of D is the *pure Herbrand base* H_B of an uncertain database D. A *Herbrand* interpretation I of D is defined as:

$$\{p@v \mid p \in H_B \text{ and } v \text{ is an information source vector}\}$$

A ground instance of a rule $(p \leftarrow q_1, \ldots, q_n)@x$ is satisfied by an interpretation I of an uncertain database D if for all $q_i@v_i \in I$ for $i \in \{1, \ldots, n\}$, then we also have $p@v \in I$, for all $v \in (x \wedge v_1 \wedge \ldots \wedge v_n)$. An interpretation I of a positive uncertain database D satisfies a rule r if it satisfies all its ground instances. An interpretation I of an uncertain database D satisfies a *model* of D if it satisfies all the rules in D.

A *Herbrand model* of a positive uncertain database D is a Herbrand interpretation of D that is also a model of D.

The following result was proven by Lakshmanan and Sadri in [11]:

Lemma 7 Let M_1 and M_2 be two Herbrand models of an uncertain database D, then $M = M_1 \cap M_2$ is also a Herbrand model of D.

It follows from the above lemma that the intersection of all Herbrand models of a positive uncertain database D is a model of D, and it is the least fixed model of D. The result of the Lemma 7 and Theorem 14 was proven in [11].

Theorem 14 Let D be a positive uncertain database. Then D has a least model M_D that is equal to the intersection of all models of D.

The least model of a positive uncertain database D characterizes the information content of D. In fact, we can regard the least model of an uncertain database as a collection of relations (instances), as in the relational model, and evaluate queries against this relational interpretation. The characterization of the least model as the intersection of the models does not provide an effective algorithm to obtain the least model. The *immediate consequence operator*, T_D, is needed to build up the *least model* of D which should be equal to the intersection of all models of D.

2.6.3 Immediate Consequence Operator T_D and its Fixed-point

As in a regular deductive database system, we need an operator so that when applied to the EDB and IDB of the uncertain database the intended model will be calculated. Lakshmanan and Sadri in [11] proved that the intended model and the least fixed point model of the positive uncertain database are equal.

For a given positive uncertain database D, the operator T_D maps interpretations of D into interpretations of D. That is:

$$T_D : \mathcal{I} \rightarrow \mathcal{I},$$

where $\mathcal{I}$ is the set of all interpretations of D. Let $I \in \mathcal{I}$, $T_D(I)$ is defined as:

$T_D(I) = \{p@v \mid (p \leftarrow (q_1, \ldots, q_n)@x$ is a ground instance of a rule in D, where $q_i@v_i \in I$ and $v \in (x \wedge v_1 \wedge \ldots \wedge v_n)\}$.

Lakshmanan and Sadri proved the following results in [11].

Lemma 8 An interpretation I of a positive uncertain database D is a model of D if and only if it is a pre-fixpoint of T_D, that is $T_D(I) \subseteq I$.

Lemma 9 The operator T_D is monotonic.

Theorem 15 If D be a positive uncertain database, then the least fixedpoint of T_D is the least model of D.

Example 7 Consider the a positive uncertain database D with the following set of facts and rules.

F_1 : edge(a,b) @ (10000)
F_2 : edge(b,c) @ (01000)
F_3 : edge(c,d) @ (00100)
R_1 : (path(X,Y) $\leftarrow$ edge(X,Y)) @ (00010)
R_2 : (path(X,Y) $\leftarrow$ edge(X,Z), path(Z,Y)) @ (00001)

The first iteration of the T_D operator yields the following:

$$edge(a,b)@(10000),\ edge(b,c)@(01000),\ edge(c,d)@(00100)$$

The second iteration, in addition to results of the first iteration, yields the following:

$$path(a,b)@(10010),\ path(b,c)@(01010),\ path(c,d)@(00110)$$

The third iteration, in addition to results of second the iteration, yields the following:

$$path(a,c)@(11011),\ path(b,d)@(01111)$$

The fourth iteration, in addition to results of the third iteration, yields the following:

$path(a,d)@(11111)$

The fifth iteration, nothing new is generated:

As a result, the *least fixpoint* of T_D for this example, which is the least model of the positive uncertain database D, is shown below:

$$edge(a,b)@(10000),\ edge(b,c)@(01000),\ edge(c,d)@(00100),$$

$$path(a,b)@(10010),\ path(b,c)@(01010),\ path(c,d)@(00110),$$

$$path(a,c)@(11011),\ path(b,d)@(01111),\ path(a,d)@(11111).$$

2.6.4 Top-Down and Bottom-Up Evaluation

Using the T_D operator to derive the minimal model of a database and evaluate the answers of a query using the derived minimal model is inefficient. What is really needed is a method that identifies the relevant part of the database to produce all answers to a query. In an approach based on logic in a regular deductive database, a query is evaluated by using logical rewriting and a logical evaluation method in sequence. This entails the notion of unification, resolution and so on. In contrast, in an approach based on relational algebra operations a query is solved by using an algebraic evaluation method in sequence. In this case, operations are expressed using the regular relational algebra operations. In the work done by Lakshmanan and Sadri [11], these approaches were extended taking into consideration the manipulation of the source vectors associated with the EDB and the IDB of the positive uncertain database.

Top-Down and *Bottom-Up* evaluation techniques are two methods that have been extensively explored for logic programming and query processing in deductive databases to yield answers to users queries. In their approach, Lakshmanan and Sadri [11] extended the top-down and the bottom-up evaluation techniques to take into consideration the manipulation of source vectors. In such an approach, the source vectors associated with answers to queries identifies the information sources that contributed to each ground instance, as well as the nature of their contribution.

A top-down approach query evaluation technique (such as the *SLD-resolution*) starts from the query and derives answers by applying the rules using backward chaining [10]. In other words, The top-down technique is based on theorem proving (SLD-resolution). Such techniques benefits from the structure of the query (goal) to

[10]the backward chaining concept corresponds to the top-down technique.

reduce the search space. However, these techniques produce one ground instance at a time and, for each instance, all corresponding source vectors that contribute to that particular instance.

A bottom-up approach query evaluation technique can yield efficient query evaluation benefiting from certain rule rewriting techniques, such as magic sets and magic templates to produce a *set* of answers at a time for each query. In general, a bottom-up evaluation technique will produce a large number of facts that is not needed in generating query answers. To reduce the large number of irrelevant facts during the bottom-up query evaluation technique, extra rules and predicates are added to the database. The extra rules and predicates are added using the magic sets and templates methods. Each new rule added to the positive uncertain database is associated with a T (*true*) source vector. The source vectors of the modified rules remains unchanged.

Using the extended methods top-down or bottom-up, answers to queries are annotated with sets of source vectors. At this point, the reliability of answer to queries is calculated using the reliability calculation algorithms introduced by sadri [18, 19] provided that the sources are independent and have predefined reliabilities. In this sense the accuracy (reliability) of answers depends on the reliability of the contributing information sources as well as on the nature of their contribution. In what follows we will give an example on both approaches adopted from [11].

Example 8 Consider the uncertain database of example 7.
Let $S = \{s_1, s_2, s_3, s_4, s_5\}$ be the set of information sources in D. Let 90%, 65%, 75%, 80%, and 85% be the reliability of the information sources s_1, s_2, s_3, s_4, and s_5 respectively.

F_1 : edge(a,b) @ (10000)

F_2 : edge(b,c) @ (01000)

F_3 : edge(c,d) @ (00100)

R_1 : (path(X,Y) $\leftarrow$ edge(X,Y)) @ (00010)

R_2 : (path(X,Y) $\leftarrow$ edge(X,Z), path(Z,Y)) @ (00001)

Let the $\leftarrow path(a,d)$ be the goal (query). We will use the extended Top-down evaluation method that Lakshmanan and Sadri introduced in [11] to evaluate this query.

Unifying path(a,b) with the head of rule R_2:

$\leftarrow edge(a,Z), path(Z,d)@(00001)$

Unifying $edge(a,Z)$ with fact F_1:

$\leftarrow path(b,d)@(10001)$

Unifying $path(b,d)$ with the head of rule R_2:

$\leftarrow edge(b,Z), path(Z,d)@(10001)$

Unifying $edge(b,Z)$ with fact F_2:

$\leftarrow path(c,d)@(11001)$

Unifying $path(c,d)$ with the head of rule R_1:

$\leftarrow edge(c,d)@(11011)$

Unifying $edge(c,d)$ with fact F_3 ($\square$ denotes the empty clause):

$\square@(11111)$

Using the reliability calculation algorithms introduced by Sadri in [18, 19] we deduce that the reliability for the goal (query) is calculated as follows:

$rel(path(a,d)) = 0.9 \times 0.65 \times \times 0.75 \times 0.80 \times 0.85 = 0.29835$

The reliability factor associated with the answer to this query is amount of certainty associated with the fact $path(a,d)$ and this certainty depends on the sources s_1, s_2, s_3, s_4, and s_5. In other words, we say that the is a probability of 29.835% that there is a path from a to d and the sources who contributed to this information are s_1, s_2, s_3, s_4, and s_5, where they all confirm this fact and no one denies.

Example 9 Consider the following rules in an uncertain database. The magic set approach will be used to demonstrate how answers to queries could be generated together their corresponding source vectors.

$$person(b)@w_1$$
$$par(a,b)@w_2$$
$$par(c,b)@w_3$$

$$(sg(X,X) \leftarrow person(X))@u$$
$$(sg(X,Y) \leftarrow par(X,X_p), sg(X_p,Y_p), par(Y,Y_p))@v$$

Assume $s(a,W)$ is the query to be answered, where "a" is a constant. The Magic sets algorithm generates the fact $m\_s(a)@T$, and the following rules:

$$(m\_s(X_p) \leftarrow sup_{2.1}(X,X_p))@T$$
$$(sup_{1.0}(X) \leftarrow m\_s(X))@T$$
$$(sup_{2.0}(X) \leftarrow m\_s(X))@T$$
$$(sup_{2.1}(X,X_p) \leftarrow sup_{2.0}(X), par(X,X_p))@T$$
$$(sup_{2.2}(X,X_p) \leftarrow sup_{2.1}(X,X_p), sg(X_p,Y_p))@T$$
$$(sg(X,X) \leftarrow sup_{1.0}(X), person(X))@u$$
$$(sg(X,Y) \leftarrow sup_{2.2}(X,Y_p), par(Y,Y_p))@v$$

Both the original database and the rewritten database generates the following answer:

$$sg(a,c)@(u \wedge v \wedge w_1 \wedge w_2 \wedge w_3)$$

2.6.5 Incorporating Negation

In recent years, representing negation in logic programming and deductive databases have been studied and it is still an active area of research. In general, the problem with such a deductive database, which allows negative as well as positive literals to appear in the heads and bodies of rules, is that it does not have a unique minimal model. Uniqueness of the minimal model is significant because the minimal model represents the semantic interpretation of the deductive database. The *perfect model*, the *well founded model*, and the *stable model* were introduced in [1, 7, 25] for general logic programs and deductive databases as the semantic interpretation to handle negation in such databases.

Lakshmanan and Sadri [11] extended the concept of *stratified databases* to handle uncertainty in the presence of negative as well as positive literals in the heads and bodies of rules provided that the database remains stratified.

2.6.6 Stratified Uncertain Databases

A deductive database D, which allows negative and positive literals in the body of its rules, is *stratified* provided that the predicates of D can be partitioned into sets $P_1, \ldots, P_k$ such that if given the following rule:

$$r : p \leftarrow q_1, \ldots, q_n, \neg q_i, \ldots, q_k,$$

where $p \in P_i$, then $q \in P_j$, for some $j < i$.

P_i $(\forall i \in \{1, \ldots, k\})$ is called a *strata* of D. Although stratified databases can have more than one minimal model, there is a unique minimal model among them that could be obtained by computing the predicates in the order of the strata, starting from the strata P_1 which have the lowest order.

Lakshmanan and Sadri in [11] extended the stratified databases to *stratified uncertain databases.* The stratified uncertain database is defined as an IST-based deductive database where the rules, without sources vectors, form a stratified program. The source vector negation is needed for the manipulation of source vectors for negative literals. For example for a rule of the form:

$$(p \leftarrow q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n)@x, \text{ where}$$

is the stratified uncertain database D, if $q_i@u_i$ $(\forall i = 1, \ldots, n$ and $r_j@v_j, (\forall j = 1, \ldots, m)$ are already obtained, then

$$p@(u_1 \wedge \ldots \wedge u_m \wedge (\neg v_1) \ldots \wedge (\neg v_n) \wedge x)$$

can be derived.

2.6.7 Summary

In the Information Source Tracking Method (IST), introduced by Sadri [18, 19], accuracy (certainty) of data is modeled by the reliability of information source(s) contributing to the data. IST-based deductive databases, introduced by Lakshmanan and Sadri [11] extended the IST concepts to deductive databases where they studied query processing techniques for the new model. SLD-resolution, a top-down evaluation technique used in logic programming, is extended to handle the manipulation of the source vectors associated with the facts and rules of the positive uncertain database. Magic sets method, a rule-rewriting technique used in conjunction with bottom-up query evaluation in deductive databases, is extended to manipulate the source vectors associated with the rules and facts of the positive uncertain database. Both approaches, in [11] were proven to be correct. Stratified databases were extended to handle negation in an approach similar to that in the regular deductive databases.

IST and IST-based deductive database are capable of modeling probabilistic data interdependencies, an issue that created difficulties in other models. In addition, both models calculate the reliabilities of answers to queries as the last step in the query processing algorithms of each model. In this case, the reliability of the contributing sources determines the accuracy of the answers to queries.

Chapter 3

The Model and The Extended Relational Algebra Operations

A formal approach to the Extended Relational Model and the Extended Relational Algebra Operations.

3.1 Our New Model

We will assume that an extended relational model will have k sources. In other words, k is the number of sources (observers) providing us with the information to be stored in the extended relations of our model. Any source or observer can contribute to any tuple or attribute value provided. Based on this intuition, we will associate an additional attribute whose values (source vectors) will be associated with each tuple in the extended relation. The value (source vector) of the additional attribute indicates the contributing sources and the condition under which the tuple exists in the extended relation. Moreover, with each attribute in the extended relation we associate a source attribute whose value, a source vector, is associated with the value of that attribute to indicate the information sources that contributed to that attribute as well as the information sources that contributed to the tuple or to the condition under which the tuple exists in the extended relation.

Consider, for example, an information source s_i that confirms the existence of a tuple t in an extended relation. Then the i^{th} element of the source vector associated with t is set to 1, and the remaining elements of the source vector are set to 0 indicating that the other sources did not participate (confirm). Having this concept in mind, we say that the source s_i contributed positively to the condition under which tuple t exists in the extended relation r. Similarly, for an information source s_j that confirms the information value for an attribute A_l of tuple t, the i^{th} and the j^{th} elements of the source vector, which is associated with the information value of attribute A_l, are set to 1 and the remaining elements of the source vector are set to 0. Setting the i^{th} and the j^{th} elements of the source vector associated with the attribute A_l of tuple t in r indicates that the value for attribute A_l is valid whenever the source s_i, who contributed to the condition under which the tuple t exists in r, is reliable and the source s_j, who contributed to the value of attribute A_l, is reliable. In other words, the validity of an attribute value of a tuple t in r depends on the source(s) contributing to the tuple t and the sources contributing to the attribute value. The source vectors that are associated with every attribute value of a tuple in the extended relational model will usually have 1 and 0 elements, showing the contributed and the non-contributed information sources respectively. Some attribute values, in answers to queries, might have source vectors with $-1, 0, 1$ elements or $\top$. A source vector with a -1 entry at position i indicates that the corresponding attribute is valid if the information source s_i is not reliable (correct). A source vector with a $\top$ entry at position i indicates that the corresponding attribute is not consistent with respect to source s_i. We allow special source vectors T (True) and F (False) to indicate that the information they are associated with is correct and incorrect respectively. The same concept will hold for the source vectors associated with the pure tuples of an extended relation. In the coming sections we will explicitly define these concepts.

An extended relation scheme $R = \{A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n}\}$ is a set of regular attributes $A_1, \ldots A_n$ and special attributes $S_{A_1}, \ldots, S_{A_n}$ called the source attributes. The special attributes $S_{A_1}, \ldots, S_{A_n}$ are associated with $A_1, \ldots, A_n$ respectively. Values for source attributes are defined to be source vectors, where every source vector has a fixed length k (k is the number of contributing sources.) Moreover, S_{A_0} is the source attribute whose values, source vectors, are associated with the pure tuples in the extended relation. A value (source vector) of S_{A_0} indicates the condition under which the tuple exists in the extended relation r. A schema for an extended relational database is a set of extended relational schemas such that the domain of source attributes remains the same in all extended relations.

An extended relation instance r is a finite subset of $D_1 \times \ldots \times D_n \times D_{S_{A_0}} \times D_{S_{A_1}} \times \ldots \times D_{S_{A_n}}$, where $D_1, \ldots, D_n$ are the domains of the attributes $A_1, \ldots, A_n$ respectively, and $D_{S_{A_0}}, D_{S_{A_1}}, \ldots, D_{S_{A_n}}$ are the domains of $S_{A_0}, S_{A_1}, \ldots, S_{A_n}$ respectively. We say that any tuple t in r is made up of values for the regular attributes $A_1, \ldots, A_n$ and values, as source vectors, for the source attributes $S_{A_0}, S_{A_1}, \ldots, S_{A_n}$. For example, consider the following extended relational scheme, $R = \{A_1, A_2, A_3, S_{A_0}, S_{A_1}, S_{A_2}, S_{A_3}\}$, where k (number of source vectors) is 4.

A_1	A_2	A_3	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
a_1	a_2	a_3	1001	1101	1011	1111

We will denote an extended tuple t' in the extended relation r by $t'@\mathcal{U} \in r$, where t is a *pure* tuple, $t@\mathcal{U}$ i.e., t is the restriction of t' to the regular attributes, and $\mathcal{U} = \{U_1, \ldots, U_m\}$ is associated with the pure tuple t, and m ($m \geq 1$) is the number of times the pure tuple t appears in the extended relation r. We define $U = (u_0, u_1, \ldots, u_n)$, where $U \in \mathcal{U}$; u_0 is the source vector associated with the pure tuple t in the extended relation r. Moreover, we define $(\forall i)$ $(1 \leq i \leq n)$ u_i as the source vector associated with the value of attribute A_i. The expression of u_i, for some $U \in \mathcal{U}$, shows the information sources as well as their nature of contribution to the attribute value for A_i of tuple t.

In the above example, there are four source vectors (s_1, s_2, s_3, s_4) $(k = 4)$ and a single pure tuple $t = (a_1, a_2, a_3)$ such that $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U\}$ and $U = (u_0, u_1, u_2, u_3)$. The source vector $u_0 = (1001)$ is associated with the pure tuple t, where sources s_1 and s_4 contribute to the condition under which t exists in the extended relation r. The source vector $u_1 = (1101)$ is associated with attribute value $A_1 = a_1$ confirmed by sources s_1, s_2, and s_4 of tuple t in r. The source vector $u_2 = (1011)$ is associated with attribute value $A_2 = a_2$ confirmed by sources s_1, s_3, and s_4 of tuple t in r. Similarly, The source vector $u_3 = (1111)$ is associated with attribute value $A_3 = a_3$ confirmed by sources s_1, s_2, s_3, and s_4 of tuple t in r. Notice that the sources s_1 and s_4, that contributed to the tuple (a_1, a_2, a_3), appear among the sources that contributed to the attribute values a_1, a_2, and a_3.

3.2 Information source vectors

Consider an extended relation r, whose extended scheme is $R = \{A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n}\}$. Let $t@\mathcal{U} \in r$, where $\mathcal{U}$ is associated with the pure tuple t in r. If $\mathcal{U} = \{U_1, \ldots, U_p\}$ and for any $U \in \mathcal{U}$, where $U = (u_0, \ldots, u_n)$ we define the intended meaning of the source vector u_i as follows:

The source constants, $\{0, +1, -1, \top\}$, are used to specify the meaning of the contribution made by the information sources to the pure tuples and to the attributes of the same tuple. In general, for a source vector $u = (a_1, a_2, \ldots, a_k)$, the value of a_i specifies the meaning of contribution of the information source s_i as follows:

$a_i = 0$ means that the source s_i neither confirms nor denies

$a_i = -1$ means that the source s_i denies

$a_i = +1$ means that the source s_i confirms

$a_i = \top$ means that the source s_i is inconsistent

The constants 0 and $\top$ denote under specification and over specification, respectively. The case of $a_i = \top$ happens when an attribute value of a tuple or a tuple is confirmed and at the same time denied by the same source s_i.

As defined by Sadri [18, 19], a source vector $u = (a_1, a_2, .., a_k)$ specifies the conjunction of the meaning of each information source s_i as given by a_i. A set of source vectors $x = \{ u_1, \ldots, u_m \}$ specifies the disjunction of the specification of the vectors $u_1, \ldots, u_m$. The following definitions follow from section 2.5.1.

For a source vector u,

$$e(u) = \bigwedge_{s_i \in S^+} f_i \bigwedge_{s_j \in S^-} \neg f_j.$$

Note that $e(u) = \text{false}$ if $(\exists i)$ $(1 \leq i \leq k)$ such that $a_i = \top$.

For a set of source vectors x,

$$e(x) = \bigvee_{u \in x} e(u),$$

We aim to answer queries in a relational database systems where data is uncertain. The uncertainty of the data is modeled by the reliability of the information sources, that provided the information to the database, and captured by the meaning given to the contribution[1] of a source vector. The answer to a query reflects its uncertainty because of the appropriate accompanying set of information source vectors that are associated with the answers of the query. The query processing algorithm determines the answer to a query along with their respective source vectors and the associated reliability for each tuple in the answer to the query. The following example illustrates our approach.

Example 10 Let us consider an application for a certain Coast Guard database. For simplicity assume that the database designer is interested in the name and location of a ship. Let the database system admit four observers that are providing the information concerning the name, location, and type of a ship. Any observer (source) could contribute to any attribute value. In other words, the first observer in the first tuple in the extended relation r confirms that the location of "Atlantica" is the "Atlantic Ocean" and the type of this ship is "Commercial". A similar interpretation holds for the other sources of the same tuple.

Let $S = \{s_1, s_2, s_3, s_4\}$ be the set of source vectors that contributed to the information presented in the extended relation r whose extended scheme is $R = \{$*Ship-Name,Ship-Location, Ship-Type,*$S_{A_0}, S_{A_1}, S_{A_2}, S_{A_3}\}$, where

S_{A_0} is the source attribute whose values (source vectors) are associated with pure tuples in r. S_{A_1}, S_{A_2}, and S_{A_3} are the source vectors associated with *Ship-Name*, *Ship-Location*, and *Ship-Type* respectively.

Relation r						
Ship-Name	*Ship-Location*	*Ship-Type*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
Atlantica	Atlantic Ocean	Commercial	1001	1001	1101	1011
Black Bird	Atlantic Ocean	Commercial	0101	0111	1101	1111
Black Bird	Atlantic Ocean	Commercial	1010	1010	1110	1011

[1]confirms, denies, don't know, or inconsistent.

Let t_1 = (Atlantica, Atlantic Ocean, Commercial) and t_2 = (Black Bird, Atlantic Ocean, Commercial). Let A_1 = *Ship-Name*, A_2 = *Ship-Location*, and A_3 = *Ship-Type*.

We say $t_1@\mathcal{U}$ and $t_2@\mathcal{V} \in r$, where $\mathcal{U} = \{ U \}$ and $\mathcal{V} = \{ V, W \}$;

$U = (u_0, u_1, u_2, u_3)$, $V = (v_0, v_1, v_2, v_3)$, $W = (w_0, w_1, w_2, w_3)$.

$u_0 = 1001$, $u_1 = 1001$, $u_2 = 1101$, $u_3 = 1011$.

$v_0 = 0101$, $v_1 = 0111$, $v_2 = 1101$, $v_3 = 1111$.

$w_0 = 1010$, $w_1 = 1010$, $w_2 = 1110$, $w_3 = 1011$.

$e(u_0) = s_1 \wedge s_4$ is the expression of the condition under which pure tuple t_1 exists in r; and $e(\{v_0, w_0\}) = (s_2 \wedge s_4) \vee (s_1 \wedge s_3)$ is the expression of the condition under which tuple t_2 exists in r.

Source vector operations *conjunction*, *disjunction*, and *negation* and theorems 8, 9, 10 were given in section 2.5.2.

It should be clear that the expression of a source vector gives the meaning of contribution to information presented as a tuple or as an attribute. Theorems 8, 9, 10 are introduced to capture the meaning of new source vectors (z) after a $\wedge, \vee, and \neg$ are performed on other source vectors like x and y.

These operations are required to identify the information sources that contributed to the answers to queries.

3.3 Extended Relational Algebra Operations

Like the regular relational database system, we expect that the query processor accepts the query and uses the extended relations giving an extended relation as the answer to the query with the corresponding reliabilities. Calculating the reliabilities is the last step done by the query processor and is based on the algorithms given by Sadri [18, 19].

Let r_1 and r_2 be two extended relations whose schemes are R_1 and R_2, where $R_1 = \{ A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n} \}$ and $R_2 = \{ B_1, \ldots, B_m, S_{B_0}, S_{B_1}, \ldots, S_{B_m} \}$. Let $t_1@\mathcal{U} \in r_1$, where $\mathcal{U} = \{U_1, \ldots, U_p\}$, p is the number of times pure tuple t_1 appears in r_1, and $U = (u_0, u_1, \ldots, u_n)$ $(\forall U \in \mathcal{U})$. Similarly, let $t_2@\mathcal{V} \in r_2$, where

$\mathcal{V} = \{V_1, \ldots, V_q\}$, $V = (v_0, v_1, \ldots, v_m)$ $(\forall V \in \mathcal{V})$, and q is the number of times pure tuple t_2 appears in the extended relation r_2.

3.3.1 Extended Selection (σ')

Let r and r' be the extended relations defined by $r' = \sigma'_F(r)$, where F is a formula involving:

1. Operands that are constants or attributes $A_i \in \{ A_1, \ldots, A_n \}$.
2. Arithmetic comparison operators $<$, $\leq$, $>$, $\geq$, $=$, $\neq$.
3. The logical operators $\wedge$ (for AND), $\vee$ (for OR), and $\neg$ (for NOT).

Formulas are defined in the usual way:

1. $F = A_j \Theta c$ is a simple formula, where c is a constant and Θ is an arithmetic comparison operator,
2. $F = A_i \Theta A_j$ is a simple formula
3. If F_1 and F_2 are formulas, then ($F_1 \wedge F_2$) is a formula
4. If F_1 and F_2 are formulas, then ($F_1 \vee F_2$) is a formula
5. If F is a formula, then $\neg(F)$ is a formula

Assume that the formula F (selection condition) is converted to a disjunctive form, i.e., $F = F_1 \vee \ldots \vee F_{q'}$, where each F_i is a conjunction of simple formulas. If a pure tuple t satisfies F, it satisfies at least one F_i. Let $F_{t_1}, \ldots, F_{t_{p'}}$ be the conjuncts satisfied by t. Consider $t@U \in r$ for any $U \in \mathcal{U}$. We are interested in finding $U'_i = (u'_{i0}, u'_{i1}, \ldots, u'_{in})$ $(1 \leq i \leq p')$ associated with pure tuple t in r'. The value u'_{i0} needs to be determined for each selected tuple in r'_1. The source vector for u'_{i0} can be obtained, as a function of $u_0, u_1, \ldots, u_n$ using the conjuncts $F_{t_1}, \ldots, F_{t_{p'}}$ as follows:

Define, $\forall i \in \{1,..,p'\}$, $\mathcal{A}_{t_i} = \{A \mid$ attribute A appears in F_{t_i} $\}$, $S_{t_i} = \{u \mid$ source vector u is associated with $A \in \mathcal{A}_{t_i}$ $\}$. Let

$$\mathcal{C}(S_{t_i}) = \bigwedge_{u \in S_{t_i}} u.$$

Finally, $r' = \{t@U'_i \mid \forall i \in \{1,\ldots,p'\}, (U'_i = (u'_{i0}, u'_{i1},\ldots, u'_{in})$, where $u'_{i0} = u_0 \wedge (\mathcal{C}(S_{t_i}))$, $u'_{ij} = u'_{i0} \wedge u_j$ $1 \leq j \leq n$, and t satisfies $F_{t_i}) \}$.

Example 11 Consider the following selection: $r'_1 = \sigma'_F r_1$, where F is the formula (*Stud-Address* = "St. Paul" AND *Stud-DOB* > "01/01/1955") OR (*Stud-DOB* > "01/01/1970" AND *Stud-DOB* < "01/01/1980").

Relation r_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	10000	10101	11100	10010
Sally	St. Paul	02/06/1954	true	01100	11000	01101
Mary	St. Jack	01/01/1973	true	10110	11010	10010
George	St. Dennis	02/02/1978	11000	11011	11011	11001
Peter	St. Paul	08/08/1975	00011	10011	11011	01011

The resulting extended relation is represented as follows:

Relation r'_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	11110	11111	11110	11110
Mary	St. Jack	01/01/1973	10010	10110	11010	10010
George	St. Dennis	02/02/1978	11001	11011	11011	11001
Peter	St. Paul	08/08/1975	11011	11011	11011	11011
Peter	St. Paul	08/08/1975	01011	11011	11011	01011

The values (source vectors) for the source attribute S_{A_0} associated with the tuples of the extended relation r_1 and some values (source vectors) for the source attributes $S_{A_1}, S_{A_2}, S_{A_3}$, of some tuples in r'_1, associated with *Stud-Name*, *Stud-Address*, and *Stud-DOB* respectively, have new values in the extended relation r'_1. For example, the source vector (11110) associated with the first tuple t of r'_1 is obtained by the conjunction operation using the source vectors (10000) associated with the pure tuple t in r_1, (11100) associated with *Stud-Address* = "St. Paul", and (10010) associated with *Stud-DOB* = "10/07/1956". These attributes satisfy the first part of the selection condition *F* (*Stud-Address* = "St. Paul" AND *Stud-DOB* > "01/01/1955") presented in the given query. The source attributes S_{A_1}, S_{A_2} and S_{A_3} have new values in r'_1. (11111) = (10101) $\wedge$ (11110), where (10101) is the source vector associated with *Stud-Name* = "John" of the first tuple in r_1 and (11110) is the source vector associated with the tuple t in r'_1; (11110) = (11100) $\wedge$ (11110), where (11100) is the source vector associated with *Stud-Address* = "St. Paul" of the first tuple in r_1 and (11110) is the source vector associated with the tuple t in r'_1; moreover, (11110) = (10010) $\wedge$ (11110), where (10010) is the source vector associated with *Stud-DOB* = "10/07/1956" of the first tuple in r_1 and (11110) is the source vector associated with the tuple t in r'_1. In the tuple (Mary, St. Jack, "01/01/1973"), the value of S_{A_0} is changed to (10010) since this tuple satisfies the second part of the selection condition *F* (*Stud-DOB* > "01/01/1970" AND *Stud-DOB* < "01/01/1980"); in this tuple the values for S_{A_1}, S_{A_2}, S_{A_3} did not change since the source vectors s_1 and s_4 are shown among the source vectors (10110), (11010), and (10010) associated with values corresponding to attributes *Stud-Name*, *Stud-Address*, and *Stud-DOB* respectively. The same argument holds for the tuple (George, St. Dennis, "02/02/1978") of the extended relation r'_1. The tuple t = (Peter, St. Paul, "08/08/1975") in r_1 is presented twice in r'_1 since it satisfies the first part of the selection condition, (*Stud-Address* = "St. Paul" AND *Stud-DOB* > "01/01/1955"), and the second part of the selection condition (*Stud-DOB* > "01/01/1970" AND *Stud-DOB* < "01/01/1980"); (11011), associated with t in r'_1, is the result of the conjunction operation using the source vectors (00011), associated with t in r_1, (11011), associated with *Stud-Address* = "St. Paul" of tuple t in r_1, and (01011) associated with *Stud-DOB* = "01/01/1956" of tuple t in r_1. The source attributes S_{A_1} and S_{A_3} have new values in r'_1. (11011) = (11011) $\wedge$ (01011), where (11011) is the source vector associated with *Stud-Address* = "St. Paul", and (01011) is the source vector associated with *Stud-DOB* = "08/08/1975". The source vector (11011) did not change since the source vectors s_1, s_2, s_4, and s_5 are among the source vectors that are contributing to *Stud-Address* = "St. Paul". Moreover, (01011) is the new source vector associated with the last tuple, as a second instance of tuple t, in the extended relation r'_1 since the tuple t in r_1 satisfies the second part of the selection condition F. The same interpretation holds for the source vectors associated with the attribute values of t in r'_1.

3.3.2 Extended Projection (Π')

Let $r' = \Pi'_X(r)$, where $X = \{A_{i_1}, \ldots, A_{i_m}\} \subseteq \{A_1, \ldots, A_n\}$. Define $S_X = \{S_{A_{i_1}}, \ldots, S_{A_{i_m}}\}$ to be the set of source attributes associated with the regular attributes in X. The extended relational scheme of r' is $R' = \{A_{i_1}, \ldots, A_{i_m}, S_{A_{i_1}}, \ldots, S_{A_{i_m}}\}$ [illegible] $t@\mathcal{U} \in r$; we would like to find $\mathcal{U}'$ such that $t'@\mathcal{U}' \in r'$, where $\mathcal{U}' = \{U'_1, \ldots, U'_p\}$, using $t@\mathcal{U} \in r$. Clearly,

$$r'_1 = \{\, t'@\mathcal{U}' \mid (\forall U \in \mathcal{U})\ (t@U \in r,\ t' = t(X) \text{ and } U' = (u_0, u_{i_1}, \ldots, u_{i_m}))\,\}$$

Example 12 Consider the projection of relation r'_1 of Example 11 The relation r'' $= \Pi'_X(r'_1)$, where $X = \{$ *Stud-Name, Stud-DOB* $\}$, is shown below.

Relation r''				
Stud-Name	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}
John	10/07/1956	11110	11111	11110
Mary	01/01/1973	10010	10010	10010
George	02/02/1978	11001	11011	11001
Peter	08/08/1975	11011	11011	11011
Peter	08/08/1975	01011	11011	01011

3.3.3 Extended Union ($\cup'$)

Let $r_3 = r_1 \cup' r_2$, where r_1 and r_2 have compatible schemes. We define $r_3 = \{t@U \mid t@U \in r_1$ or $t@U \in r_2\}$.

Example 13 As an example, let r_1 and r_2 be the following extended relations on schemas $R_1 = R_2 = \{$ *Stud-Name, Stud-Address, Stud-DOB*, $S_{A_0}, S_{A_1}, S_{A_2}, S_{A_3}\}$.

Relation r_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	10000	10101	11100	10010
Sally	St. Paul	02/06/1954	00011	01111	11011	01111
Mary	St. Jack	01/01/1973	11001	11001	11101	11011
George	St. Dennis	02/02/1978	10001	10011	10011	10001
Peter	St. Paul	08/08/1975	10101	10101	11101	11101

Relation r_2						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
Mark	St. Jack	03/03/1976	10101	10101	11101	11101
John	St. Paul	10/07/1956	10000	10101	11100	10010
George	St. Dennis	02/02/1978	10011	11111	11011	11111
Peter	St. Paul	08/08/1975	00100	01100	11100	01101

Then $r_3 = r_1 \cup' r_2$ is as follows:

Relation r_3						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
Mark	St. Jack	03/03/1976	10101	10101	11101	11101
John	St. Paul	10/07/1956	10000	10101	11100	10010
Sally	St. Paul	02/06/1954	00011	01111	11011	01111
Mary	St. Jack	01/01/1973	11001	11001	11101	11011
George	St. Dennis	02/02/1978	10001	10011	10011	10001
George	St. Dennis	02/02/1978	10011	11111	11011	11111
Peter	St. Paul	08/08/1975	10101	10101	11101	11101
Peter	St. Paul	08/08/1975	00100	01100	11100	01101

3.3.4 Extended Cartesian Product ($\times'$)

Consider the two extended relations r_1 and r_2 defined on the schemes R_1 and R_2, of Section 3.3, respectively, and let $r_3 = r_1 \times' r_2$. The scheme R_3 of the extended relation r_3 is defined as: $R_3 = \{A_1, \ldots, A_n, B_1, \ldots, B_m, S_{A_0}, S_{A_1}, \ldots, S_{A_n}, S_{B_1}, \ldots, S_{B_m}\}$, where $S_{A_1}, \ldots, S_{A_n}, S_{B_1}, \ldots, S_{B_m}$ are the source attributes associated with the regular attributes $A_1, \ldots, A_n, B_1, \ldots, B_m$ respectively. S_{A_0} is the source attribute whose values (source vectors) are associated with the tuples of the extended relation r_3.

Let $W = U \underset{p}{\wedge} V$ be defined as $W = (u_0 \wedge v_0, u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_m)$. Moreover, we define the catenation of the pure tuples t_1 and t_2 as $t_3 = (t_1.t_2)$ to be the regular catenation of two pure tuples t_1 and t_2. Then,

$$r_1 \times' r_2 = \{\ t_3@W \mid t_1@U \in r_1,\ t_2@V \in r_2, \text{ and } W = U \underset{p}{\wedge} V\ \}$$

Example 14 Let r_1 and r_2 be two extended relations on the schemes R_1 and R_2 respectively. R_1 = { *Stud-Name*, $S_{A_{10}}, S_{A_{11}}$ } and R_2 = { *Prof-name*,*Prof-DOB*, $S_{A_{20}}, S_{A_{21}}, S_{A_{22}}$ }

Relation r_1		
Stud-Name	$S_{A_{10}}$	$S_{A_{11}}$
John	0110	1110
Sally	1100	1110

Relation r_2				
Prof-Name	*Prof-DOB*	$S_{A_{20}}$	$S_{A_{21}}$	$S_{A_{22}}$
Heather	01/01/1968	1000	1001	1000
Heather	01/01/1968	0100	1100	0100
Stacy	05/05/1960	1001	1001	1001

Then, $r_3 = r_1 \times' r_2$, is as follows:

Relation r_3						
Stud-Name	*Prof-Name*	*Prof-DOB*	$S_{A_{10}}$	$S_{A_{11}}$	$S_{A_{21}}$	$S_{A_{22}}$
John	Heather	01/01/1968	1110	1110	1111	1110
John	Heather	01/01/1968	0110	1110	1110	0110
John	Stacy	05/05/1960	1111	1111	1111	1111
Sally	Heather	01/01/1968	1100	1110	1101	1100
Sally	Heather	01/01/1968	1100	1110	1100	1100
Sally	Stacy	05/05/1960	1101	1111	1101	1101

3.3.5 Extended Join ($\bowtie'$)

Consider two extended relations r_1 and r_2 on the schemes R_1 and R_2, respectively. Let the set of common attributes be X, i.e. $X = \{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\}$, and let $X_1 = \{A_1, \ldots, A_n\} - X$. We define $S_{X_A} = \{u \mid$ source vector u is associated with $t_1(A), (\forall A), (A \in X)\}$. Similarly, let $X_2 = \{B_1, \ldots, B_m\} - X$. We define $S_{X_B} = \{v \mid$ source vector v is associated with $t_2(B), (\forall B), (B \in X)\}$.

Now, let $S_X = S_{X_A} \cup S_{X_B}$. If $r_3 = r_1 \bowtie' r_2$, then the extended scheme R_3 of the extended relation r_3 is defined as: $R_3 = \{A_{j'_1}, \ldots, A_{j'_{n'}}, A_{j_1}, \ldots, A_{j_{n_1}}, B_{i'_1}, \ldots, B_{i'_{m'}}, S_{A_0}, S_{A_{j'_1}}, \ldots, S_{A_{j'_{n'}}}, S_{A_{j_1}}, \ldots, S_{A_{j_{n_1}}}, S_{B_{i'_1}}, \ldots, S_{B_{i'_{m'}}}\}$, where S_{A_0} is the source attribute whose values (source vectors) are associated with tuples of the extended relation r_3, $S_{A_{j'_1}}, \ldots, S_{A_{j'_{n'}}}$ are the source attributes associated with the attributes $A_{j'_1}, \ldots, A_{j'_{n'}}$, $S_{A_{j_1}}, \ldots, S_{A_{j_{n_1}}}$ are the source attributes associated with the attributes $A_{j_1}, \ldots, A_{j_{n_1}}$, and $S_{B_{i'_1}}, \ldots, S_{B_{i'_{m'}}}$ are the source attributes associated with the attributes $B_{i'_1}, \ldots, B_{i'_{m'}}$.

Define $C(S_X) = \bigwedge_{u \in S_X} u$, we will define $t_3@W \in r_3$,where $t_3 = t_1 o t_2$ and $W = U \bigwedge_J V = (w_0, w_{j'_1}, \ldots, w_{j'_{n'}}, w_{j_1}, \ldots, w_{j_{n_1}}, w_{i'_1}, \ldots, w_{i'_{m'}})$, where $w_0 = (u_0 \wedge v_0 \wedge C(S_X))$, $w_{j'_l} = w_0 \wedge u_{j'_l} (\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w_0 \wedge u_{j_l}$ or $w_{j_l} = w_0 \wedge v_{j_l}$ $(\forall l \in \{1, \ldots, n_1\})$, and $w_{i'_l} = w_0 \wedge v_{i'_l} (\forall l \in \{1, \ldots, m'\})$. w_0 is the source vector associated with the pure tuple t_3, $w_{j'_1}, \ldots, w_{j'_{n'}}$ are the source vectors associated with $t_1(A_{j'_1}), \ldots, t_1(A_{j'_{n'}})$ respectively, $w_{j_1}, \ldots, w_{j_{n_1}}$ are the source vectors associated with $t_1(A_{j_1})$ $(A_{j_1} = B_{i_1})$, $\ldots$, $t_1(A_{j_{n_1}})$ $(A_{j_{n_1}} = B_{i_{m_1}})$ respectively, and $w_{i'_1}, \ldots, w_{i'_{m'}}$ are the source vectors associated with $t_2(B_{i'_1}), \ldots, t_2(B_{i'_{m'}})$ respectively.

$$r_3 = \{t_3@W \mid t_1@U \in r_1, t_2@V \in r_2, t_3 = t_1 o t_2, W = U \bigwedge_J V\},$$

where $t_1 o t_2$ denotes the join of (joinable) tuples t_1 and t_2.

Example 15 As an example, let r_1, which represents the extended relation of students, have the extended relational scheme $R_1 = \{$ *Stud-ID*, *Stud-Name*, $S_{A_{10}}$, $S_{A_{11}}$, $S_{A_{12}}$ $\}$. let r_2, which represents the extended relation of students that are teaching courses, have the extended relational scheme $R_2 = \{$ *Stud-ID*, *Course-Num*, *Course-Date*,$S_{A_{20}}$,$S_{A_{21}}$, $S_{A_{22}}$,$S_{A_{23}}$ $\}$, Let $r_3 = r_1 \bowtie' r_2$, where $R_3 = R_2$ (in this case).

Relation r_1				
Stud-ID	*Stud-Name*	$S_{A_{10}}$	$S_{A_{11}}$	$S_{A_{12}}$
12345	John	010	010	110
12346	Sally	100	100	101

Relation r_2						
Stud-ID	*Course-Num*	*Course-Date*	$S_{A_{20}}$	$S_{A_{21}}$	$S_{A_{22}}$	$S_{A_{23}}$
12345	COMP 546	M 8:00-9:45	100	100	101	100
12346	ELEC 520	T 8:00-9:15	001	001	111	011
12347	DESC 554	M 8:00-9:00	100	110	101	100

Relation r_3								
Stud-ID	*Stud-Name*	*Course-Num*	*Course-Date*	$S_{A_{30}}$	$S_{A_{31}}$	$S_{A_{32}}$	$S_{A_{33}}$	$S_{A_{34}}$
12345	John	COMP 546	M 8:00-9:45	110	110	110	111	110
12346	Sally	ELEC 520	T 8:00-9:15	101	101	101	111	111

To understand the result r_3, let us consider tuple $t_1@U \in r_1$, where t_1 = (12345, John) and U = { (010, 010, 110) }. u_0 = (010) is the source vector associated with tuple t_1, u_1 = (010) is the source vector associated with the attribute value *Stud-ID* = 12345, and u_2 = (110) is the source vector associated with the attribute value *Stud-Name* = "John". In relation r_2, we consider $t_2@V$, where t_2 = (12345, COMP 546, M 8:00-9:45) and V = { (100, 100, 101, 100) }. v_0 = (100) is the source vector associated with tuple t_2, v_1 = (100) is the source vector associated with attribute value *Stud-ID* = 12345, v_2 = (101) is the source vector associated with *Course-Num* = COMP 546, and v_3 = (100) is the source vector associated with *Course-Date* = M 8:00-9:45. When the tuples t_1 and t_2 are joined, we will get $t_3@W \in r_3$, where $t_3 = t_1 o t_2$ (regular join) and $W = (w_0, w_1, w_2, w_3, w_4)$. $= (u_0 \wedge v_0 \wedge u_1 \wedge v_1, w_0 \wedge u_1, w_0 \wedge u_2, w_0 \wedge v_2, w_0 \wedge v_3)$ = (110, 110, 110, 111, 110).

Theorem 16 $r_1 \bowtie' r_2 = \prod'_Y(\sigma'_F(r_1 \times' r_2))$, where $Y = \{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$ and

$$F = \bigwedge_{A \in \{A_1,\ldots,A_n\} \cap \{B_1,\ldots,B_m\}} (r_1.A = r_2.A).$$

Proof: We wil' ~rove the following:

(1). $r_1 \bowtie' r_2 \subseteq \prod'_Y(\sigma'_F(r_1 \times' r_2))$

(2). $\prod'_Y(\sigma'_F(r_1 \times' r_2)) \subseteq r_1 \bowtie' r_2$

(1) Consider $t_3 \in r_1 \bowtie' r_2$, then $(\forall U \in \mathcal{U})$ $(\forall V \in \mathcal{V})$ ($t_3 = t_1 o t_2$ and $W = U \underset{J}{\wedge} V$, where $t_1@U \in r_1$, $t_2@V \in r_2$, and t_1 and t_2 join). $W = (w_0, w_{j'_1}, \ldots, w_{j'_n}, w_{j_1}, \ldots,$

$w_{j_{n_1}}$, $w_{i'_1}, \ldots, w_{i'_{m'}}$), where $w_0 = u_0 \wedge v_0 \wedge C(S_X)$. $w_{j'_l} = w_0 \wedge u_{j'_l} (\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w_0 \wedge u_{j_l} (\forall l \in \{1, \ldots, n_1\})$, $w_{i'_l} = w_0 \wedge v_{i'_l} (\forall l \in \{1, \ldots, m'\})$. Let $r''_3 = r_1 \times' r_2$. $t_1@U \in r_1$ and $t_2@V \in r_2$, we have $t'_3 = t_1.t_2$ and $W'' = U \underset{p}{\wedge} V = (w''_0, w''_1, \ldots, w''_n,$ $w''_1, \ldots, w''_m)$, where $w''_0 = u_0 \wedge v_0$, $w''_j = u_j \wedge w''_0$ $(\forall j \in \{1, \ldots, n\})$, $w''_i = v_i \wedge w''_0$ $(\forall i \in \{1, \ldots, m\})$. It follows that $t'_3@W''_l \in r_1 \times' r_2$. If we consider $r'_3 = \sigma'_F(r''_3)$, then we have $t'_3@W' \in r'_3$; $W' = (w'_0, w'_{j'_1}, \ldots, w'_{j'_{n'}}$ $w'_{j_1}, \ldots, w'_{j_{n_1}}$ $w'_{i'_1}, \ldots, w'_{i'_{m'}}$ $w'_{i_1}, \ldots, w'_{i_{n_1}})$, where $w'_0 = w''_0 \wedge C(S)$, $w'_{j'_l} = w'_0 \wedge w''_{j'_l}$ $(\forall l \in \{1, \ldots, n'\})$, $w'_{j_l} = w'_0 \wedge w''_{j_l}$ $(\forall l \in \{1, \ldots, n_1\})$, $w'_{i'_l} = w'_0 \wedge w''_{i'_l}$ $(\forall l \in \{1, \ldots, m'\})$, and $w'_{i_l} = w'_0 \wedge w''_{i_l}$ $(\forall l \in \{1, \ldots, n_1\})$. If we let $r_3 = \prod'_Y(r'_3)$, then $W = (w_0, w_{j'_1}, \ldots, w_{j'_{n'}},$ $w_{j_1}, \ldots, w_{j_{n_1}}, w_{i'_1}, \ldots, w_{i'_{m'}})$, where $w_0 = w'_0$, $w_{j'_l} = w'_{j'_l} (\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w'_{j_l} (\forall l \in \{1, \ldots, n_1\})$, and $w_{i'_l} = w'_{i'_l} (\forall l \in \{1, \ldots, m'\})$. Therefore, we deduce that $t_3@W \in \prod'_Y(\sigma'_F(r_1 \times' r_2))$.

(2). Consider $t_3 \in \prod'_Y(\sigma'_F(r_1 \times' r_2))$, then there exits $t_1@U \in r_1$ and $t_2@V \in r_2$, where $t'_3 = t_1.t_2$, $W'' = U \underset{p}{\wedge} V = (w''_0, w''_1, \ldots, w''_n, w''_1, \ldots, w''_m)$, where $w''_0 = u_0 \wedge v_0$, $w''_l = w''_0 \wedge u_l$ $(\forall l \in \{1, \ldots, n\})$, $w''_l = w''_0 \wedge v_l$ $(\forall l \in \{1, \ldots, m\})$, and $t'_3@W'' \in r_1 \times' r_2$. It follows that if we let $r'_3 = \sigma'_F(r''_3)$, then we have $t'_3@W' \in r'_3$; $W' = (w'_0, w'_{j'_1}, \ldots, w'_{j'_{n'}}, w'_{j_1}, \ldots, w'_{j_{n_1}}, w'_{i'_1}, \ldots, w'_{i'_{m'}}, w'_{i_1}, \ldots, w'_{i_{m_1}})$, where $w'_0 = w''_0 \wedge C(S)$, $w'_{j'_l} = w'_0 \wedge w''_{j'_l}$ $(\forall l \in \{1, \ldots, n'\})$, $w'_{j_l} = w'_0 \wedge w''_{j_l}$ $(\forall l \in \{1, \ldots, n_1\})$, $w'_{i'_l} = w'_0 \wedge w''_{i'_l}$ $(\forall l \in \{1, \ldots, m'\})$, $w'_{i_l} = w'_0 \wedge w''_{i_l}$ $(\forall l \in \{1, \ldots, m_1\})$, and $t@W' \in r'_3$. Now, let $r_3 = \prod'_Y(r'_3)$, then $W = (w_0, w_{j'_1}, \ldots, w_{j'_{n'}},$ $w_{j_1}, \ldots, w_{j_{n_1}}, w_{i'_1}, \ldots, w_{i'_{m'}})$, where $w_0 = w'_0$, $w_{j'_l} = w'_{j'_l}$ $(\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w'_{j_l}$ $(\forall l \in \{1, \ldots, n_1)\}$, $w_{i'_l} = w'_{i'_l}$ $(\forall l \in \{1, \ldots, m'\})$. Therefore, we deduce that $t_3@W \in r_1 \bowtie' r_2$.

3.3.6 Extended Intersection ($\cap'$)

For two compatible extended relations r_1 and r_2 we define $r_3 = r_1 \cap' r_2$ whose extended scheme, $R_3 = \{A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n}\}$, is compatible with the extended schemes of r_1 and r_2.

It follows that the extended intersection is a special case of an extended join; hence the set of common attributes $X_A = R_1 = R_2$. Then,

$$r_1 \cap' r_2 = \{t@W \mid (W = U \underset{I}{\wedge} V, t@U \in r_1 \; and \; t@V \in r_2)\},$$

where $W = U \underset{I}{\wedge} V = (w_0, w_1, \ldots, w_n)$ such that $w_k = (u_0 \wedge v_0 \wedge \mathcal{C}(S_X))$, $0 \leq k \leq n$.

Example 16 Consider the extended relations r_1 and r_2.

Relation r_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	100	110	110	110
Sally	St. Paul	02/06/1954	001	011	101	011
Mary	St. Jack	01/01/1973	100	100	101	101
George	St. Dennis	02/03/1975	100	101	100	100
Peter	St. Paul	08/08/1975	101	101	101	111

Relation r_2						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
Mark	St. Jack	03/03/1976	010	110	010	110
John	St. Paul	10/07/1956	010	110	110	010
George	St. Dennis	02/03/1975	001	001	101	101
Peter	George Vanier	03/06/1978	001	011	101	011

Then $r_3 = r_1 \cap r_2$ is as follows:

Relation r_3						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	110	110	110	110
George	St. Dennis	02/03/1975	101	101	101	101

In this example, we have $t_1@U \in r_1$, where t_1 = (John, St. Paul, 10/07/1956) and U = (100, 110, 110, 110); $t_1@V \in r_2$, where V = (010, 110, 110, 010).

We notice that the pure tuple t_1 exists in both extended relations. t_1 will appear in the extended relation r_3 with a source vector W that is calculated as follows. $W = (w_0, w_1, w_2, w_3)$, where $w_0 = w_1 = w_2 = w_3 = 100 \wedge 010 \wedge 110 \wedge 110 \wedge 110 \wedge 110 \wedge 110 \wedge 110 \wedge 010 = 110$.

3.3.7 Extended Set Difference ($-'$)

Let r_1 and r_2 be two extended relations having schemes $R_1 = R_2$ respectively. Let $r_3 = r_1 -' r_2$. In this case, r_3 will have a scheme $R_3 = R_1 = R_2$.

Definition 29 For each $t@U \in r$, where $U = (u_0, \ldots, u_n)$ we define the set of contributing source vectors $vectors(U) = \{u_0, u_1, \ldots, u_n\}$. Then

$$r_1 -' r_2 = \{t@U \mid t@U \in r_1 \; and \; \nexists V \; such that \; t@V \in r_2\}$$

$$\cup$$

$$\{t@W \mid t@U \in r_1, t@V \in r_2, \; and \; w_0 \in (\bigvee_{j=1}^{p} \bigwedge(vectors(U_j)) \bigwedge \neg(\bigvee_{j=1}^{q} \bigwedge(vectors(V_j)),$$

$$where \; W = (w_0, \ldots, w_n) \in \mathcal{W}, w_i = w_0 \;, i = 1, \ldots, n\}$$

$$\cup$$

$\{t@W \mid t@U \in r_1, \; there \; exists \; V \; s.t. \; t@V \in r_2, W = (w_0, \ldots, w_n), \; where \; w_0 = u_0 \wedge u'_1, \wedge \ldots, \wedge u'_n, u'_j = u_j \; or \; u'_j = \neg u_j \; and \; at \; least \; one \; u'_j = \neg u_j, \; and \; \forall i = 1, \ldots, n, \; w_i = w_0 \wedge u_i\}$

Example 17 Let r_1 and r_2 be two extended relations having the same schemes R_1 and R_2 respectively.

Relation r_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	100	110	100	110
Sally	St. Paul	02/06/1954	001	011	101	011
Mary	St. Jack	01/01/1973	100	100	101	101
George	St. Dennis	02/03/1975	100	101	110	100
Peter	St. Paul	08/08/1975	101	101	101	111

A pure tuple t that exists in r_1 indicates that the student corresponding to the pure tuple t is registered in the university and is or is not attending courses at the university. A pure tuple t in r_2 indicates that the student corresponding to the pure tuple t is registered in the university and is registered in some courses at the same university.

Relation r_2						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	101	101	111	101
Mary	St. Jack	01/01/1973	100	100	101	101
George	St. Dennis	02/03/1975	001	011	011	001
Peter	St. Paul	08/08/1975	001	101	001	101

Assume a query is interested in the list of students who are registered in the university but are not registered in any course offered by the university. As an answer to this query we are interested in listing the tuples $t_i@W_i \in r_1 -' r_2$ taking into consideration the manipulation of the source vectors contributing to the tuples in the extended relations (r_1 and r_2) and the source vectors contributing to the individual attributes.

Relation r_3						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	11-1	11-1	11-1	11-1
John	St. Paul	10/07/1956	1-10	$\top$	1-10	$\top$
Sally	St. Paul	02/06/1954	001	011	101	011
Mary	St. Jack	01/01/1973	10-1	10-1	$\top$	$\top$
George	St. Dennis	02/03/1975	1-1-1	$\top$	$\top$	1-1-1
George	St. Dennis	02/03/1975	1-11	1-11	$\top$	1-11
George	St. Dennis	02/03/1975	11-1	$\top$	11-1	11-1
Peter	St. Paul	08/08/1975	1-11	1-11	1-11	$\top$

If a pure tuple t appears in r_1 but does not appear in r_2, then this indicates, in this example, that the student is registered in the university but he is not registered in any course since the pure tuple t does not appear in the extended relation r_2. On the other hand, if a pure tuple t appears in both extended relations r_1 and r_2, then this indicates that the student is registered in the university and is registered in some courses too. Note that in both cases, for a pure tuple to exist in any extended relation the sources contributing to the pure tuple t must be *true*.

Consider, for instance, the tuple $t@U \in r_1$, where t = (John, St. Paul, 10/07/1956) and $U = (100, 110, 110, 110)$. Moreover, $t@V \in r_2$, where $V = (101, 101, 111, 101)$. We are interested in finding $t@W \in r_1 -' r_2$. If we consider $t@U \in r_1$ we notice that the tuple exists in r_1 whenever sources contributing to the tuple as well as to the individual attribute values are reliable. Based on this, pure tuple t exists in r_1 provided that sources s_1 and s_2, which is the result of the conjunction between the source vectors (100), (110), (100), and (110), are reliable. Similarly, pure tuple t exists in r_2 provided that sources s_1, s_2 and s_3 are reliable. Now it should be clear that pure tuple t exists in r_3 when the sources contributing to t in r_1 are reliable and the sources contributing to t in r_2 are not. In other words, pure tuple $t@W \in r_3$, where

$$W = (110 \wedge \neg(111), 110 \wedge \neg(111), 110 \wedge \neg(111), 110 \wedge \neg(111)) = (11{-}1, 11{-}1, 11{-}1, 11{-}1).$$

The previous result is not the only case to be considered. There are other cases that should be listed too:

Given $vectors(U) = \{u_1, u_2, u_3\} = \{100, 110\}$

$w_{01} = u_0 \wedge \neg u_1 \wedge u_2 \wedge u_3 = 100 \wedge \neg 110 \wedge 100 \wedge 110 = \top$

$w_{02} = u_0 \wedge u_1 \wedge \neg u_2 \wedge u_3 = 100 \wedge 110 \wedge \neg 100 \wedge 110 = \top$

$w_{03} = u_0 \wedge u_1 \wedge u_2 \wedge \neg u_3 = 100 \wedge 110 \wedge 100 \wedge \neg 110 = \top$

$w_{04} = u_0 \wedge \neg u_1 \wedge \neg u_2 \wedge u_3 = 100 \wedge \neg 110 \wedge \neg 100 \wedge 110 = \top$

$w_{05} = u_0 \wedge \neg u_1 \wedge u_2 \wedge \neg u_3 = 100 \wedge \neg 110 \wedge 100 \wedge \neg 110 = 1-10$

$w_{06} = u_0 \wedge u_1 \wedge \neg u_2 \wedge \neg u_3 = 100 \wedge 110 \wedge \neg 100 \wedge \neg 110 = \top$

$w_{07} = u_0 \wedge \neg u_1 \wedge \neg u_2 \wedge \neg u_3 = 100 \wedge \neg 110 \wedge \neg 100 \wedge \neg 110 = \top$

In this case $\forall i = \{1, \ldots, 7\}$ and $\forall j = \{1, \ldots, n\}$, we will calculate $w_j = w_{0i} \wedge u_j$ to get $w_1 = 1-10 \wedge 110 = \top$, $w_2 = 1-10 \wedge 100 = 1-10$, and $w_3 = 1-10 \wedge 110 = \top$.

As a result, $t@(11-1, 11-1, 11-1, 11-1) \in r_3$ and $t@(1-10, \top, 1-10, \top) \in r_3$. In other words, pure tuple t exists in extended relation r_3 provided that source s_1, s_2 but *not* s_3 are reliable or source s_1 but not s_2 are reliable.

Chapter 4

Semantics and Correctness of Operations

We will present the formal semantics of our model and we will extend the Alternate Worlds Model introduced by Sadri [19].

4.1 Alternate Worlds Model.

In this section we will formalize the semantics of our model. We will extend the approach used by Sadri [19] to prove the correctness of the extended relational algebra operations. "An extended relation represents a set of regular relations. Once we identify this set, we can process a query on the extended relations by processing it against the (regular) relations represented by the extended relations. Of course this approach is not computationally possible in general. What we need is a query processing methodology that is applied to extended relations directly, is efficient, and generates the same answers that would be obtained by processing the query against represented (regular) relations. A query processing methodology that satisfies the latter condition is called "precise."" Sadri [19]

In what follows, we will characterize the set of (regular) relations represented by an extended relation.

Given an extended relation r on the (extended) scheme R , $r = \{ t_1@\mathcal{U}_1, t_2@\mathcal{U}_2, \ldots, t_p@\mathcal{U}_p \}$; we define r^* as a function from the set of subsets of information sources $S = \{ s_1, s_2, ..., s_k \}$ to the set of (regular) relations Rel on the scheme $R - I$, where

I is the set of source attributes, that is,

$$r^* : 2^S \rightsquigarrow Rel$$

Let $Q \subseteq S$ be a set of information sources. Assign truth values "*true*" to sources in Q, and "*false*" to other sources. (We will denote this assignment by the $truth(Q)$). Let $t = (a_1, a_2, \ldots, a_n)$ be a pure tuple such that $t@U \in r$ and $e(u_0) = true$ under truth(Q). Let t' be a tuple in $r^*(Q)$; we define t' using $t@U$ and Q as $t' = \mathcal{COR}(t, U, Q)$, where $t' = (b_1, b_2, \ldots, b_n)$ such that there exists at least one $(b_i \neq NULL)$, and

$b_i = a_i$ if $e(u_i) = true$ under truth(Q)

$b_i =$ NULL[1] if $e(u_i) = false$ under truth(Q).

$r^*(Q) = \{ \ t' \mid t' = \mathcal{COR}(t, U, Q)$, where $t@U \in r$ and $e(u_0) = true$ under truth(Q). $\}$

Definition: The Alternate World of an extended relation r represents the range of function r^*. If we define $rep(r) = r^*$, then the set of (regular) relations $r^*(Q), (\forall Q)(Q \subseteq S)$, is called the *alternate world model* of the extended relation r. In other words, the alternate word representation of an extended relation r is the set of regular relations where each regular relation contains tuples $t' = \mathcal{COR}(t, U, Q)$ *corresponding* to tuples $t@U \in r$ that are valid provided that the sources in Q are correct and all other sources are incorrect, for all $Q \subseteq S$.

Example 18 Consider the extended relation r_1 that represents the information gathered by sources s_1, s_2, and s_3 on students.

Relation r_1						
Stud-Name	*Stud-Address*	*Stud-DOB*	S_{A_0}	S_{A_1}	S_{A_2}	S_{A_3}
John	St. Paul	10/07/1956	100	110	110	100
Sally	St. Paul	02/06/1954	001	011	101	011
Mary	St. Jack	01/01/1973	100	100	101	101
George	St. Dennis	02/03/1975	100	101	100	100
Peter	St. Paul	08/08/1975	101	101	101	011

We would like to find the alternate world representation of r_1. We will consider all subsets (Q) of $\{ s_1, s_2, s_3 \}$ and find their corresponding regular relations. Given $2^S = \{ \{\}, \{s_1\}, \{s_2\}, \{s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, \{s_1, s_2, s_3\} \}$, then the alternate

[1]In our model we assume that all NULL values are different

world representation of r_1 consists of the empty regular relation corresponding to the empty set Q and the following:

$rep(\{s_1\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
NULL	NULL	10/07/1956
Mary	NULL	NULL
NULL	St. Dennis	02/03/1975

$rep(\{s_2\})$ and $rep(\{s_3\})$ are empty.

$rep(\{s_1, s_2\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Mary	NULL	NULL
NULL	St. Dennis	02/03/1975

$rep(\{s_1, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
NULL	NULL	10/07/1956
NULL	St. Paul	NULL
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975
Peter	St. Paul	NULL

$rep(\{s_2, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
Sally	NULL	02/06/1954

$rep(\{s_1, s_2, s_3\})$		
Stud-Name	*Stud-Address*	*Stud-DOB*
John	St. Paul	10/07/1956
Sally	St. Paul	02/06/1954
Mary	St. Jack	01/01/1973
George	St. Dennis	02/03/1975
Peter	St. Paul	08/08/1975

Definition: Let r be an extended relation, and Θ be the unary relational algebra operator. An extended operator Θ' is precise if

$$rep(\Theta'(r)) = \Theta(rep(r))$$

for all extended relations r, where $\Theta(rep(r))$ represents a function f such that $f(Q) = \Theta(r^*(Q))\ \forall Q \subseteq S$.

The graphical representation of this definition is depicted as follows:

$$\begin{tikzcd}
r \arrow[r] \arrow[d] & \Theta'(r) \arrow[d] \\
rep(r) \arrow[r] & rep(\Theta'(r)) = \Theta(rep(r))
\end{tikzcd}$$

Definition: Let r_1 and r_2 be extended relations, and let Θ be a binary relational algebra operator. An extended operator Θ' is precise if

$$rep(r_1\Theta' r_2) = rep(r_1)\Theta rep(r_2)$$

for all extended relations r_1 and r_2, where $rep(r_1)\Theta rep(r_2)$ represents a function f such that $f(Q) = r_1{}^*(Q)\Theta r_2{}^*(Q)\ \forall Q \subseteq S$.

The graphical representation of this definition is depicted as follows:

$$\begin{tikzcd}
r \arrow[d] & s \arrow[r] \arrow[d] & (r)\ \Theta'\ (s) \arrow[d] \\
rep(r) & rep(s) \arrow[r] & rep((r)\ \Theta'(s)) = (rep(r)\Theta(rep(s))
\end{tikzcd}$$

4.2 Extended Relational Algebra Operations are Precise.

In this section, we will prove that the extended relational algebra operations are precise.

4.2.1 Extended Selection is Precise

Theorem 17 (σ' is precise) The extended relational algebra operation Selection is precise.

Proof: Let r_1 be an extended relation whose scheme is defined as $R_1 = \{ A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n} \}$. Let r_2 be the extended relation whose scheme is $R_2 = R_1$ and let $r_2 = \sigma'_F(r_1)$. Let $F = F_1 \vee \ldots, \vee F_q$ where each F_i is a conjunction of simple conditions. Let S be the set of source vectors.
We will show that for any $Q \subseteq S$

1. If $t' \in r_2{}^*(Q)$, then t' satisfies the condition F and $t' \in r_1{}^*(Q)$
2. If $t' \in r_1{}^*(Q)$ and t' satisfies the condition F, then $t' \in r_2{}^*(Q)$.

(1). If $t' \in r_2{}^*(Q)$, then there exist t and U such that $t@U \in r_2$, $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'(A) = NULL\}$. Now, since $r_2 = \sigma'_F(r_1)$, then there should exist t and $\mathcal{V}$, where $t@\mathcal{V} \in r_1$ and t satisfies F. Since t satisfies F, then t satisfies at least one F_i. Let $F_{t_1}, \ldots, F_{t_{q'}}$ be the conjuncts satisfied by t. It follows that there should exist $V = (v_0, v_1, \ldots, v_n)$ such that $t@V \in r_1$, pure tuple t satisfy F_{t_l} for some $l \in \{1, \ldots, q'\}$, and

- $u_0 = v_0 \wedge \mathcal{C}(S_{t_l})$ and $e(u_0) = true$ under truth(Q), where u_0 is the source vector associated with pure tuple t in r_2. It follows that by theorem 8 $e(v_0) = true$ under truth(Q), where v_0 is the source vector associated with pure tuple t in r_1.

- $\forall A_i \in \mathcal{A}, \exists u_i$ associated with $t(A_i)$ in r_2 such that $u_i = u_0 \wedge v_i$ where $e(u_i) = true$ under truth(Q). It follows that by theorem 8 $e(v_i) = true$ under truth(Q), where v_i is the source vector associated with $t(A_i)$ of tuple t in r_1.

- $\forall A_j \in \mathcal{B}, \exists u_j$ associated with $t(A_j)$ in r_2 such that $u_j = u_0 \wedge v_j$ where $e(u_j) = false$ under truth(Q) and $e(u_0) = true$. It follows that by theorem 8 $e(v_j) = false$ under truth(Q), where v_j is the source vector associated with $t(A_j)$ of tuple t in r_1.

Now, pure tuple t satisfies the selection condition F_{t_l} and $e(v_0) = true$ under truth(Q); moreover, since $\forall A_i \in \mathcal{A}$, $e(v_i) = true$ under truth(Q) and $\forall A_j \in \mathcal{B}$, $e(v_j) = false$ under truth(Q), we deduce that $t' \in r_1{}^*(Q)$, where $t' = \mathcal{COR}(t, V, Q)$ and t' satisfies F_{t_l}. Therefore, $rep(\sigma'_F(r_1)) \subseteq \sigma_F(rep(r_1))$.

(2). If $t' \in r_1{}^*(Q)$ and t' satisfies F, then there exist t and V such that $t@V \in r_1$, $e(v_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, V, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'(A) = NULL\}$. Now, since $r_2 = \sigma'_F(r_1)$, where $t@V \in r_1$, and since t satisfies $F = F_1 \vee \ldots \vee F_q$, then t satisfies at least one F_l. Let $F_{t_1}, \ldots, F_{t_{q'}}$ be the conjuncts satisfied by t. It follows that there exist some $U \in \mathcal{U}$ such that $t@U \in r_2$, where $U = (u_0, u_1, \ldots, u_n)$, $u_0 = v_0 \wedge \mathcal{C}(S_{t_l})$, and

- $u_0 = v_0 \wedge \mathcal{C}(S_{t_l})$ and $e(v_0) = true$ under truth(Q), where v_0 the source vector associated with pure tuple t in r_1. Moreover, $e(\mathcal{C}(S_{t_l})) = true$ under truth(Q) since t' satisfies F. It follows that by theorem 8 $e(u_0) = true$ under truth(Q), where u_0 is the source vector associated with pure tuple t in r_2.

- $\forall A_i \in \mathcal{A}$, $\exists v_i$ associated with $t(A_i)$ in r_1 such that $e(v_i) = true$ under truth(Q), it follows that, since $u_i = u_0 \wedge v_i$ and $e(u_0) = true$ under truth(Q), then by theorem 8 $e(u_i) = true$ under truth(Q), where u_i is the source vector associated with $t(A_i)$ of tuple t in r_2.

- $\forall A_j \in \mathcal{B}, \exists v_j$ associated with $t(A_j)$ in r_1 such that $e(v_j) = false$ under truth(Q), it follows that, since $u_j = u_0 \wedge v_j$ and $e(u_0) = true$, then by theorem 8 $e(u_j) = false$ under truth(Q), where u_j is the source vector associated with $t(A_j)$ of tuple t in r_2.

Now, pure tuple t satisfies the selection condition F and $e(u_0) = true$ under truth(Q); moreover, since $\forall A_i \in \mathcal{A}$, $e(u_i) = true$ under truth(Q) and $\forall A_j \in \mathcal{B}$, $e(u_j) = false$ under truth(Q), we deduce that $t' \in r_2{}^*(Q)$, where $t' = \mathcal{COR}(t, U, Q)$. Therefore, $\sigma_F(rep(r_1)) \subseteq rep(\sigma'_F(r_1))$.

4.2.2 Extended Projection is Precise

Theorem 18 ($\prod'$ is precise) The extended relational algebra operation projection is precise.

Proof: Let $r_2 = \prod'_X(r_1)$, and S be the set of source vectors. We will show that:
(1). $r_2{}^*(Q) \subseteq \prod_X r_1{}^*(Q)$,
(2). $\prod_X r_1{}^*(Q) \subseteq r_2{}^*(Q)$
for all $Q \subseteq S$.

(1). Consider a tuple $t' \in r_2{}^*(Q)$, then there exist t and U such that $t@U \in r_2$, $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$

and $\mathcal{B} = \{A \mid t'(A) = NULL\}$. Now, since $r_2 = \prod'_X(r_1)$, then there should exist a tuple t_1 and $\mathcal{V}$, where $t_1@\mathcal{V} \in r_1$ and $t = t_1(X)$. It follows that there exist some $V = (v_0, v_1, \ldots, v_n)$, where $u_0 = v_0$, and $u_{j_k} = v_{j_k}$ $\forall k \in \{1, \ldots, n\}$.

Since $v_0 = u_0$ and $e(u_0) = true$ under truth(Q), then $e(v_0) = true$ under truth(Q). $\forall A_i \in \mathcal{A}$, $\exists u_i$ associated with $t(A_i)$ in r_2 such that $u_i = v_i$, where $e(u_i) = true$ under truth(Q). It follows that $e(v_i) = true$ under truth(Q), where v_i is the source vector associated with $t_1(A_i)$ of tuple t_1 in r_1. $\forall A_j \in \mathcal{B}$, $\exists u_j$ associated with $t(A_j)$ in r_2 such that $u_j = v_j$, where $e(u_j) = false$ under truth(Q). It follows that $e(v_j) = false$ under truth(Q), where v_j is the source vector associated with $t_1(A_j)$ of tuple t_1 in r_1.

Now, since $e(v_0) = true$ under truth(Q) and $\forall A_i \in \mathcal{A}$, $e(v_i) = true$ under truth(Q) and $\forall A_j \in \mathcal{B}$, $e(v_j) = false$ under truth(Q), we deduce that $t'_1 \in r_1{}^*(Q)$, where $t'_1 = \mathcal{COR}(t_1, V, Q)$. Therefore $t' = t'_1(X)$. Hence, for every $t' \in r_2{}^*(Q)$ we can find a tuple $t'_1 \in r_1{}^*(Q)$ such that $t' = t'_1(X)$ and $e(v_0) = true$ under truth(Q) for some $V \in \mathcal{V}$. Therefore, $r_2{}^*(Q) \subseteq \prod_X(r_1{}^*(Q))$.

(2). If $t' \in \prod_X(r_1{}^*(Q))$ such that there exist $t'_1 \in r_1{}^*(Q)$ and $t' = t'_1(X)$. $t'_1 \in r_1{}^*(Q)$, then there exist t_1 and V such that $t_1@V \in r_1$, $e(v_0) = true$ under truth(Q), and $t'_1 = \mathcal{COR}(t_1, V, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'(A) = NULL\}$. Now, since $r_2 = \prod'_X(r_1)$, then there exist a tuple $t@U \in r_2$ such that $t = t_1(X)$ and $e(u_0) = true$ under truth(Q), where $(u_0 = v_0)$. $\forall A_i \in \mathcal{A}$, $\exists v_i$ associated with $t_1(A_i)$ in r_1 such that $u_i = v_i$, where $e(v_i) = true$ under truth(Q). It follows that $e(u_i) = true$ under truth(Q), where u_i is the source vector associated with $t(A_i)$ of tuple t in r_2. $\forall A_j \in \mathcal{B}$, $\exists v_j$ associated with $t_1(A_j)$ in r_1 such that $u_j = v_j$, where $e(v_j) = false$ under truth(Q). It follows that $e(u_j) = false$ under truth(Q), where u_j is the source vector associated with $t(A_j)$ of tuple t in r_2.

Now, since $e(u_0) = true$ under truth(Q), $\forall A_i \in \mathcal{A}$, $e(u_i) = true$ under truth(Q), and $\forall A_j \in \mathcal{B}$, $e(u_j) = false$ under truth(Q), we deduce that $t' \in r_2{}^*(Q)$, where $t' = \mathcal{COR}(t, U, Q)$. Therefore, $\prod_X(r_1{}^*(Q)) \subseteq r_2{}^*(Q)$.

4.2.3 Extended Union is Precise

Theorem 19 ($\cup'$ **is precise) The extended relational algebra operation** *Union* **is precise.**

Proof: We would like to show that for all extended relations r_1 and r_2 we have:

$$rep(r_1 \cup' r_2) = rep(r_1) \cup rep(r_2),$$

where $\cup'$ is the extended union operation and $\cup$ is the regular union operation.

Let S be the set of information sources, and $Q \subseteq S$. Let $r_3 = r_1 \cup' r_2$. We will show that

1. $r_3{}^*(Q) \subseteq r_1{}^*(Q) \cup r_2{}^*(Q)$
2. $r_1{}^*(Q) \cup r_2{}^*(Q) \subseteq r_3{}^*(Q)$

where $r_1{}^*, r_2{}^*$, and $r_3{}^*$ are the functions represented by the extended relations r_1, r_2, and r_3, respectively (see previous sections).

(1). In this case $t' \in r_3{}^*(Q)$; hence there exist t and U such that $t@U \in r_3$, $e(u_0) =$ *true* under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'(A) = NULL\}$. Now, since $r_3 = r_1 \cup' r_2$ and $t@U \in r_3$, then $t@U \in r_1$ or $t@U \in r_2$.

- If $t@U \in r_1$ or $t@U \in r_2$ but $t@U$ does not belong to both we have: $e(u_0) = true$ under truth(Q), $\forall A_i \in \mathcal{A}$, $\exists u_i$ such that $e(u_i) = true$ under truth(Q), $\forall A_j \in \mathcal{B}$, $\exists u_j$ such that $e(u_j) = false$ under truth(Q). It follows that $t' \in r_1{}^*(Q)$ or $t' \in r_2{}^*(Q)$ but not to both. Hence $t' \in r_1{}^*(Q) \cup r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) \cup r_2{}^*(Q)$.

- If $t@U \in r_1$ and $t@U \in r_2$, we have: $e(u_0) = true$ under truth(Q), $\forall A_i \in \mathcal{A}$, $\exists u_i$ such that $e(u_i) = true$ under truth(Q), however, in this case, $\mathcal{B} = \emptyset$ and hence $t' = t$. It follows that $t' \in r_1{}^*(Q) \cup r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) \cup r_2{}^*(Q)$.

(2). If $t' \in r_1{}^*(Q) \cup r_2{}^*(Q)$, then $t' \in r_1{}^*(Q)$ or $t' \in r_2{}^*(Q)$.

- If $t' \in r_1{}^*(Q)$ or $t' \in r_2{}^*(Q)$ but not in both, then there exist t and U such that $t@U \in r_1$ or $t@U \in r_2$ such that $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'(A) = NULL\}$.

- If $t' \in r_1{}^*(Q)$ and $t' \in r_2{}^*(Q)$, then there exist t and U such that $t@U \in r_1$ and $t@U \in r_2$, where $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Let $\mathcal{A} = \{A \mid t'(A) \neq NULL\}$, and $\mathcal{B} = \{A \mid t'(A) = NULL\}$, where in this case we have $\mathcal{B} = \emptyset$ and hence $t' = t$.

Now, In both cases, since $r_3 = r_1 \cup r_2$, it follows that $t@U \in r_3$ and therefore, $r_1{}^*(Q) \cup r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

4.2.4 Extended Cartesian Product is Precise

Theorem 20 ($\times'$ is precise) The extended relational algebra operation Cartesian Product is precise.

Proof: We would like to show that for all extended relations r_1 and r_2 we have:

$$rep(r_1 \times' r_2) = rep(r_1) \times rep(r_2),$$

where $\times'$ is the extended cartesian product operation and $\times$ is the regular cartesian product operation.

Let S be the set of information sources, and $Q \subseteq S$. Let $r_3 = r_1 \times' r_2$. We will show that

1. $r_3{}^*(Q) \subseteq r_1{}^*(Q) \times r_2{}^*(Q)$
2. $r_1{}^*(Q) \times r_2{}^*(Q) \subseteq r_3{}^*(Q)$

where $r_1{}^*, r_2{}^*$, and $r_3{}^*$ are the functions represented by the extended relations r_1, r_2, and r_3, respectively.

(1). If $t'_3 \in r_3{}^*(Q)$, then there exist t_3 and W such that $t_3@W \in r_3$, $e(w_0) = true$ under truth(Q), and $t'_3 = \mathcal{COR}(t_3, W, Q)$. Let $\mathcal{A} = \{A \mid t'_3(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'_3(A) = NULL\}$. Now, since $r_3 = r_1 \times' r_2$, then there should exist tuples t_1 and t_2 such that $t_1@U \in r_1$ and $t_2@V \in r_2$, where $t_3 = t_1.t_2$. Moreover, $W = U \underset{p}{\wedge} V$, $w_0 = u_0 \wedge v_0$, and

- $w_0 = u_0 \wedge v_0$ and $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with pure tuple t_3 in r_3. It follows that by theorem 8 $e(u_0) = true$ and $e(v_0) = true$ under truth(Q), where u_0 is associated with pure tuple t_1 in r_1 and v_0 is associated with t_2 in r_2.

- $\forall A_i \in \mathcal{A} \cap \{A_1, \ldots, A_n\}$, $\exists w_i$ associated with $t_3(A_i)$ in r_3 such that $w_i = w_0 \wedge u_i$ and $e(w_i) = \mathit{true}$ under truth(Q). It follows that by theorem 8 $e(u_i) = \mathit{true}$ under truth(Q), where u_i is the source vector associated with $t_1(A_i)$ of tuple t_1 in r_1.

- $\forall B_i \in \mathcal{A} \cap \{B_1, \ldots, B_m\}$, $\exists w_i$ associated with $t_3(B_i)$ in r_3 such that $w_i = w_0 \wedge v_i$ and $e(w_i) = \mathit{true}$ under truth(Q). It follows that by theorem 8 $e(v_i) = \mathit{true}$ under truth(Q), where v_i is the source vector associated with $t_2(B_i)$ of tuple t_2 in r_2.

- $\forall A_j \in \mathcal{B} \cap \{A_1, \ldots, A_n\}$, $\exists w_j$ associated with $t_3(A_j)$ in r_3 such that $w_j = w_0 \wedge u_j$, where $e(w_j) = \mathit{false}$ and $e(w_0) = \mathit{true}$ under truth(Q). It follows that by theorem 8 $e(u_j) = \mathit{false}$ under truth(Q), where u_j is the source vector associated with $t_1(A_j)$ of tuple t_1 in r_1.

- $\forall B_j \in \mathcal{B} \cap \{B_1, \ldots, B_m\}$, $\exists w_j$ associated with $t_3(B_j)$ in r_3 such that $w_j = w_0 \wedge v_j$, where $e(w_j) = \mathit{false}$ and $e(w_0) = \mathit{true}$ under truth(Q). It follows that by theorem 8 $e(v_j) = \mathit{false}$ under truth(Q), where v_j is the source vector associated with $t_2(B_j)$ of tuple t_2 in r_2.

Now, $e(u_0) = \mathit{true}$ under truth(Q), $\forall A_i \in \mathcal{A} - \{B_1, \ldots, B_m\}$ $e(u_i) = \mathit{true}$ under truth(Q), and $\forall A_j \in \mathcal{B} - \{B_1, \ldots, B_m\}$ $e(u_j) = \mathit{false}$ under truth(Q). It follows that $t'_1 \in r_1{}^*(Q)$, where $t'_1 = \mathcal{COR}(t_1, U, Q)$. Similarly, $e(v_0) = \mathit{true}$ under truth(Q), $\forall B_i \in \mathcal{A} - \{A_1, \ldots, A_n\}$ $e(v_i) = \mathit{true}$ under truth(Q), and $\forall B_j \in \mathcal{B} - \{A_1, \ldots, A_n\}$ $e(v_j) = \mathit{false}$ under truth(Q). It follows that $t'_2 \in r_2{}^*(Q)$, where $t'_2 = \mathcal{COR}(t_2, V, Q)$. Moreover, Since $e(u_0 \wedge v_0) = \mathit{true}$ under truth(Q) and tuple $t'_1 \in r_1{}^*(Q)$ and $t'_2 \in r_2{}^*(Q)$, then $t'_3 = t'_1.t'_2 \in r_1{}^*(Q) \times r_2{}^*(Q)$ and $W = U \underset{p}{\wedge} V$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) \times r_2{}^*(Q)$.

(2). If $t'_3 \in r_1{}^*(Q) \times r_2{}^*(Q)$, then there exist tuples t'_1 and t'_2 in $r_1{}^*(Q)$ and $r_2{}^*(Q)$ respectively such that $t'_3 = t'_1.t'_2$. Since $t'_1 \in r_1{}^*(Q)$, then there exist t_1 and U such that $t_1@U \in r_1$, $e(u_0) = \mathit{true}$ under truth(Q), and $t'_1 = \mathcal{COR}(t_1, U, Q)$. Similarly, since $t'_2 \in r_2{}^*(Q)$, then there exist t_2 and V such that $t_2@V \in r_2$, $e(v_0) = \mathit{true}$

under truth(Q), and $t'_2 = \mathcal{COR}(t_2, V, Q)$. Let $\mathcal{A} = \{A \mid t'_1(A) \neq NULL\} \cup \{B \mid t'_1(B) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'_1(A) = NULL\} \cup \{B \mid t'_1(B) = NULL\}$. Now, since $r_3 = r_1 \times' r_2$, then $t_3@W \in r_3$, where $t_3 = t_1.t_2$, $W = U \underset{p}{\wedge} V = (w_0, u_1, \ldots, u_p, v_1, \ldots, v_q)$, where $w_0 = u_0 \wedge v_0$, and

- $w_0 = u_0 \wedge v_0$, where $e(u_0) = true$ and $e(v_0) = true$ under truth(Q), where u_0 is the source vector associated with t_1 in r_1 and v_0 is the source vector associated with t_2 in r_2. It follows that by theorem 8 $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with t_3 in r_3.

- $\forall A_i \in \mathcal{A} \cap \{A_1, \ldots, A_n\}$, $\exists u_i$ such that u_i is associated with $t_1(A_i)$, where $e(u_i) = true$ under truth(Q). It follows that, since $w_i = w_0 \wedge u_i$, then by theorem 8 $e(w_i) = true$ under truth(Q), where w_i is the source vector associated with $t_3(A_i)$ in r_3.

- $\forall B_i \in \mathcal{A} \cap \{B_1, \ldots, B_m\}$, $\exists v_i$ such that v_i is associated with $t_2(B_i)$, where $e(v_i) = true$ under truth(Q). It follows that, since $w_i = w_0 \wedge v_i$, then by theorem 8 $e(w_i) = true$ under truth(Q), where w_i is the source vector associated with $t_3(B_i)$ in r_3.

- $\forall A_j \in \mathcal{B} \cap \{A_1, \ldots, A_n\}$, $\exists u_j$ such that u_j is associated with $t_1(A_j)$, where $e(u_j) = false$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_j = w_0 \wedge u_j$, then by theorem 8 $e(w_i) = false$, where w_i is the source vector associated with $t_3(B_i)$ of tuple t_3 in r_3.

- $\forall B_j \in \mathcal{B} \cap \{B_1, \ldots, B_m\}$, $\exists v_j$ such that v_j is associated with $t_2(B_j)$, where $e(v_j) = false$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_j = w_0 \wedge v_j$, then by theorem 8 $e(w_i) = false$, where w_i is the source vector associated with $t_3(B_i)$ of tuple t_3 in r_3.

Now, $t_3@W \in r_3$ and $\forall A_i \in \mathcal{A}$, $e(w_i) = true$ under truth(Q), and $\forall A_j \in \mathcal{B}$, $e(w_j) = false$ under truth(Q), we deduce that $t'_3 \in r'_3$, where $t'_3 = \mathcal{COR}(t_3, W, Q)$. Therefore, $r_1{}^*(Q) \times r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

4.2.5 Extended Join is Precise

Theorem 21 ($\bowtie'$ is precise) The extended relational algebra operation Join is precise.

Proof: Let S be the set of information sources and r_1 and r_2 be two extended relations whose schemes are $R_1 = \{ A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n} \}$ and $R2 = \{ B_1, \ldots, B_m, S_{B_0}, S_{B_1}, \ldots, S_{B_m} \}$. Let $X = \{ A_1, \ldots, A_n \} \cap \{ B_1, \ldots, B_m \}$. Let $Q \subseteq S$. Let $r_3 = r_1 \bowtie' r_2$, where $\bowtie'$ is the extended Join operation. We will prove that for every $Q \subseteq S$:

1. $r_3^*(Q) \subseteq r_1^*(Q) \bowtie r_2^*(Q)$
2. $r_1^*(Q) \bowtie r_2^*(Q) \subseteq r_3^*(Q)$

where $\bowtie$ is the regular Join performed on regular relations.

(1). If $t'_3 \in r_3^*(Q)$, then there exist t_3 and W such that $t_3@W \in r_1$, $e(w_0) = true$ under truth(Q), and $t'_3 = \mathcal{COR}(t_3, W, Q)$. Let $\mathcal{A} = \{A \mid t'_3(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'_3(A) = NULL\}$. Now, since $r_3 = r_1 \bowtie' r_2$, then there should exist tuples t_1 and t_2 such that $t_1@U \in r_1$ and $t_2@V \in r_2$, where $t_3 = t_1 o t_2$ (tuples t_1 and t_2 join). Moreover, $W = U \underset{J}{\wedge} V = (w_0, w_{j'_1}, \ldots, w_{j'_{n'}}, w_{j_1}, \ldots, w_{j_{n_1}}, w_{i'_1}, \ldots, w_{i'_{m'}})$, where $w_0 = (u_0 \wedge v_0 \wedge \mathcal{C}(S_X))$, $w_{j'_l} = w_0 \wedge u_{j'_l} (\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w_0 \wedge u_{j_l}$ or $w_{j_l} = w_0 \wedge v_{j_l}$ $(\forall l \in \{1, \ldots, n_1\})$, $w_{i'_l} = w_0 \wedge v_{i'_l} (\forall l \in \{1, \ldots, m'\})$, and

- $w_0 = u_0 \wedge v_0 \wedge \mathcal{C}(S_X)$ and $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with t_3 in r_3. It follows that by theorem 8 $e(u_0) = true$ and $e(v_0) = true$ under truth(Q), where u_0 is associated with pure tuple t_1 in r_1 and v_0 is associated with pure tuple t_2 in r_2.

- $\forall A_i \in \mathcal{A} \cap \{A_1, \ldots, A_n\} - X$, $\exists w_i$ associated with $t_3(A_i)$ in r_3 such that $w_i = w_0 \wedge u_i$ and $e(w_i) = true$ under truth(Q). It follows that by theorem 8 $e(u_i) = true$ under truth(Q), where u_i is the source vector associated with $t_1(A_i)$ of tuple t_1 in r_1.

- $\forall B_i \in \mathcal{A} \cap \{B_1, \ldots, B_m\} - X$, $\exists w_i$ associated with $t_3(B_i)$ in r_3 such that $w_i = w_0 \wedge v_i$ and $e(w_i) = true$ under truth(Q). It follows that by theorem 8 $e(v_i) = true$

under truth(Q), where v_i is the source vector associated with $t_2(B_i)$ of tuple t_2 in r_2.

- $\forall A_i \in \mathcal{A} \cap X$, $\exists w_i$ associated with $t_3(A_i)$ of tuple t_3 in r_3 such that $w_i = w_0 \wedge u_i$ or $w_i = w_0 \wedge v_i$ and $e(w_i) = true$ under truth(Q). It follows that by theorem 8 $e(u_i) = true$ under truth(Q) and $e(v_i) = true$ under truth(Q), where u_i is the source vector associated with $t_1(A_i)$ of tuple t_1 in r_1 and v_i is the source vector associated with $t_2(A_i)$ of tuple t_2 in r_2.

- $\forall A_j \in \mathcal{B} \cap \{A_1, \ldots, A_n\} - X$, $\exists w_j$ associated with $t_3(A_j)$ of tuple t_3 in r_3 such that $w_j = w_0 \wedge u_j$, where $e(w_j) = false$ and $e(w_0) = true$ under truth(Q). It follows that by theorem 8 $e(u_j) = false$ under truth(Q), where u_j is the source vector associated with $t_1(A_j)$ of tuple t_1 in r_1.

- $\forall B_j \in \mathcal{B} \cap \{B_1, \ldots, B_m\} - X$, $\exists w_j$ associated with $t_3(B_j)$ of tuple t_3 in r_3 such that $w_j = w_0 \wedge v_j$, where $e(w_j) = false$ and $e(w_0) = true$ under truth(Q). It follows that by theorem 8 $e(v_j) = false$ under truth(Q), where v_j is the source vector associated with $t_2(B_j)$ of tuple t_2 in r_2.

Now, $e(u_0) = true$ under truth(Q) and $\forall A_i \in \mathcal{A} - \{B_1, \ldots, B_m\}$, $e(u_i) = true$ under truth(Q) and $\forall A_j \in \mathcal{B} - \{B_1, \ldots, B_m\}$, $e(u_j) = false$ under truth(Q). It follows that $t'_1 \in r_1{}^*(Q)$, where $t'_1 = \mathcal{COR}(t_1, U, Q)$. Similarly, $e(v_0) = true$ under truth(Q) and $\forall B_i \in \mathcal{A} - \{A_1, \ldots, A_n\}$, $e(v_i) = true$ under truth(Q) and $\forall B_j \in \mathcal{B} - \{A_1, \ldots, A_n\}$, $e(v_j) = false$ under truth(Q). It follows that $t'_2 \in r_2{}^*(Q)$, where $t'_2 = \mathcal{COR}(t_2, V, Q)$. Moreover, Since $e(\mathcal{C}(S_X)) = true$ under truth(Q), then tuple $t'_1 \in r_1{}^*(Q)$ and $t'_2 \in r_2{}^*(Q)$ join on X. Hence $t'_3 = t'_1 o t'_2 \in r_1{}^*(Q) \bowtie r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) \bowtie r_2{}^*(Q)$.

(2). If $t'_3 \in r_1{}^*(Q) \bowtie r_2{}^*(Q)$, then there exist tuples t'_1 and t'_2 in $r_1{}^*(Q)$ and $r_2{}^*(Q)$ respectively such that $t'_3 = t'_1 o t'_2$. Since $t'_1 \in r_1{}^*(Q)$, then there exist U, where $e(u_0) = true$ under truth(Q) and $t'_1 = \mathcal{COR}(t_1, U, Q)$. Similarly, since $t'_2 \in r_2{}^*(Q)$, then there exist V, where $e(v_0) = true$ under truth(Q) and $t'_2 = \mathcal{COR}(t_2, V, Q)$. Let $\mathcal{A} = \{A \mid t'_1(A) \neq NULL\} \cup \{B \mid t'_1(B) \neq NULL\}$ and $\mathcal{B} = \{A \mid t'_1(A_j) = NULL\}$

$\cup \{B \mid t'_1(B_j) = NULL\}$. Now, since $r_3 = r_1 \bowtie' r_2$ and $t_1@U \in r_1$ and $t_2@V \in r_2$, then $t_3@W \in r_3$, where $t_3 = t_1 o t_2$ and $W = U \underset{J}{\wedge} V = (w_0, w_{j'_1}, \ldots, w_{j'_{n'}}, w_{j_1}, \ldots, w_{j_{n_1}}, w_{i'_1}, \ldots, w_{i'_{m'}})$, where $w_0 = (u_0 \wedge v_0 \wedge \mathcal{C}(S_X))$, $w_{j'_l} = w_0 \wedge u_{j'_l} (\forall l \in \{1, \ldots, n'\})$, $w_{j_l} = w_0 \wedge u_{j_l}$ or $w_{j_l} = w_0 \wedge v_{j_l}$ $(\forall l \in \{1, \ldots, n_1\})$, $w_{i'_l} = w_0 \wedge v_{i'_l} (\forall l \in \{1, \ldots, m'\})$, and

- $w_0 = u_0 \wedge v_0 \wedge \mathcal{C}(S_X)$, where $e(u_0) = true$ under under truth(Q), $e(v_0) = true$ under truth(Q), and $e(\mathcal{C}(S_X)) = true$ under truth(Q). u_0 is the source vector associated with t_1 in r_1 and v_0 is the source vector associated with t_2 in r_2. It follows that by theorem 8 $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with t_3 in r_3.

- $\forall A_i \in \mathcal{A} \cap \{A_1, \ldots, A_n\} - X$, $\exists u_i$ such that u_i is associated with $t_1(A_i)$, where $e(u_i) = true$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_i = w_0 \wedge u_i$, then by theorem 8 $e(w_i) = true$ under truth(Q), where w_i is the source vector associated with $t_3(A_i)$ of tuple t_3 in r_3.

- $\forall B_i \in \mathcal{A} \cap \{B_1, \ldots, B_m\} - X$, $\exists v_i$ such that v_i is associated with $t_1(B_i)$, where $e(v_i) = true$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_i = w_0 \wedge u_i$, then by theorem 8 $e(w_i) = true$ under truth(Q), where w_i is the source vector associated with $t_3(B_i)$ of tuple t_3 in r_3.

- $\forall A_i \in \mathcal{A} \cap X$, $\exists u_i$ *and* v_i such that u_i is associated with $t_1(A_i)$ and v_i is associated with $t_2(B_i)$, where $e(u_i) = e(v_i) = true$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_i = w_0 \wedge u_i$ or $w_i = w_0 \wedge v_i$, then by theorem 8 $e(w_i) = true$ under truth(Q), where w_i is the source vector associated with $t_3(A_i) = t_3(B_i)$ of tuple t_3 in r_3, where $A_i = B_i$.

- $\forall A_j \in \mathcal{B} \cap \{A_1, \ldots, A_n\} - X$, $\exists u_j$ such that u_j is associated with $t_1(A_j)$, where $e(u_j) = false$ and $e(w_0) = true$ under truth(Q). It follows that, since $w_j = w_0 \wedge u_j$, then by theorem 8 $e(w_j) = false$ under truth(Q), where w_j is the source vector associated with $t_3(A_j)$ of tuple t_3 in r_3.

- $\forall B_j \in \mathcal{B} \cap \{B_1, \dots, B_m\} - X$, $\exists v_j$ such that v_j is associated with $t_2(B_j)$, where $e(v_j) = \textit{false}$ and $e(w_0) = \textit{true}$ under truth(Q). It follows that, since $w_j = w_0 \wedge v_j$, then by theorem 8 $e(w_j) = \textit{false}$ under truth(Q), where w_j is the source vector associated with $t_3(B_j)$ of tuple t_3 in r_3.

Now, $t_3@W \in r_3$ and $\forall A_{i_1}, B_{i_2} \in \mathcal{A}$, $e(w_{i_1}) = e(w_{i_2}) = \textit{true}$ under truth(Q), and $\forall A_{j_1}, B_{j_2} \in \mathcal{B}$, $e(w_{j_1}) = e(w_{j_2}) = \textit{false}$ under truth(Q), we deduce that $t'_3 \in r'_3$, where $t'_3 = \mathcal{COR}(t_3, W, Q)$. Therefore, $r_1{}^*(Q) \bowtie r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

4.2.6 Extended Intersection is Precise

Theorem 22 ($\cap'$ is precise) The extended relational algebra operation Intersection is precise.

Proof: We would like to show that for all extended relations r_1 and r_2 we have:

$$rep(r_1 \cap' r_2) = rep(r_1) \cap rep(r_2), \text{ where}$$

$\cap'$ is the extended intersection and $\cap$ is the regular intersection operation.

Let S be the set of information sources, and $Q \subseteq S$. Let $r_3 = r_1 \cap' r_2$. We will show that

1. $r_3{}^*(Q) \subseteq r_1{}^*(Q) \cap r_2{}^*(Q)$
2. $r_1{}^*(Q) \cap r_2{}^*(Q) \subseteq r_3{}^*(Q)$

where $r_1{}^*, r_2{}^*$, and $r_3{}^*$ are the functions represented by the extended relations r_1, r_2, and r_3, respectively.

(1). If $t' \in r_3{}^*(Q)$, then there exist t and W such that $t@W \in r_3$, where $e(w_0) = \textit{true}$ under truth(Q) and $t' = \mathcal{COR}(t, W, Q)$. Let $\mathcal{A} = \{A \mid t(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t(A) = NULL\}$. Now, since $r_3 = r_1 \cap' r_2$, then there should exist U and V such that $t@U \in r_1$ and $t@V \in r_2$. Moreover, $W = U \underset{I}{\wedge} V$ (as previously defined) and $\forall i = 0, \dots, n$, $w_i = u_0 \wedge v_0 \wedge \mathcal{C}(S_X)$, where $e(w_i) = \textit{true}$ under truth(Q). It follows that by theorem 8 $e(u_i) = \textit{true}$, $e(v_i) = \textit{true}$, and $e(\mathcal{C}(S_X)) = \textit{true}$ under truth(Q). Now, we have $t@U \in r_1$ where $e(u_0) = \textit{true}$ under truth(Q) and $\forall A_i \in \mathcal{A}$, $e(u_i) = \textit{true}$ under truth(Q). It follows that $t' \in r_1{}^*(Q)$, where $t' = \mathcal{COR}(t, U, Q)$. Similarly, we have $t@V \in r_2$, where $e(v_0) = \textit{true}$ under truth(Q) and $\forall A_i \in \mathcal{A}$,

$e(v_i) = true$ under truth(Q). It follows that $t' \in r_2^*(Q)$, where $t' = \mathcal{COR}(t, V, Q)$. Hence $t' \in r_1^*(Q) \cap r_2^*(Q)$. Therefore, $r_3^*(Q) \subseteq r_1^*(Q) \cap r_2^*(Q)$.

(2). If $t' \in r_1^*(Q) \cap r_2^*(Q)$, then $t' \in r_1^*(Q)$ and $t' \in r_2^*(Q)$. Since $t' \in r_1^*(Q)$, then there exist t and U such that $t@U \in r_1$, where $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, Q)$. Similarly, since $t' \in r_2^*(Q)$, then there exist t and V such that $t@V \in r_2$, where $e(v_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, V, Q)$. Let $\mathcal{A} = \{A \mid t(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t(A) = NULL\}$. Now, since $r_3 = r_1 \cap' r_2$, then $t@W \in r_3$, and $W = U \underset{I}{\wedge} V$ (as previously defined), and $\forall i = 0, \ldots, n$, $w_i = u_0 \wedge v_0 \wedge \mathcal{C}(S_X)$, where $e(u_0) = true$ under under truth(Q), $e(v_0) = true$ under truth(Q), and $e(\mathcal{C}(S_X)) = true$ under truth(Q). Source vector u_0 is associated with t in r_1 and source vector v_0 is associated with t in r_2. It follows that by theorem 8 $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with t in r_3. Now, $t@W \in r_3$, $e(w_0) = true$ under truth(Q) and $\forall A_i \in \mathcal{A}$, $e(w_i) = true$. We deduce that $t' \in r'_3$, where $t' = \mathcal{COR}(t, W, Q)$. Therefore, $r_1^*(Q) \cap r_2^*(Q) \subseteq r_3^*(Q)$.

4.2.7 Extended Set Difference is Precise

Theorem 23 ($-'$ is precise) The extended relational algebra operation Set Difference is precise.

Proof: We would like to show that for all extended relations r_1 and r_2 having the same compatible extended relational schemes

$$rep(r_1 -' r_2) = rep(r_1) - rep(r_2), \text{ where}$$

$-'$ is the extended set difference and $-$ is the regular set difference operation.

Let S be the set of information sources, and $Q \subseteq S$. Let $r_3 = r_1 -' r_2$. We will show that

1. $r_3^*(Q) \subseteq r_1^*(Q) - r_2^*(Q)$
2. $r_1^*(Q) - r_2^*(Q) \subseteq r_3^*(Q)$

where r_1^*, r_2^*, and r_3^* are the functions represented by the extended relations r_1, r_2, and r_3, respectively.

(1). If $t' \in r_3{}^*(Q)$, then there exist t and $W \in \mathcal{W}$ such that $t@W \in r_3$, where $e(w_0) = \mathit{true}$ under truth(Q) and $t' = \mathcal{COR}(t, W, Q)$. Let $\mathcal{A} = \{A \mid t(A) \neq NUL\}$ and $\mathcal{B} = \{A \mid t(A) = NULL\}$. Now, since $r_3 = r_1 -' r_2$, then we have three cases to look at.

- Case (a). In this case, $t@W \in r_1$ and t does not appear in r_2. we have $e(w_0) = \mathit{true}$ under truth(Q), $\forall A_i \in \mathcal{A}$, $\exists w_i$ associated with $t(A_i)$ in r_1, where $e(w_i) = \mathit{true}$ under truth(Q), and $\forall A_j \in \mathcal{B}$, $\exists w_j$ associated with $t(A_j)$ in r_1, where $e(w_j) = \mathit{false}$ under truth(Q). It follows that $t' = \mathcal{COR}(t, W, Q) \in r_1{}^*(Q)$ and $t' \notin r_2{}^*(Q)$. Hence $t' \in r_1{}^*(Q) - r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) - r_2{}^*(Q)$.

- Case (b). In this case,

$$x = (\bigvee_{j=1}^{p} \bigwedge(vectors(U_j))) \bigwedge \neg(\bigvee_{j=1}^{q} \bigwedge(vectors(V_j))),$$

for $W \in \mathcal{W}$, $W = (w_0, \ldots, w_n)$, then for all $i \in \{1, \ldots, n\}$ $w_i = w_0$, where $w_0 \in x$, and $e(x) = \mathit{true}$ under truth(Q). Hence by theorem 8

$$e(\bigvee_{j=1}^{p} \bigwedge(vectors(U_j))) = e(\neg(\bigvee_{j=1}^{q} \bigwedge(vectors(V_j))) = \mathit{true}$$

under truth(Q).

Now, since $e((\bigwedge(vectors(U_1))) \bigvee \ldots \bigvee(\bigwedge(vectors(U_p)))) = \mathit{true}$ under truth(Q), then there exist at least one $u = \bigwedge(vectors(U))$, for some $U \in \{U_1, \ldots, U_p\}$, such that $e(u) = \mathit{true}$ under truth(Q), where $U = (u_0, \ldots, u_n)$. Now, by theorem 8, $\forall i \in \{0, \ldots, n\}$, $e(u_i) = \mathit{true}$ under truth(Q). It follows that $e(u_0) = \mathit{true}$ and $\forall A_i \in \mathcal{A}$, $\exists u_i$ such that $e(u_i) = \mathit{true}$ under truth(Q) and hence $t' = \mathcal{COR}(t, U, Q) \in r_1{}^*(Q)$. Notice that in this case $\mathcal{B} = \emptyset$.

Similarly, by theorem 8, since $e(\neg((\bigwedge(vectors(V_1))) \bigvee \ldots \bigvee(\bigwedge(vectors(V_q)))) = \mathit{true}$ under truth(Q), then $\forall v = \bigwedge(vectors(V))$, $(V \in \{V_1, \ldots, V_q\})$, $e(v) = \mathit{false}$ under truth(Q), where $V = (v_0, \ldots, v_n)$. Hence, there exist at least one $v_i \in \{v_0, \ldots, v_n\}$ such that $e(v_i) = \mathit{false}$ under truth(Q). Hence, $\nexists V \in$

$\{V_1, \ldots, V_q\}$ such that $\forall A_i \in \mathcal{A}$, $e(v_i) = true$ under truth(Q) and in this case $t' \notin r_2{}^*(Q)$.

Now, since $t' = \mathcal{COR}(t, U, Q) \in r_1{}^*(Q)$ and $t' = \mathcal{COR}(t, U, Q) \notin r_2{}^*(Q)$, then $t' \in r_1{}^*(Q) - r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) - r_2{}^*(Q)$.

- Case (c). In this case, $t@W \in r_1$, $W = (w_0, \ldots, w_n)$, where there exists U and V such that: $t@U \in r_1$, $t@V \in r_2$ and $w_0 = u_0 \wedge u'_1, \wedge \ldots, \wedge u'_n$, $u'_j = u_j$ or $u'_j = \neg u_j$ and at least one $u'_j = \neg u_j$, and $\forall i = 1, \ldots, n$ $w_i = w_0 \wedge u_i$. Moreover,

  1. Since $e(w_0) = true$ under truth(Q), where w_0 is the source vector associated with t in r_3, it follows that, by theorem 8 $e(u_0) = true$ under truth(Q), where u_0 is the source vector associated with t in r_1.
  2. $\forall A_i \in \mathcal{A}$, $\exists w_i$ associated with $t(A_i)$ of tuple t in r_3 such that $e(w_i) = true$ under truth(Q) and $w_i = w_0 \wedge u_i$. It follows that by theorem 8 $e(u_i) = true$ under truth(Q), where u_i is the source vector associated with $t(A_i)$ of tuple t in r_1.
  3. $\forall A_j \in \mathcal{B}$, $\exists w_j$ associated with $t(A_j)$ of tuple t in r_3 such that $e(w_j) = false$ and $e(w_0) = true$ under truth(Q), where $w_j = w_0 \wedge u_j$. It follows that by theorem 8 $e(u_j) = false$ under truth(Q), where u_j is the source vector associated with $t(A_j)$ of tuple t in r_1.

  Now, we have $t@U \in r_1$, where $e(u_0) = true$, $\forall A_i \in \mathcal{A}$, $\exists u_i$ such that $e(u_i) = true$ under truth(Q), and $\forall A_j \in \mathcal{B}$, $\exists u_j$ such that $e(u_j) = false$ under truth(Q). Hence $t' \in r_1{}^*(Q)$. Moreover, since in our model all $NULL$ values are assumed to be different then it is always true (In Case C) that $t' \notin r_2{}^*(Q)$. Therefore, $r_3{}^*(Q) \subseteq r_1{}^*(Q) - r_2{}^*(Q)$.

(2). If $t' \in r_1{}^*(Q) - r_2{}^*(Q)$, then $t' \in r_1{}^*(Q)$ and $t' \notin r_2{}^*(Q)$. Now, since $t' \in r_1{}^*(Q)$, then there should exist a pure tuple t and $U \in \mathcal{U}$, associated with t in r_1, such that $t@U \in r_1$, $e(u_0) = true$ under truth(Q), and $t' = \mathcal{COR}(t, U, r_1)$. Let $\mathcal{A} = \{A \mid t(A) \neq NULL\}$ and $\mathcal{B} = \{A \mid t(A) = NULL\}$. Moreover, since $r_3 = r_1 -' r_2$ and $t' \notin r_2{}^*(Q)$ then we consider the following cases:

- Case (a). In this case, $t@U \in r_1$ and t does not appear in r_2. We have $e(u_0) = \mathit{true}$ under truth(Q), Since $\forall A_i \in \mathcal{A}$ $e(u_i) = \mathit{true}$ under truth(Q), and $\forall A_j \in \mathcal{A}$ $e(u_j) = \mathit{false}$ under truth(Q), it follows that $t@U \in r_3$ and $t' = \mathcal{COR}(t, U, r_3)$. Hence $t' \in r_3{}^*(Q)$. Therefore, $r_1{}^*(Q) - r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

- Case (b). In this case

$$e(\bigvee_{j=1}^{p} \bigwedge(\mathit{vectors}(U_j))) = e(\neg(\bigvee_{j=1}^{q} \bigwedge(\mathit{vectors}(V_j)))) = \mathit{true}$$

  under truth(Q), for $W \in \mathcal{W}$, $W = (w_0, \ldots, w_n)$, then for all $i \in \{1, \ldots, n\}$ $w_i = w_0$, s.t. $w_0 \in x$. It follows by theorem 8 $e(x) = \mathit{true}$ under truth(Q) and there should exist at least one $w \in x$ such that $e(w) = \mathit{true}$ under truth(Q) and $W = (w_0, \ldots, w_n)$ such that $\forall i \in \{0, \ldots, n\}$ $e(w) = e(w_i) = \mathit{true}$ under truth(Q). Hence $t@W \in r_3$ and $e(w_0) = \mathit{true}$ under truth(Q) and since $\forall A_i \in \mathcal{A}$, $e(w_i) = \mathit{true}$ under truth(Q), where w_i is associated with $t(A_i)$ of tuple t in r_3, then $t' = \mathcal{COR}(t, W, r_3)$. Note that in this case $\mathcal{B} = \emptyset$. Hence, $t' \in r_3{}^*(Q)$. Therefore, $r_1{}^*(Q) - r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

- Case (c). In this case, $t@U \in r_1$, and there exist V s.t. $t@V \in r_2, W = (w_0, \ldots, w_n)$, where $w_0 = u_0 \wedge u'_1 \wedge \ldots \wedge u'_n, u'_j = u_j$ or $u'_j = \neg u_j$ and at least one $u'_j = \neg u_j$, and $\forall i = 1, \ldots, n$, $w_i = w_0 \wedge u_i$. Moreover,

  1. since $w_0 = u_0 \wedge u'_1 \wedge \ldots \wedge u'_n$, and $\forall A_i \in \mathcal{A}$, $\exists u_i$ such that $e(u_i) = \mathit{true}$ under truth(Q), where $u_i = u'_i$; similarly, $\forall A_j \in \mathcal{B}$ $\exists u_j$ such that $e(u_j) = \mathit{false}$ or $e(\neg u_j) = \mathit{true}$ under truth(Q), where $\neg u_j = u'_j$. It follows that by theorem 8 $e(w_0) = \mathit{true}$ under truth(Q).where w_0 is the source vector associated with t in r_3.

  2. $\forall A_i \in \mathcal{A}$, $\exists u_i$, associated with $t(A_i)$ of tuple t in r_1, such that $e(u_i) = \mathit{true}$ under truth(Q). Since $w_i = w_0 \wedge u_i$, then by theorem 8 $e(w_i) = \mathit{true}$ under truth(Q), where w_i is the source vector associated with $t(A_i)$ of pure tuple t in r_3.

3. $\forall A_j \in \mathcal{B}$, $\exists u_j$, associated with $t(A_j)$ of tuple t in r_1, such that $e(u_j) = false$ under truth(Q). Since $w_j = w_0 \wedge u_j$, then by theorem 8 $e(w_j) = false$ under truth(Q), where w_j is the source vector associated with $t(A_j)$ of pure tuple t in r_3.

   Now, we have $t@W \in r_3$, where $e(w_0) = true$ under truth(Q), and $\forall A_i \in \mathcal{A}$, $\exists w_i$, such that $e(w_i) = true$ under truth(Q), and $\forall A_j \in \mathcal{B}$, $\exists w_j$ such that $e(w_i) = false$ under truth(Q). It follows that $t' = \mathcal{COR}(t, W, r_3)$. Therefore, $t' \in r_3{}^*(Q)$ and $r_1{}^*(Q) - r_2{}^*(Q) \subseteq r_3{}^*(Q)$.

Chapter 5

Reliability Calculation Algorithms

We will use the Reliability Calculation Algorithms, Presented by Sadri [18, 19], in our new model and prove their correctness using the semantics of the Alternate Worlds Model of our new frame work.

5.1 Reliability Calculation

Knowing the reliability of information sources that contributed to the data found in the tuple in an extended relational database system, we can calculate the reliability of answers to a query in the extended relational database.

Definition 30 The reliability of a source is defined as the probability that a tuple coming from that source is valid. We designate the reliability of source j by $re(s_j)$. Moreover, We assume that different information sources are independent.

Let r be an extended relation which is the result of some query and let $R = \{A_1, \ldots, A_n, S_{A_0}, S_{A_1}, \ldots, S_{A_n}\}$ be the extended relational scheme of r. Consider a tuple $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U\}$. Let $U = (u_0, u_1, \ldots, u_n)$, where u_0 is the source vector associated with pure tuple t and $u_1, \ldots, u_n$ are the source vectors associated with the attribute values $t(A_1), \ldots, t(A_n)$ of pure tuple t in r.

Given the reliability of the contributing source vectors, we would like to calculate the reliability or the probability that tuple t is in the answer to the query. Recall that $vectors(U) = \{u_0, u_1, \ldots, u_n\}$. Let v be the source vector obtained by taking the conjunction of all source vectors in the set $vectors(U)$ as follows:

$$v = \bigwedge vectors(U)$$

Now, we consider the sources $s_{i_1}, \ldots, s_{i_p}$ corresponding to the elements of v with a value of 1. In other words, $v(i_1) = \ldots = v(i_p) = 1$, where $\forall i \in \{i_1, \ldots i_p\}$, $v(i)$ represent the i^{th} element of the source vector v. Similarly, consider the sources $s_{j_1}, \ldots, s_{j_q}$ corresponding to the elements of v with a value of -1. In other words, $v(j_1) = \ldots = v(j_q) = -1$, where $\forall j \in \{j_1, \ldots j_q\}$, $v(j)$ represents the j^{th} element of the source vector v. Given the reliability of source vectors $s_1, \ldots, s_k$, the reliability of the pure tuple t $rel(t)$ is:

$$rel(t@U) = \prod_{k=i_1}^{i_p} re(s_k) \prod_{l=j_1}^{j_q} (1 - re(s_l))$$

$rel(t@U)$ indicates the probability or the reliability with which pure tuple t exists in r. In other words pure tuple t exists in r provided that the sources $s_{i_1}, \ldots, s_{i_p}$ are reliable (oı correct) and the sources $s_{j_1}, \ldots, s_{j_q}$ are not reliable (or not correct).

Definition 31 Two source vectors u and v are said to be *independent* if for no source s_j both of them have a nonzero entry at the j^{th} element of u and v. This definition could be extended to define independence between sets of source vectors. A set of source vectors $\{u_1, \ldots, u_n\}$ is *independent* if the source vectors in $\{u_1, \ldots, u_n\}$ are pair wise independent

Let $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U_1, \ldots, U_p\}$ and let

$$x = \{v \mid v_i = \bigwedge vectors(U_i), i = 1, \ldots, p\}$$

If V is a set of independent source vectors, then the reliability of t is calculated as follows:

$$rel(t@\mathcal{U}) = 1 - \prod_{v \in x} (1 - rel(t@v)).$$

5.1.1 Reliability Calculation Algorithms

In the previous section we defined the reliability of the pure tuple t when the set $x = \{v \mid v_i = \bigwedge vectors(U_i), i = 1, \ldots, p\}$ is a set of independent source vectors. It is sometimes desirable, in some application domains, to calculate the reliability of t when x is a set of dependent source vectors. In this section, we will calculate the reliability of the pure tuple t when the set x has dependent source vectors. In pursuing this line, we will use the reliability calculation algorithms introduced by Sadri [18, 19].

In what follows we assume $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U_1, \ldots, U_p\}$ is associated with pure tuple t in r. As usual, let $x = \{v \mid v_i = \bigwedge vectors(U_i), i = 1, \ldots, p\}$ be a dependent set of source vectors.

Algorithm 3 To calculate the reliability of the pure tuple $t@x$, we proceed as follows:

$$K_1 = \sum_{i=1}^{p} re(v_i)$$

$$K_2 = \sum_{i=1}^{p} \sum_{j>i}^{p} re(v_i \wedge v_j)$$

$$K_3 = \sum_{i=1}^{p} \sum_{j>i}^{p} \sum_{k>j}^{p} re(v_i \wedge v_i \wedge v_k)$$

$$\vdots$$

Then

$$re(t@\mathcal{U}) = K_1 - K_2 + K_3 - \ldots + (-1)^{p-1} K_p \tag{5.1}$$

Algorithm 4 Consider the expression for $t@x$

$$e(t@x) = \bigvee_{i=1}^{p} e(t@v_i)$$

This expression is in disjunctive form, (also called "sum-of-products") where each $e(t@v_i)$ is a conjunct. Convert this expression into *disjunctive normal form* (as in Algorithm 2), i.e.

$$e(t@x) = \bigvee_{i=1}^{q} e(t@w_i)$$

where each $e(t@w_i)$ is a conjunct in which all variables $f_1, \ldots, f_k$ (maybe negated) appear. Note that the disjunctive form of a Boolean expression is unique (up to a permutation of conjuncts). Then

$$re(t) = re(t@w_1) + \ldots + re(t@w_q) \tag{5.2}$$

Example 19 Let r be an extended relation whose extended scheme is $R = \{$ *Stud-id*, *Stud-Address*, $S_{A_0}, S_{A_1}, S_{A_2}\}$ and let t be the pure tuple (John, St. Paul). Let $t@\mathcal{U} \in r$, where $\mathcal{U} = \{(001, 101, 001), (100, 110, 110)\}$ In this case, $x = \{110, 101\}$, where $110 = \bigwedge vectors((001, 101, 001))$ and $101 = vectors((100, 110, 110))$. Moreover, let the reliability of information sources be 60%, 80%, and 90%, respectively. We calculate the reliability of t using the two algorithms. Using Algorithm 3:

$K_1 = re(t@(101) + re(t@(110) = 0.54 + 0.48 = 1.02$

$K_2 = re(t@(101) \wedge (110) = re(t@(111)) = 0.432$

$re(t) = K_1 - K_2 = 1.02 - 0.432 = 0.588$

Using Algorithm 4, first we convert $e(x)$ to disjunctive normal form:

$$e(t) = (f_1 \wedge f_3) \vee (f_1 \wedge f_2) = (f_1 \wedge f_2 \wedge f_3) \vee (f_1 \wedge \neg f_2 \wedge f_3) \vee (f_1 \wedge f_2 \wedge \neg f_3)$$

obtaining $t@(\{(111), (1-11), (11-1)\})$. Now

$$re(t) = re(t@(111)) + re(t@(1-11)) + re(t@(11-1))$$

$$re(t) = 0.432 + 0.108 + 0.048 = 0.588$$

5.2 Probabilistic Approach (Proof of Correctness)

Reliability calculation algorithms (1, 2) were proven correct by Sadri in [19] for the case of IST model. In that approach the semantics of the alternate world model was used to prove the correctness of the Reliability calculation algorithms. In this section, a similar approach is used to prove the correctness of the reliability calculation algorithms. We will justify that the reliability calculation algorithms in our case are correct using the semantics of the alternate worlds model.

In the coming sections, we assume $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U_1, \ldots, U_p\}$ is associated with pure tuple t in r. As usual, let $x = \{v \mid v_i = \bigwedge vectors(U_i), i = 1, \ldots, p\}$ be a dependent set of source vectors.

Definition 32 The reliability of correctness of an information source s_i is called its *reliability* , and is denoted by P_{s_i}.

Recall that the information content of an extended relation r is a function r^*, where, for a given $Q \subseteq S$, $r^*(Q) = \{ \ t' \mid t' = \mathcal{COR}(t, U, Q)$, where $t@U \in r$ and $e(t@u_0) = true$ under truth(Q). } However, in this particular case we have:

$r^*(Q) = \{t \mid t@\mathcal{U} \in r$, and $e(t@x) = true$ under truth(Q) }

With each $Q \subseteq S$, and with each $r^*(Q)$, we associate a probability $P(Q)$ as follows:

$$P(Q) = \prod_{s_i \in Q} P_{s_i} \prod_{s_i \in S-Q} (1 - P_{s_i}) \tag{5.3}$$

In this way, we have associated with each (regular) relation $r^*(Q)$ in the alternate world set of r a possibility $P(Q)$ which is the probability that r represents $r^*(Q)$ (when the sources in Q are true and all the other sources are not true). In other words, not only we know r represents $r^*(Q)$; but we also have a quantitative measure of the likelihood that r represents $r^*(Q)$.

Example 20 Consider the following extended relation. Assume the reliabilities of the three sources are as follows: $p_1 = 65\%, P_2 = 70\%$, and $P_3 = 90\%$. Then, the probabilities attached to the regular relations in the alternate world of the given extended relation is, corresponding to the set $2^S = \{\emptyset, \{s_1\}, \{s_2\}, \{s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, \{s_1, s_2, s_3\}\}$, are 0.0105%, 0.0195%, 0.0245%, 0.0945%, 0.0455%, 0.1755%, 0.2205%, 0.4095%, respectively.

Sadri in [18, 19] defined the reliability of t to be the degree to which r represents t. This, in turn, can be equated to the probability that t is present in the alternate world of r. We will make this notion precise below.

Definition 33 The *reliability* of a pure tuple t, represented by an extended relation r (i.e. $t@\mathcal{U} \in r$ for a set of source vectors $x = \{v \mid v_i = \bigwedge vectors(U_i), i = 1, \ldots, p\}$), is

$$re(t) = \sum_{t \in r^*(Q)} P(Q) \tag{5.4}$$

For the simple case, let us assume that $x = \{v\}$. In other words, when $t@U \in r$, we have x is a singleton. Let $v = (a_1, \ldots, a_k)$. The set of information sources S could be partitioned into three sets: $S^+(v), S^-(v)$, and $S^0(v)$ as follows:

$S^+(v) = \{s_i \mid a_i = 1\}$, $S^-(v) = \{s_i \mid a_i = -1\}$, and $S^0(v) = \{s_i \mid a_i = 0\}$.
$S^+(v)$ is the set of information sources contributing positively to a tuple $t@v$; $S^-(v)$ is the set of information sources contributing negatively to $t@v$; and $S^0(v)$ is the set of information sources not contributing to $t@v$.

Lemma 10 If $t@U \in r$, and $x = \{v\}$, where $v = \bigwedge vectors(U)$, then the pure tuple t appears in (regular) relations $r^*(Q)$, obtained from the alternate world of r, if and only if $S^+(v) \subseteq Q \subseteq S^+(v) \cup S^0(v)$. *Proof:* Recall that

$$e(v) = \bigwedge_{s_i \in S^+(v)} f \bigwedge_{s_i \in S^-(v)} \neg f$$

Simply, $e(t@v) = true$ under truth(Q) if and only if $S^+(v) \subseteq Q \subseteq S^+(v) \cup S^0(v)$.

Theorem 24 If $t@U \in r$ ($t@U$ is the only extended tuple in r corresponding to the pure tuple t) then

$$re(t) = \prod_{s_i \in S^+(v)} P_{s_i} \prod_{s_i \in S^-(v)} (1 - P_{s_i}) \tag{5.5}$$

where $re(t)$ is the reliability of t.

Proof: By the definition of the reliability of a tuple (Equation 5.4), and lemma 10, we have:

$$re(t) = \sum_{S^+(v) \subseteq Q \subseteq S^+(v) \cup S^0(v)} P(Q) \tag{5.6}$$

If we replace $P(Q)$ by its expression from equation 5.3, and reduce Equation 5.6 obtain:

$$re(t) = \prod_{s_i \in S^+(v)} P_{s_i} \prod_{s_i \in S^-(v)} (1 - P_{s_i})$$

Let us, now, concentrate on a more general case, where t has a set of source vectors associated with it. Two algorithms for reliability calculation were presented in section 5.1.1 to calculate the reliability of t. We will use Equation 5.4, which is based on the alternate world model, to justify these algorithms.

Lemma 11 Assume $t@\mathcal{U} \in r$, where $\mathcal{U} = \{U_1, \ldots, U_p\}$ and $\forall U \in \mathcal{U}, U = (u_0, \ldots, u_n)$. Let $x = \{v \mid v = \bigwedge vectors(U_i), i = 1, \ldots, p\}$. Then $t \in r^*(Q)$ if and only if $S^+(v_i) \subseteq Q \subseteq S^+(v_i) \cup S^0(v_i)$, for at least one $i, 1 \leq i \leq p$.

Proof: Obvious from Equation 5.3, and the fact from Section 2.5.1

$$e(t@x) = \bigvee_{1=1}^{p} e(t@v_i)$$

Now we can derive an expression, similar to Equation 5.6 for the reliability of a tuple t, where x is the set of source vectors associated with t. Let

$$\mathcal{Q} = \{Q_i \mid S^+(v_i) \subseteq Q_i \subseteq S^+(v_i) \cup S^0(v_i), i = 1, \ldots, p\} \tag{5.7}$$

then

$$re(t) = \sum_{Q \in \mathcal{Q}} P(Q) \tag{5.8}$$

where $re(t@\mathcal{U})$ is the reliability of t when $t@\mathcal{U} \in r$, and $re(t@U)$ is the reliability of t when *only* $t@U_i \in r$. This is because some of $P(Q)$'s in Equation 5.8 may be added more than once when we sum up $re(t@U_i)$'s. This observation led to the reliability calculation algorithm presented earlier which we will summarize below.

5.2.1 Reliability Calculation Algorithms

Before we prove the correctness of algorithms 3 and 4, we first prove the following lemma:

Lemma 12 Let $\mathcal{Q} = \{Q_i \mid S^+(u_i) \subseteq Q_i \subseteq S^+(u_i) \cup S^0(u_i)\}$, $i = 1, 2$. Then

$$\sum_{Q \in \mathcal{Q}_1 \cup \mathcal{Q}_2} P(Q) = \sum_{Q \in \mathcal{Q}_1} P(Q) + \sum_{Q \in \mathcal{Q}_2} P(Q) - \sum_{Q \in \mathcal{Q}_1 \cap \mathcal{Q}_2} P(Q) \tag{5.9}$$

Proof: This lemma is based on the principle of inclusion and exclusion [14]. Intuitively, since, some of the Q's may be repeated in the first and second summation on the right hand side of Equation 5.9, we will deduct from the summation of the first two terms on the right hand side the third summation on the right hand side of Equation 5.9.

This lemma can be generalized to more than two sets. The generalization will alternate adding and subtracting the effect of elements appearing in only one set, in two sets, in three sets, etc ...

Theorem 25 Algorithm 3 correctly computes the reliability of a pure tuple t according to Equation 5.4.

Proof of theorem 25: We obtained Equation 5.1 for the reliability of t using lemma 11, so we will show that the algorithm computes $re(t)$ according to Equation 5.8.

In the light of Lemma 12, all we need to show is the following: Let $\mathcal{Q}_i = \{Q_i \mid S^+(v_i) \subseteq Q_i \subseteq S^+(v_i) \cup S^0(v_i)\}$, $i = 1,2$. Then $\mathcal{Q} = \mathcal{Q}_1 \cap \mathcal{Q}_2$ if and only if $\mathcal{Q}\{Q \mid S^+(v) \subseteq Q \subseteq S^+(v) \cup S^0(v)\}$, where $v = v_1 \wedge v_2$. This follows from Theorem 8. Since $e(t@v_i) = \mathit{true}$ under truth(Q_i) if and only if $Q_i \in \mathcal{Q}_i$, and $e(t@v) = e(t@v_1) \wedge e(t@v_2)$ is true if and only if both $e(t@v_1) = e(t@v_2) = \mathit{true}$, if and only if $Q \in \mathcal{Q}_1 \cap \mathcal{Q}_2$. The generalization to the case of more than two source vectors is straight forward and we omit it here.

Theorem 26 Algorithm 4 correctly computes the reliability of a tuple t according to Equation 5.4.

Proof: The observation to make here is if $e(t@v_i)$ is a conjunct in which all variables $f_1, \ldots, f_k$ (may be negated) appear, then $S^0(v_i) = \emptyset$ and the set $\mathcal{Q}_i = \{Q_i \mid S^+(v_i) \subseteq Q_i \subseteq S^+(v_i)\}$ is a singleton. In other words, once converted to disjunctive normal form, each $t@v_i$ makes $t \in r^*(Q)$ for exactly one Q. Since there are no repetitions in disjunctive normal form, Equation 5.2 correctly computes the reliability of t as specified by Equation 5.8.

Chapter 6

Implementation

Architecture, subsystems/modules ...
Decisions made for the implementation, and justification.
Capabilities of the prototype.

6.1 Introduction

In this chapter we will introduce the environment under which we are implementing our model. Define the capabilities of each tool used in that environment. We will present the architecture of our design, list the functions of the system modules in use, justification of our implementation, and the capabilities of the prototype. Towards the end of this chapter we will point out some future enhancements so that the system will be more and more updated to best serve the needs of users.

6.2 Environment and Tools Needed

In this prototype we will use the following major development tools: INGRES which is a relational database system, Embedded SQL, which is an embedding of the database INGRES/SQL into a procedural programming host language (in this prototype we will use the C language), and the User Interface Motif Under X-windows (UIM/X), which is a graphical users interface that helps in building user interfaces using Motif libraries.

6.2.1 The Intelligent Database (INGRES)

INGRES is a relational database management system that allows any number of users (end-users or application programmers or both) to access any number of relational databases by means of either the SQL ("Structured Query Language"), the QUEL ("Query Language"), or the ESQL Embedded SQL.

6.2.2 Embedded SQL (ESQL)

Embedded SQL (ESQL) is an embedding of the database language SQL into a procedural programming language, known in this context as the host language. The statements in (ESQL) compose a superset of the statements found in interactive INGRES/SQL. *Embedded SQL* allows the users to develop highly-interactive, forms-based applications. These forms-based applications usually include statements that manipulate database, although the users are free to create applications using the forms capability without database access.

The ESQL statements are easily intermixed with the full range of host language statements. Furthermore, host language variables may be used in ESQL statements to represent many of the elements of database and form manipulation, such as database expressions, from names, and so on.

ESQL statements are the same from one host language to another. This is an advantage since if the user is familiar with the ESQL statements he can easily embed the ESQL into the host language he knows best.

ESQL provides the application with full access to INGRES database. All statements available in interactive SQL to manipulate and manage data are also available, generally unchanged, in ESQL. ESQL can provide the application with the following:

- *Manipulate data structures:* ESQL statements allow the programmer to create, destroy and modify the underlying tables dynamically. The programmer can also use ESQL to create views on his database.

- *Manipulate data:* The data manipulation statements **select, insert, update,** and **delete** can be used respectively to retrieve, append, update and delete data from the database tables.

- *Manage group of statements as transaction:* By utilizing INGRES's transaction management statements, the programmer can process a group of database or host language statements as a single transaction. Transaction management includes the ability to abort a transaction, either in whole or in part.

- *Perform a host of other database management functions:* Among the other INGRES/SQL features that the programmer can embed in his program are the ability to create permits and integrities on tables, set INGRES run-time options and copy data between tables and files. Because ESQL statements are nearly identical to their interactive counterparts, the programmer can easily prototype the database aspects of his application *via* interactive SQL.

ESQL provides the programmer a uniquely powerful set of forms-based application statements that allow the programmer to utilize the INGRES *Forms Run-time System* (FRS) to create a limitless assortment of forms-based applications. The programmer can create the form himself through INGRES/FORMS: Visual-Forms-Editor (VIFRED). The forms can contain a mixture of forms and *fields*, into which data is entered, either by the application or the users, and *trim*, which serves as explanatory material. Once the form is created, ESQL/FORMS statements enable the programmer to manage the created forms. The functions of ESQL includes:

- *Display of forms.* The programmer can specify in his application that a particular form should be displayed. The user can also specify the movement from one form to another within a single application

- *Transferring of data into and out of a form.* The user can move data from database into a form and, conversely, from the form into the database.

- *Operation Specification.* The user can specify operations to perform on data in the form. The operations can use any of the capabilities of the ESQL and the host language. As part of the operation, the user can specify several methods by which the application program can activate the operations -for example by choosing a menu item, by pressing a control or function key or by leaving a particular field in the form.

- *Miscellaneous operations.* ESQL/FORMS statements are available to validate data, provide messages and help screens to the users, clear the screen and control the user's ability to enter information in the form, in addition to numerous other useful capabilities.

One of the outstanding features of the INGRES Forms Run-Time System is the *table field.* A table field is a form field that can display many columns and rows of data at a time. Its tabular structure mirrors the nature of the relational database and is suited to the display of large amount of data from a database. A form containing both fields and simple fields presents a natural medium for the handling of master-detail type applications.

The table field ordinary acts as a window to an underlying set of data usually data retrieved from the database. This data set contain many more rows of data than can be displayed at one time in the table field. Table field operations are available to manage the display of data set, so that a user may scroll to rows that do not initially appear in the window. The user can add, delete or update rows in the data set, with the FRS handling the bookkeeping involved in tracking the changes until such time as the changes are merged back into the database.

Status information is available to the programmer at all times by means of the SQL communication area (SQLCA) and the **inquire_frs** statement of ESQL/FORMS. The The SQLCA is a data structure included in the program. It contains information concerning the status of the program itself, including the effect of data manipulation statements on the database. Statements in the application program can reference data in the SQLCA to determine appropriate action. Information regarding the status of the FRS and of the various form objects is easily accessed by means of the **inquire_frs** statements.

The Embedded SQL preprocessor converts the Embedded SQL statements in a program into the host language source code statements. These statements call a run-time library that provides the interface to INGRES. The host language statements originally in the program are passed through the preprocessor without being altered. Once the program has been preprocessed, it must be compiled and linked in the usual fashion for the host language.

6.2.3 UIM/X

Well designed applications with iconic user interfaces have many advantages: they are easy to learn, easy to use, and can provide users with the support needed to efficiently work with the application.

UIM/X is a comprehensive, second-generation Graphical User Interface (GUI) Builder. It enhances programmer productivity by enabling software developers to interactively create, modify, test, and generate code for the user interface portion of their applications. UIM/X can also create graphical interfaces for existing keyboard-oriented application, with no need to modify or restructure the underlying application.

The UIM/X is fully-integrated with the OSF/Motif toolkit. Developers use a What You See Is What You Get (WYSIWYG) editor to graphically choose from any of the OSF/Motif widgets (such as pushbuttons, scrollbars, popup menus, and so on) and draw their interfaces.

The UIM/X also contains a built-in C interpreter. From within UIM/X, developers can create the C-code link between the user interface and their application. UIM/X provides its own library functions for users. Moreover, UIM/X generates high-performance, error free C or C++ code, as well as a customizable main program and makefile.

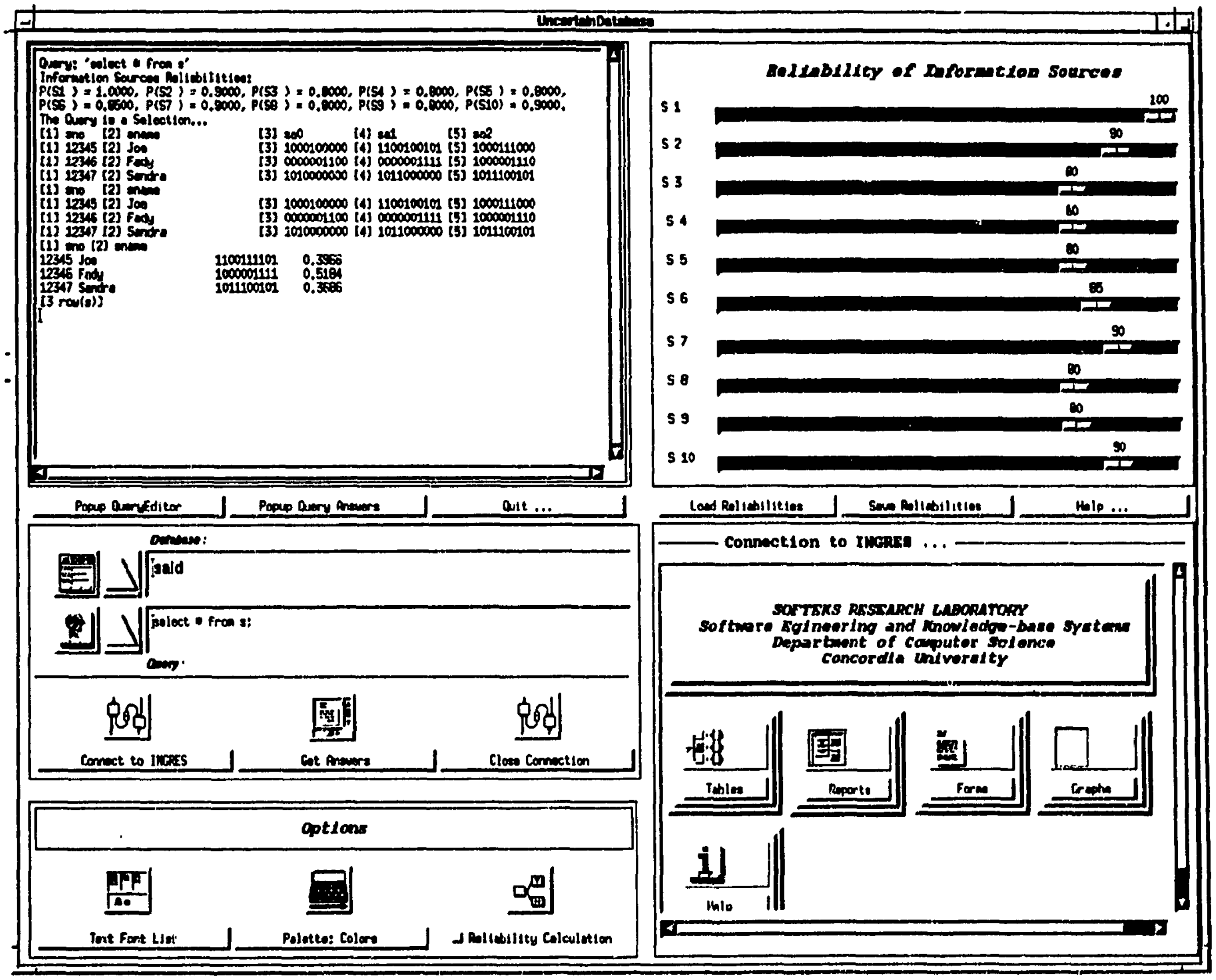
6.3 System Architecture

In this section we will quickly review the system architectural design. We will consider an INGRES database system with different extended relations (tables) as we defined in earlier chapters. In other words, we assume that the user created his/here tables under INGRES before using our prototype.

The user interface part is developed using UIM/X and it is capable of reading the database name and the query typed by the user. The reliability of the information sources could be changed by moving the horizontal scales in the user interface left and right adjusting the reliability of the information sources. For each query typed by the user the system will show the query and the reliability of the information sources before listing the answers to queries in the scrolled text of the interface.

Referring to Figure 6.1, the user should enter the name of the database ('said' in this case) and the query. The query syntax is the same as the INGRES SQL syntax. Now, Given that the user entered the query, the button 'Connect to INGRES' or the icon above that button should be pressed. This establishes a connection to INGRES. The Interface at this point will turn the color of the up arrows to red indicating that the user can not change the database name and the query. At this point, the system expects the user to press the 'Get Answers' pushbutton or the icon above that pushbutton. When the 'Get Answers' pushbutton is pressed the interface will send the arguments to INGRES by running a subprocess that is able to process the query and display the answers in the scrolled text of the user interface.

The subprocess will accept the query and will perform the appropriate query transformation to the query typed by the user. Having transformed the query, the system will send the query to the INGRES Query Processor to bring the query answers to the subprocess. The subprocess is responsibie of manipulating the source vectors associated with the pure tuples and their attribute values. Processing of the source vectors associated with the pure tuples and the attribute values in the answers to queries is based on the correct semantics we presented in earlier chapters. In this prototype we will show the answers as returned by the INGRES Query Processor, the manipulated tuples, as well as the reliability of the each pure tuple as calculated by the system. Remember that the reliability calculation is an optional step that the user(s) will specify.



```
Query: 'select * from s'
Information Sources Reliabilities:
P(S1 ) = 1.0000, P(S2 ) = 0.9000, P(S3 ) = 0.8000, P(S4 ) = 0.8000, P(S5 ) = 0.8000,
P(S6 ) = 0.8500, P(S7 ) = 0.9000, P(S8 ) = 0.8000, P(S9 ) = 0.8000, P(S10) = 0.9000.
The Query is a Selection...
[1] sno   [2] sname               [3] sa0        [4] sa1        [5] sa2
[1] 12345 [2] Joe                 [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady                [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra              [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] sno   [2] sname
[1] 12345 [2] Joe                 [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady                [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra              [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] sno [2] sname
12345 Joe                     1100111101    0.3956
12346 Fady                    1000001111    0.5184
12347 Sandra                  1011100101    0.3586
(3 row(s))
```

Figure 6.1: The Uncertain Database Management System.

From an architectural point of view, we realize from Figure 6.2 how the previous steps in query answering are performed.

- *Step 1*: The query is sent from the User Interface to the Subprocess.
- *Between Step 2 and Step 3*: The subprocess will identify the query by building a structured query template, perform the appropriate query transformation, and send the transformed query to INGRES. Of course, whenever an error (system error) happens, the subprocess will not send the query to INGRES and in this case, the user is prompted with the error message in the scrolled text area of the interface. If there were no errors detected, the subprocess will send the query to INGRES.
- *Step 2*: The transformed query is sent to INGRES Query Processor.
- *Step 3*: The INGRES Query Processor will calculate the answers to the query and returns them to the subprocess.
- *Step 4*: The answers to users queries are returned back to the subprocess that uses the embedded SQL to communicate with INGRES.
- *Between Step 4 and Step 5*: The subprocess will print the answers to the interface without manipulation, perform the correct manipulation to the source vectors associated with the pure tuples and their attribute values and send the results to the scrolled text of the user interface. As a last step in the subprocess, reliability of answers to users queries is performed based on the status of the toggle button in the interface window.

6.4 Subsystems and Models

In this section we will explicitly clarify two subsystems: the User Interface and the Subprocess.

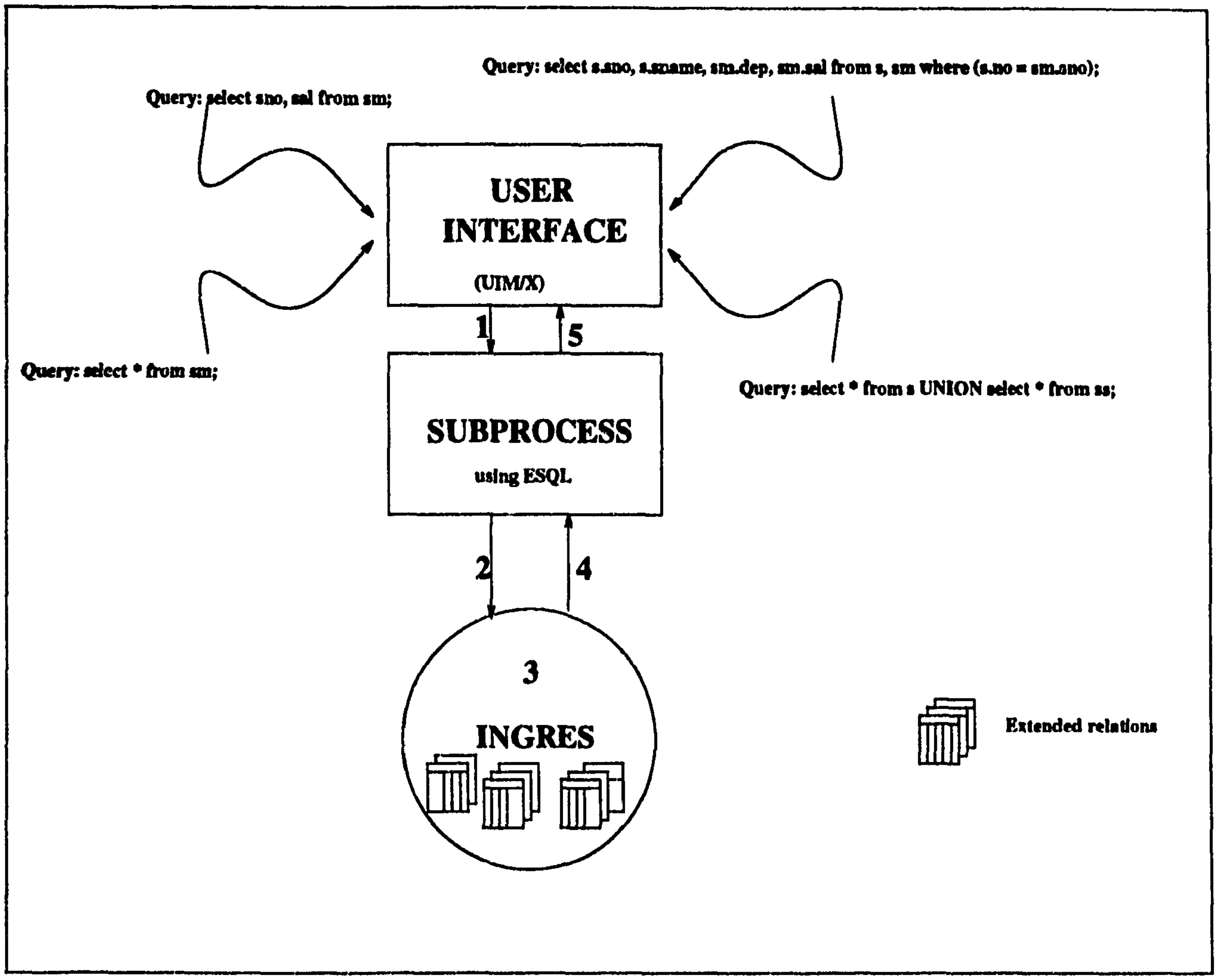

Query: select s.sno, s.sname, sm.dep, sm.sal from s, sm where (s.no = sm.sno);
Query: select sno, sal from sm;
USER INTERFACE
(UIM/X)
Query: select * from sm;
Query: select * from s UNION select * from ss;
1
5
SUBPROCESS
using ESQL
2
4
3
INGRES
Extended relations

Figure 6.2: System Architecture.

6.4.1 The User Interface

UIM/X is used to develop the user interface which consists of one main file called 'QE.c" that uses other three sub models "QueryEditor.c", "ScrolledText.c", and "UncertainDatabase.c". The last model is the interface shown in Figure 6.1. The code of these models is included in the Appendix chapter.

"QE.c" declares the global variables and functions as well as creating the interface of Figure 6.1. The "UncertainDatabase.c" model will communicate between the user(s) and other models. For example, this interface will accept the query and give control to the subprocess to execute the query, popup the "ScrolledText.c" and "QueryEditor.c" interfaces. 'QueryEditor.c" is not implemented in this prototype, however, the interface exists as a code and does not perform any internal processing. The "QueryEditor.c" interface is capable of displaying the answer to the last query performed in a wider scrolled text window. Any future extensions to this project should enhance the power of these two interfaces.

6.4.2 The Subprocess

During *Step 2* the subprocess is given control by the interface by passing two arguments: the database name and the query. The subprocess uses ESQL statements to allocate a buffer that stores the query typed by the user. As a second step, the subprocess creates a *Structured QueryTemplate* as pointed out in Figure 6.3. The Structured Query Template is a data structure that stores the attribute names, number of attributes required, number of tables needed, table names, the selection condition if any, and the query type. Moreover, the subprocess will transform the typed query to a query that includes all attributes of all mentioned tables in the query, the selection condition will not be affected. At this point, the subprocess uses ESQL statements that prepares and describes the transformed query that should be sent to the INGRES query processor.

The INGRES query processor will act on the query and produce a set of answers to be sent to the subprocess. Note that the subprocess and INGRES will communicate by means of the SQLDA and the SQLCA as mentioned earlier in this chapter. When the answers are sent from INGRES to the subprocess (*Step 4*), the subprocess branches to the appropriate procedure to manipulate the answers given by INGRES. The branch is based on the query type. For example, if the query is a "Cartesian Product" the appropriate manipulation method for source vectors is applied which is based on the semantics we proved in earlier chapters. The structured query template is used during the manipulation to identify the needed attribute values and to test the selection condition, when it is present, of the typed query. As noted before, the answers are sent to the scrolled text area of the user interface.

6.5 Capabilities of the Prototype

This prototype is capable of answering queries that does not require more than one extended relational algebra operation. In other words, a query that requires only a Join is best served by this prototype. The prototype assumes a constant set of information sources (10 information sources). The extended relations should be created by the user using the "tables" utility of INGRES. Algorithm 3 is used to calculate the reliability of the pure tuples in the answers to queries.

One important aspect that makes our model important is the capability of allowing the same pure tuple to duplicate in an extended relation. Consider, for example, the pure tuple t that appears in the extended relation created by the user using the "tables" utility of INGRES in Figure 6.4.

The pure tuple t shows that the system is able to combine multiple supports of the same evidence to find the reliability of the evidence itself based on the reliability of the information sources who contributed to that evidence. This is a very important aspect in many application domains like medicine and military systems.

A more interesting aspect to look at is the condition under which our new model is equivalent to the IST model introduced by Sadri [18, 19]. In other words, as Figure 6.5 indicates, our new model is equivalent to the IST when the source vector associated with each pure tuple in the extended relation is the same for all attribute values of

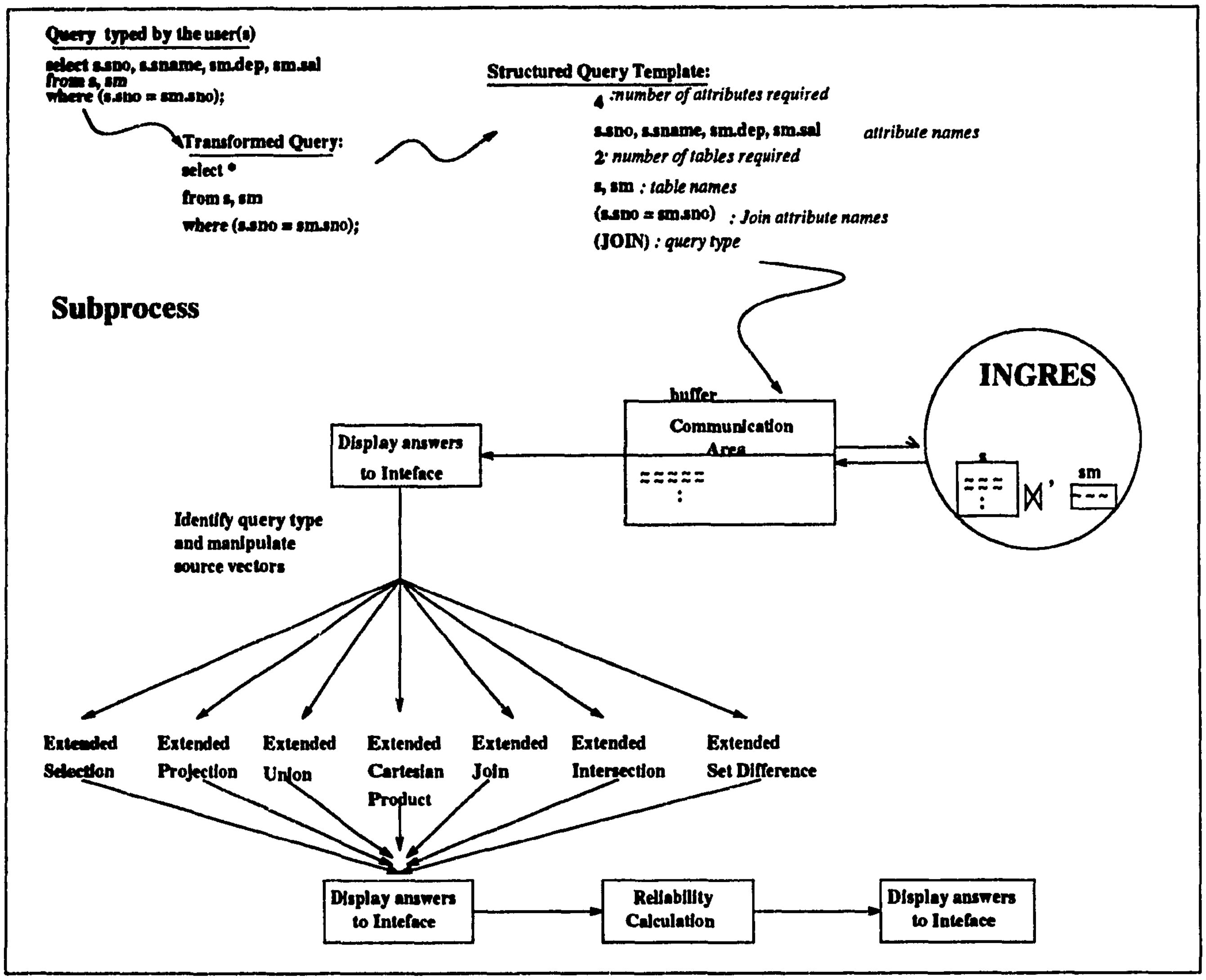
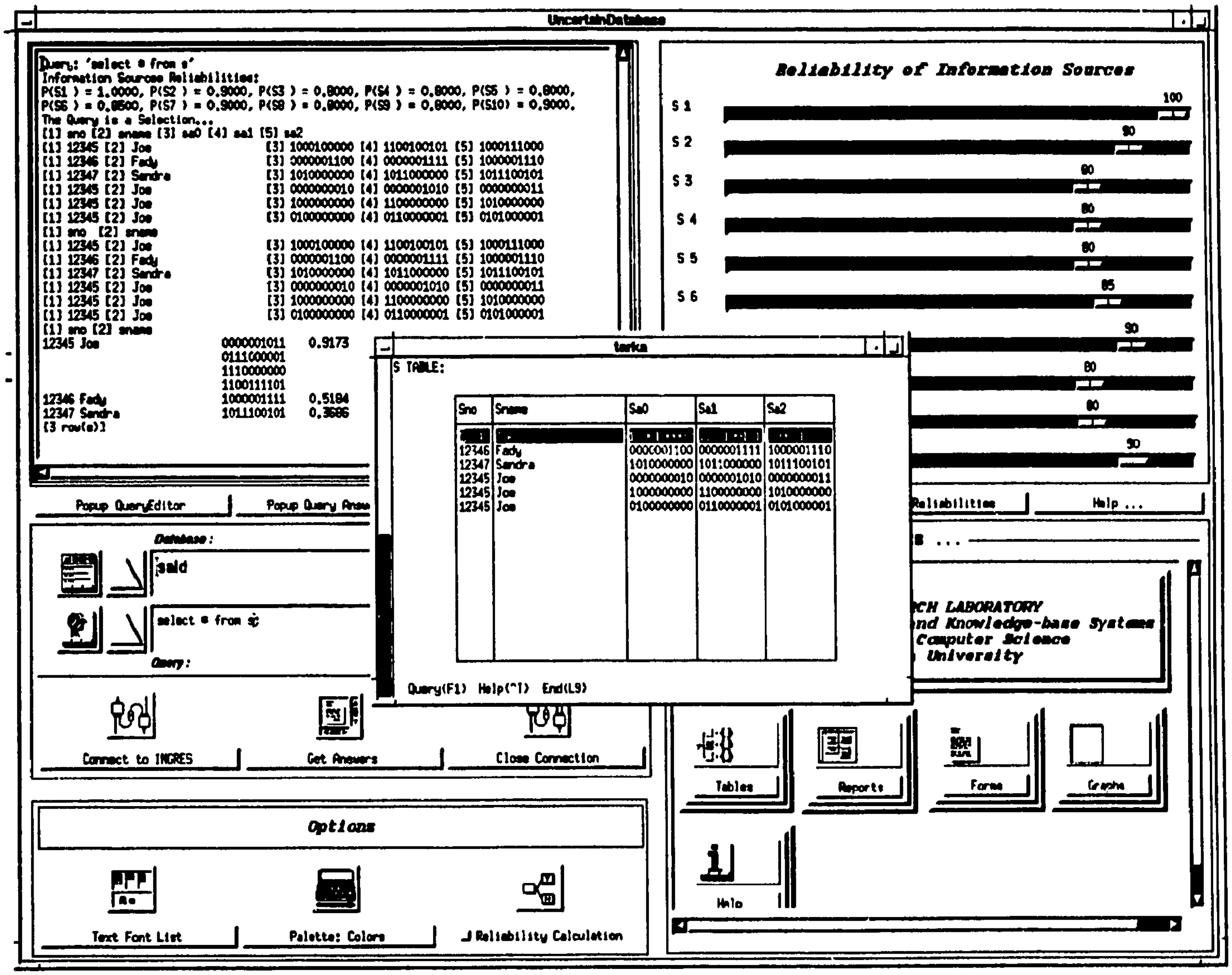

Query typed by the user(s)
select s.sno, s.sname, sm.dep, sm.sal
from s, sm
where (s.sno = sm.sno);
Transformed Query:
select *
from s, sm
where (s.sno = sm.sno);
Structured Query Template:
4 :number of attributes required
s.sno, s.sname, sm.dep, sm.sal
attribute names
2: number of tables required
s, sm : table names
(s.sno = sm.sno) : Join attribute names
(JOIN) : query type
Subprocess
INGRES
s
sm
buffer
Communication Area
Display answers to Inteface
Identify query type and manipulate source vectors
Extended Selection
Extended Projection
Extended Union
Extended Cartesian Product
Extended Join
Extended Intersection
Extended Set Difference
Display answers to Inteface
Reliability Calculation
Display answers to Inteface

Figure 6.3: Subprocess Architecture.



```
Query: 'select * from s'
Information Sources Reliabilities:
P(S1 ) = 1.0000, P(S2 ) = 0.9000, P(S3 ) = 0.8000, P(S4 ) = 0.8000, P(S5 ) = 0.8000,
P(S6 ) = 0.8500, P(S7 ) = 0.9000, P(S8 ) = 0.8000, P(S9 ) = 0.8000, P(S10) = 0.9000.
The Query is a Selection...
[1] sno [2] sname [3] sa0 [4] sa1 [5] sa2
[1] 12345 [2] Joe              [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady             [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra           [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] 12345 [2] Joe              [3] 0000000010 [4] 0000001010 [5] 0000000011
[1] 12345 [2] Joe              [3] 1000000000 [4] 1100000000 [5] 1010000000
[1] 12345 [2] Joe              [3] 0100000000 [4] 0110000001 [5] 0101000001
[1] sno  [2] sname
[1] 12345 [2] Joe              [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady             [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra           [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] 12345 [2] Joe              [3] 0000000010 [4] 0000001010 [5] 0000000011
[1] 12345 [2] Joe              [3] 1000000000 [4] 1100000000 [5] 1010000000
[1] 12345 [2] Joe              [3] 0100000000 [4] 0110000001 [5] 0101000001
[1] sno [2] sname
12345 Joe                  0000001011    0.9173
                           0111000001
                           1110000000
                           1100111101
12346 Fady                 1000001111    0.5184
12347 Sandra               1011100101    0.3686
(3 row(s))
```

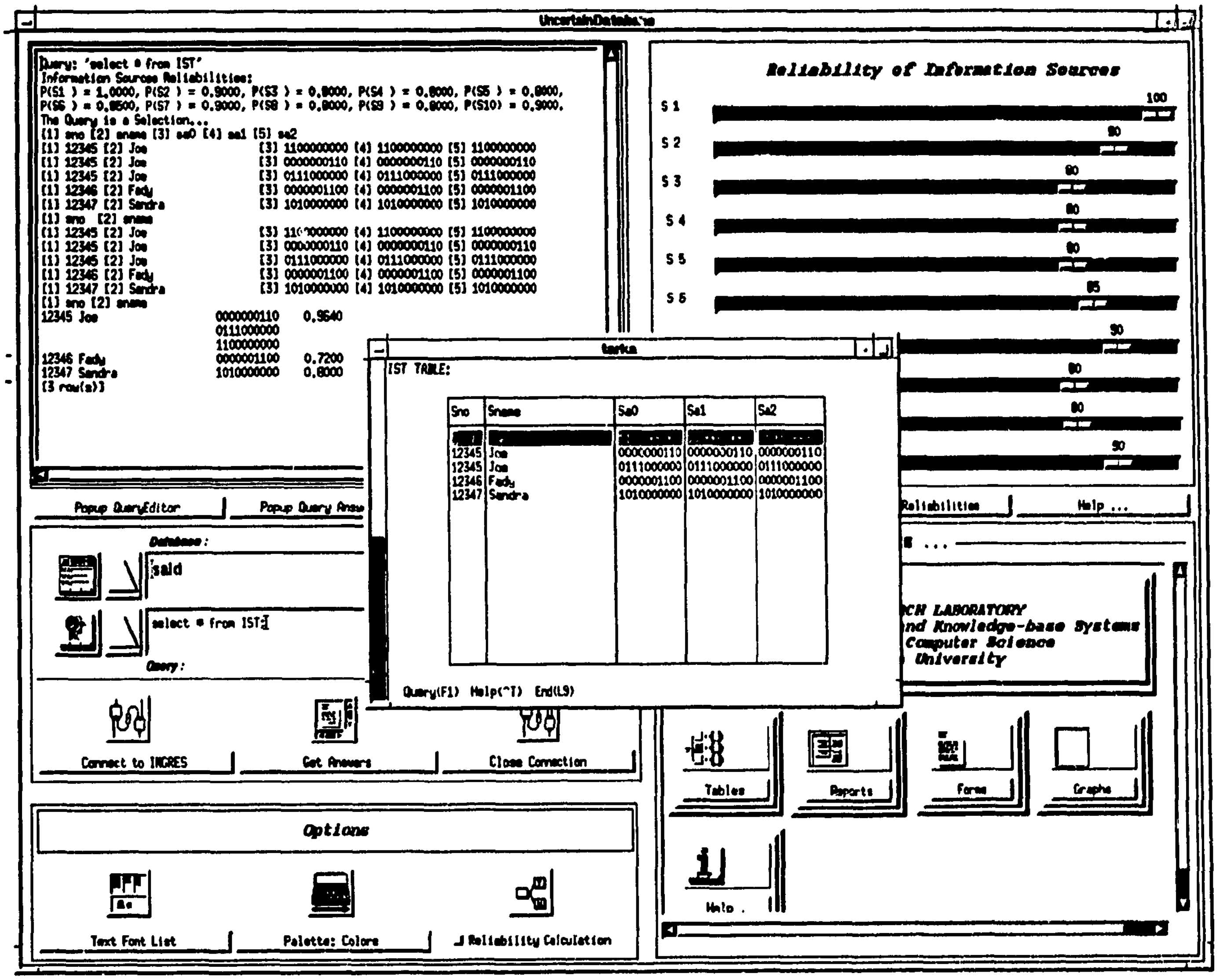
S TABLE:

Sno	Sname	Sa0	Sa1	Sa2
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
12346	Fady	0000001100	0000001111	1000001110
12347	Sandra	1010000000	1011000000	1011100101
12345	Joe	0000000010	0000001010	0000000011
12345	Joe	1000000000	1100000000	1010000000
12345	Joe	0100000000	0110000001	0101000001

Figure 6.4: Reliability of Multiple Source vectors Supporting The Same Evidence.

the same pure tuple. In such situations the IST model is preferable to our new model since our new model requires more space to be stored and more complicated source vector manipulation to get the reliabilities of the tuples in the answers to queries.

Let us now consider the case when our new model is equivalent to the regular relational model. Consider the situation where the information sources are all 100% reliable. In this case, as in Figure 6.6, we have three tuples in the result of answers to queries and the reliability of each is 100% which is always the case in the regular relational model.

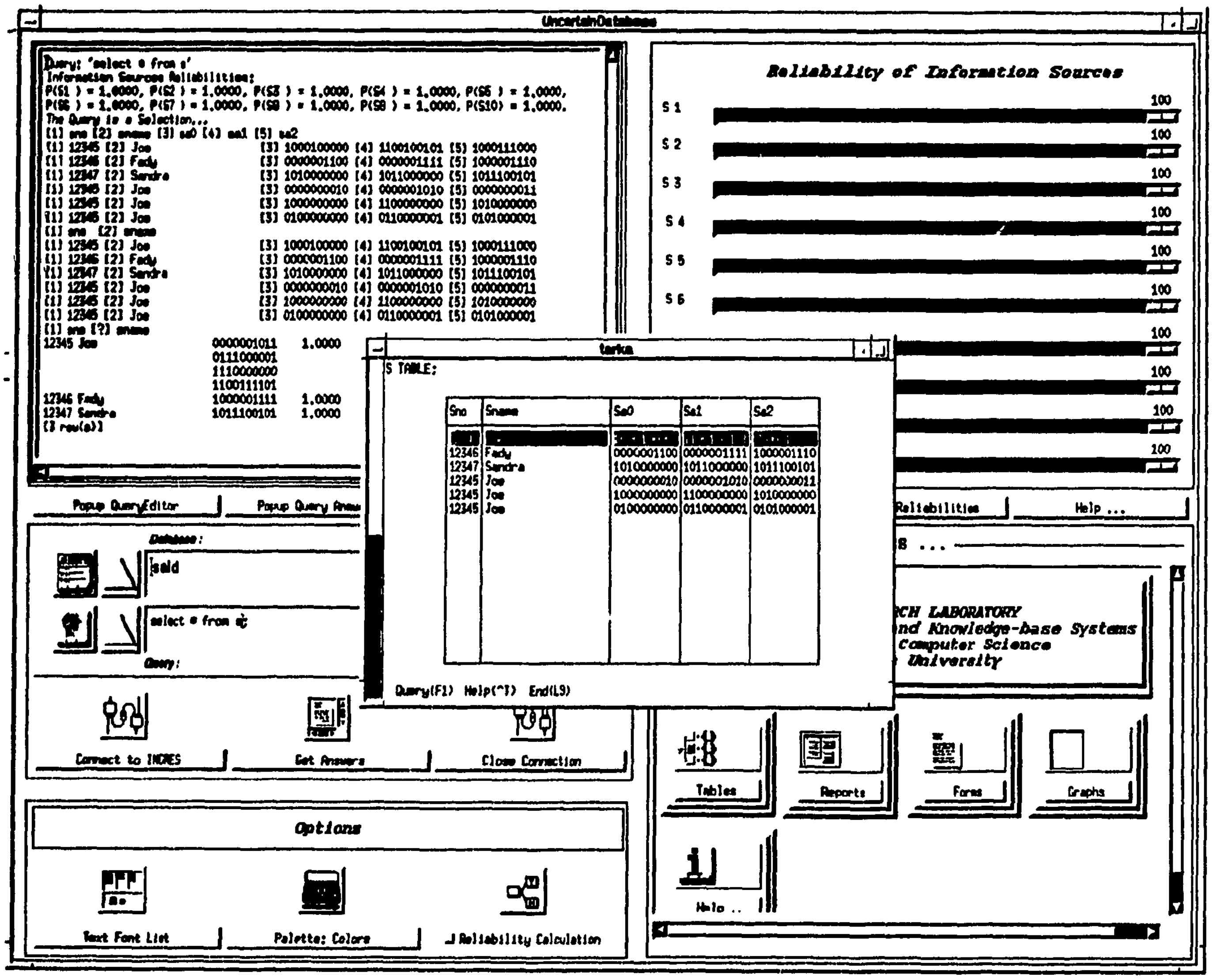


```
Query: 'select * from IST'
Information Sources Reliabilities:
P(S1 ) = 1.0000, P(S2 ) = 0.9000, P(S3 ) = 0.8000, P(S4 ) = 0.8000, P(S5 ) = 0.8000,
P(S6 ) = 0.8500, P(S7 ) = 0.9000, P(S8 ) = 0.8000, P(S9 ) = 0.8000, P(S10) = 0.9000.
The Query is a Selection...
[1] sno [2] sname [3] sa0 [4] sa1 [5] sa2
[1] 12345 [2] Joe              [3] 1100000000 [4] 1100000000 [5] 1100000000
[1] 12345 [2] Joe              [3] 0000000110 [4] 0000000110 [5] 0000000110
[1] 12345 [2] Joe              [3] 0111000000 [4] 0111000000 [5] 0111000000
[1] 12346 [2] Fady             [3] 0000001100 [4] 0000001100 [5] 0000001100
[1] 12347 [2] Sandra           [3] 1010000000 [4] 1010000000 [5] 1010000000
[1] sno  [2] sname
[1] 12345 [2] Joe              [3] 1100000000 [4] 1100000000 [5] 1100000000
[1] 12345 [2] Joe              [3] 0000000110 [4] 0000000110 [5] 0000000110
[1] 12345 [2] Joe              [3] 0111000000 [4] 0111000000 [5] 0111000000
[1] 12346 [2] Fady             [3] 0000001100 [4] 0000001100 [5] 0000001100
[1] 12347 [2] Sandra           [3] 1010000000 [4] 1010000000 [5] 1010000000
[1] sno [2] sname
12345 Joe                0000000110    0.9640
                         0111000000
                         1100000000
12346 Fady               0000001100    0.7200
12347 Sandra             1010000000    0.8000
(3 row(s))
```

IST TABLE:

Sno	Sname	Sa0	Sa1	Sa2
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
12345	Joe	0000000110	0000000110	0000000110
12345	Joe	0111000000	0111000000	0111000000
12346	Fady	0000001100	0000001100	0000001100
12347	Sandra	1010000000	1010000000	1010000000

Figure 6.5: Our Model and IST model (in this case they are the equivalent).



```
Query: 'select * from s'
Information Sources Reliabilities:
P(S1 ) = 1.0000, P(S2 ) = 1.0000, P(S3 ) = 1.0000, P(S4 ) = 1.0000, P(S5 ) = 1.0000,
P(S6 ) = 1.0000, P(S7 ) = 1.0000, P(S8 ) = 1.0000, P(S9 ) = 1.0000, P(S10) = 1.0000.
The Query is a Selection...
[1] sno [2] sname [3] sa0 [4] sa1 [5] sa2
[1] 12345 [2] Joe               [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady              [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra            [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] 12345 [2] Joe               [3] 0000000010 [4] 0000001010 [5] 0000000011
[1] 12345 [2] Joe               [3] 1000000000 [4] 1100000000 [5] 1010000000
[1] 12345 [2] Joe               [3] 0100000000 [4] 0110000001 [5] 0101000001
[1] sno  [2] sname
[1] 12345 [2] Joe               [3] 1000100000 [4] 1100100101 [5] 1000111000
[1] 12346 [2] Fady              [3] 0000001100 [4] 0000001111 [5] 1000001110
[1] 12347 [2] Sandra            [3] 1010000000 [4] 1011000000 [5] 1011100101
[1] 12345 [2] Joe               [3] 0000000010 [4] 0000001010 [5] 0000000011
[1] 12345 [2] Joe               [3] 1000000000 [4] 1100000000 [5] 1010000000
[1] 12345 [2] Joe               [3] 0100000000 [4] 0110000001 [5] 0101000001
[1] sno [?] sname
12345 Joe              0000001011    1.0000
                       0111000001
                       1110000000
                       1100111101
12346 Fady             1000001111    1.0000
12347 Sandra           1011100101    1.0000
(3 row(s))
```

S TABLE:

Sno	Sname	Sa0	Sa1	Sa2
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
12346	Fady	0000001100	0000001111	1000001110
12347	Sandra	1010000000	1011000000	1011100101
12345	Joe	0000000010	0000001010	0000000011
12345	Joe	1000000000	1100000000	1010000000
12345	Joe	0100000000	0110000001	0101000001

Query(F1) Help(^T) End(L9)

Figure 6.6: Our Model and the Regular Relational Model (in this case they are equivalent).

Chapter 7

Conclusion and Future Directions

In this work we extended the IST model presented by Sadri in [18, 19]. In this new framework, the information sources or observers that provided the extended relational database with the information can contribute to that information with more flexibility. Any source vector can contribute to any attribute value in any tuple or to the whole tuple. In pursuing this line, we concentrated on the extended relational algebra operations and extended the concept of the "Alternate Worlds Model", that Sadri presented in [19], to prove the correctness of the extended relational algebra operations. Sadri in [18, 19] introduced and proved the correctness of two important reliability calculation algorithms that calculate the reliabilities of answers to queries -done as a last optional step after query processing. In this regard, we extended these reliability calculation algorithms so that they fit into our new model and proved that the extended algorithms are correct using the semantics of the "Alternate World Model".

7.1 IST and Our New Model

IST introduced by Sadri is a clean and effective model to handle uncertainty in relational databases. In Chapter 1 section 2.5, we presented the IST model and presented the extended relational algebra operations that are based on a clean semantics which is the alternate world model. Dependencies and independencies of information is handled, manipulation of conflicting information, query processing, and reliability calculations are all topics of high importance to any application domain that gathers information from independent sources or observers and records the collected information in an uncertain database implemented using the IST model.

If we take a closer look at the IST, we realize that the IST can capture a situation where sources are contributing to the whole tuple as one unit. Consider for example the following situation. Information sources s_1, s_3, and s_4 informs the uncertain database system that they are sure that "*Mike*" is living at "5453 *Harvard Street*" and his phone number is '534 – 3234". IST perfectly models this situation and stores this information in an extended relation r as:

$$(Mike,\ 5453\ Harvard\,Street,\ 534\ - 3234)@(1011) \in r,$$

where (1011) is the source vector associated with the pure tuple

$$t = (Mike,\ 5453\ Harvard\,Street,\ 534\ - 3234)$$

and indicates that the sources s_1, s_3, and s_4 are confirming positively to t.

Let us now consider a different situation where source s_1 confirms all the information presented in the previous tuple, i.e. s_1 confirms that "*Mike*" is living at "5453 *Harvard Street*" and his phone number is '534 – 3234". Let us assume that source s_2 do not agree with the information that source s_1 is providing, i.e., source s_2 denies the fact that "*Mike*" is living at "5453 *Harvard Street*" and his phone number is '534 – 3234". We will, also, assume that source s_3 agrees that the phone number of "*Mike*" is '534 – 3234" but do not know anything about his address. Moreover assume that the information source s_4 did not know anything about the previous fact.

The IST model can not capture this situation. Our new model is able to capture such a situation and store it in the extended relational database system as follows:

$$(Mike,\ 5453\ Harvard\,Street,\ 534 - 3234)@(1-100, 1-110, 1-100, 1-110) \in r,$$

where $u_0 = 1 - 100$ is the source vector associated with the pure tuple t, $u_1 = 1 - 110$ is the source vector associated with the attribute value "Mike", $u_2 = 1 - 100$ is the source vector associated with the attribute value "*5453 Harvard Street*", $u_3 = 1-110$ is the source vector associated with the attribute value "534-3234".

Note that both situations, now, could be stored into the new model, which is an extension of the IST model, as follows:

$$(Mike, 5453\ Harvard\ Street, 534-3234)@(1011, 1011, 1011, 1011) \in r,$$

$$(Mike, 5453\ Harvard\ Street, 534-3234)@(1-10, 1-11, 1-10, 1-11) \in r,$$

In this sense we realize a very important observation of our new model under which our new semantic is equivalent to that introduced by Sadri [18, 19]. This observation states that if all the source vectors associated with the attribute values of any pure tuple t in any extended relation r and the source vector associated with the pure tuple t are equal, then our semantics is equivalent to the semantics introduced by Sadri [18, 19]. In other words, if we have a model where for any pure tuple t $u_0 = u_1 = \ldots = u_n$ in any r, where n is the number of regular attributes in the extended relational scheme R of r as in section 3.1, then the semantics of our new model is equivalent to the semantics of the IST model. In this context, the extended relational algebra operations of section 3.3 are equivalent to the extended relational algebra operations of the IST. Moreover, the reliability calculation algorithms are also equivalent.

Note that when the previous observation is true, then our new model is less preferable to the IST because of space consideration issues.

7.2 Conclusion and Future Work

Our future work is to consider a model where the set of information sources is dynamic and discuss issues arising from introducing functional dependencies and aggregate operations, to our new model. As for this model, we will also look at an important aspect in query answering that provides partial information together with their reliabilities as answers to queries. In such a work we need to define new algorithms that are able to capture this concept. To enhance the power of the user interface in our framework, we are looking forward to extend the syntax of query answering by letting the user specify probabilistic arguments as part of the query syntax. In this sense, the query processor will not only bring answers associated with their reliabilities, but will use probabilistic theories based on our semantics to best serve queries. Of course, in any of these future extensions a complete and sound semantics is always required.

Insertions, Deletions, and other utilities that should be present in an extended relational database will constitute an important part of our research direction.

Another area of interest is to extend the work done by Lakshmanan and Sadri [11]. In this regard, as done in [22] we will introduce uncertainty not only to facts and rules as done in [11], but we will extend this concept to incorporate uncertainty to the constants presented by the facts to introduce a language for uncertain deductive databases. Based on the new language, we will extended the Herbrand Models as an extended interpretation for the new language, look for an extended immediate consequence operator to calculate the intended model of an uncertain database, define the least fixed point model and prove that the least fixed point model intersects with the extended immediate consequence operator.

Another interesting and promising area for future work is the uncertainty that we will introduce in Object Oriented Databases.

Evidential independence should be studied more carefully. Kifer and Li [10] introduced two kinds of evidential independence. A future work could look into other types of evidential independence and apply them to our model and study the behavior of our system under each type of evidential independence.

As an implementation we are interested in implementing our models in the areas of Medicine, Military, Pattern Recognition, Object Recognition, Artificial intelligence and Expert Systems.

Bibliography

[1] K. R. Apt, H. Blair and QA. Walker, "Towards a Theory of Deductive Knowledge". in *Foundations of Deductive and Logic Programming*, J. Minker, (ed.), Morgan-Kaufmann, 1988,89-148.

[2] J. F. Baldwin, "Evidential Support Logic Programming", *Fuzzy Sets and System*, 24, (1987), 1-26.

[3] J. F. Baldwin and M. R. M. Monk, "Evidence Theory, Fuzzy Logic and Logic", ITRC Tech. Rep. # 109, University of Bristol, UK, 1987.

[4] D. Barbara, H. Garcia-Molina, and D. Porter, "The Management of Probabilistic Data", *IEEE Transactions on Knowledge and Data Engineering*, Vol., 4, No. 5, October 1992. 487-502.

[5] E. F. Codd "A Relational Model for large shared data banks", *Commun. ACM*, vol 13, no. 6, pp. 377-387, June 1970.

[6] C. Date, *An Introduction to Database Systems, Vol.* I, Reading, MA: Addison-Wesley, 4th ed., 1986.

[7] M. Gelfond, and Lifschitz, "The Stable Model Semantics for Logics for Logic Programming." Proc. Fifth Logic Programming Symposium, 1988, pp 1070-1080.

[8] D. E. Heckerman and E. J. Horrovits, "On the Expressive Power of Rule-Based Systems for Reasoning with Uncertainty", *Automated Reasoning*, 1987, 121-126.

[9] A. Kaufmann. *Introduction to the Theory of Fuzzy Subsets*, volume I. Academic Press, 1975.

[10] M. Kifer, and A. Li (1988), "On the Semantics of Rule-Based Expert Systems with Uncertainty", 2^{nd} International Conference on Database Theory (LNCS 326), eds. M. Gyssens, J. Paredacus and D. Van Gucht, Springer Verlag, Berlin Heidelberg New York, 102-117.

[11] V. S. Lakshmanan and F. Sadri, "Modeling Uncertainty in Deductive Databases." Technical report, Concordia University, May 1993.

[12] K-C Liu, and R. Sunderraaman, "Indefinite and Maybe Information in Relational Data Bases.", *ACM Transactions on Database Systems*, Vol. 15, No. 1, March 1990, pp 1-39.

[13] K-C. Liu, and R. Sunderraaman, "A Generalized Model for Indefinite and Maybe Information.", *IEEE Transactions on Knowledge and Data Engineering,* Vol. 3, No. 1, March 1991, pp 65-77.

[14] C. L. Lu, "Introduction to Combinatorial Mathematics", McGraw-Hill, 1968.

[15] D. Maier, "The Theory of Relational Database.", Computer Science Press 1993.

[16] T. C. Prsymusinski, "On the Deductive Semantics of Deductive Databases and Logic Programs", in *Foundation of Deductive Databases and Logic Programming,* J. Minker, (ed), Morgan-Kaufmann, Los Alotos, CA, 1988, 193-216.

[17] R. Reiter, "A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values.", ACM, Vol. 33, No. 2, Apr. 1986.

[18] F. Sadri "Reliability of Answers to queries in Relational Database.", *IEEE Transaction on Knowledge and data Engineering,* Vol. 3, No. 2, June 91, pp 245-251.

[19] F. Sadri "Modeling Uncertainty in Databases.", *Proceeding of the 1991 IEEE International Conference on Data Engineering,* pp 122-131.

[20] F. Sadri "Information Source Tracking Method: Efficiency Issues." *Manuscript. December 1992.* Submitted for publication.

[21] F. Sadri "Integrity Constraints in the Information Source Tracking Method." To appear in IEEE Transaction on Knowledge and Data Engineering.

[22] J. N. Said "Intelligent Computations in Managing Uncertainy in Deductive Databases" Technical report Concordia University.

[23] E. Shapiro, "Logic Programs with Uncertainties: A Tool for Implementation Rule-Based Systems", *IJCAI-83,* 1983, 529-532.

[24] V. S. Subrahmanian, "On The Semantics of Quantitative Logic Programs", *IEEE Symposium on Logic Programming,* 1987, 173-182.

[25] A. Van Gelder, K. Roth, and J.S. Schilpf, "Unfounded Sets and Well Founded Semantics for General Logic Programs.", Proceedings of the 1988 ACM Symposium on Principles of Databases Systems, pp 221-230.

[26] M. H. van Emden, "Quantitative Deduction and its Fixpoint Theory", *The Journal of Logic Programming,* 1986, 47-53.

[27] J. D. Ullman, *Principles of Database and Knowledge-Base Systems,* Baltimore, MD: Computer Science, 1988.

[28] L. A. Zadeh. Fuzzy sets. *Information and Control.* pages 338-353, 1956.

Chapter 8

Appendix

Code for The User Interface, ESQL, and Makefiles.

8.1 User Interface code

```
/*****************************************************************************************************
** This is the User Interface main program file QE.c
*****************************************************************************************************/

#include <UxLib.h>
#include <X11/Xlib.h>

/*---------------------------------------------
 * Insert application global declarations here
 *--------------------------------------------*/

#ifdef _NO_PROTO
main(argc,argv)
int  argc;
char  *argv[];
#else
main( int argc, char *argv[])
#endif /* _NO_PROTO */
{
        /*-------------------------------------------------------
        ** Declarations.
        ** The default identifier - mainIface will only be declared
        ** if the interface function is global and of type swidget.
        ** To change the identifier to a different name, modify the
        ** string mainIface in the file "main.dat".  If "mainIface"
        ** is declared, it will be used below where the return value
        ** of  PJ_INTERFACE_FUNCTION_CALL will be assigned to it.
        **------------------------------------------------------*/

   swidget mainIface;

        /*-------------------------------
        ** Interface function declaration
        **------------------------------*/

        swidget create_UncertainDatabase();
        swidget UxParent = NULL;


        /*---------------------
        ** Initialize Program
        **--------------------*/

#ifdef XOPEN_CATALOG
        if (XSupportsLocale()) {
                XtSetLanguageProc(NULL,(XtLanguageProc)NULL,NULL);
        }
#endif

        UxInitCat();
        UxTopLevel = XtAppInitialize(&UxAppContext, "QE",
     NULL, 0, &argc, argv, NULL, NULL, 0);
        UxAppInitialize("QE", &argc, argv);

        /*-----------------------------------------------------
        ** Insert initialization code for your application here
        **----------------------------------------------------*/


        /*---------------------------------------------------------------
        ** Create and popup the first window of the interface.  The
        ** return value can be used in the popdown or destroy functions.
        ** The swidget return value of  PJ_INTERFACE_FUNCTION_CALL will
        ** be assigned to "mainIface" from  PJ_INTERFACE_RETVAL_TYPE.
        **--------------------------------------------------------------*/

        mainIface = create_UncertainDatabase(UxParent);

        UxPopupInterface(mainIface, no_grab);

        /*-------------------------
        ** Enter the event loop
        **------------------------*/

          UxMainLoop();

}


/*******************************************************************************
** UncertainDatabase.c
**
**      Associated Header file: UncertainDatabase.h
*******************************************************************************/

#include <stdio.h>
#include "UxLib.h"
#include "UxLabelG.h"
#include "UxRowCol.h"
#include "UxTogB.h"
#include "UxScText.h"
#include "UxScrW.h"
#include "UxScale.h"
#include "UxPushBG.h"
#include "UxSep.h"
#include "UxArrB.h"
#include "UxText.h"
#include "UxLabel.h"
```

```
#include "UxForm.h"
#include "UxFrame.h"
#include "UxPushB.h"
#include "UxBboard.h"

/*-------------------------------------------------------------------------------------------
**      Includes, Defines, and Global variables from the Declarations Editor:
**------------------------------------------------------------------------------------------*/

#include <UxLib.h>
#include "UxSubproc.h"
handle h;
extern swidget create_QueryEditor();
extern swidget create_ScrolledText();
/* extern swidget create_QueryAnswers(); */

static swidget UncertainDatabase;
static swidget pushButton1;
static swidget pushButton3;
static swidget frame5;
static swidget form3;
static swidget pushButton7;
static swidget pushButton2;
static swidget pushButton4;
static swidget label3;
static swidget label4;
static swidget text1;
static swidget text2;
static swidget arrowButton1;
static swidget arrowButton2;
static swidget separator1;
static swidget pushButtonGadget11;
static swidget pushButtonGadget14;
static swidget pushButtonGadget15;
static swidget pushButtonGadget21;
static swidget pushButtonGadget22;
static swidget pushButton12;
static swidget frame7;
static swidget form5;
static swidget frame8;
static swidget scrolledWindowText2;
static swidget scrolledText2;
static swidget frame9;
static swidget form6;
static swidget scaleH2;
static swidget scaleH10;
static swidget scaleH7;
static swidget scaleH6;
static swidget scaleH8;
static swidget scaleH9;
static swidget scaleH4;
static swidget scaleH5;
static swidget scaleH3;
static swidget scaleH11;
static swidget label9;
static swidget label10;
static swidget label11;
static swidget label12;
static swidget label13;
static swidget label14;
static swidget label15;
static swidget label16;
static swidget label17;
static swidget label18;
static swidget label19;
static swidget pushButton13;
static swidget pushButton14;
static swidget pushButton15;
static swidget frame10;
static swidget form7;
static swidget pushButton16;
static swidget frame12;
static swidget label20;
static swidget pushButton17;
static swidget toggleButton1;
static swidget pushButtonGadget12;
static swidget pushButtonGadget16;
static swidget pushButtonGadget26;
static swidget frame11;
static swidget form8;
static swidget scrolledWindow2;
static swidget form9;
static swidget frame13;
static swidget rowColumn2;
static swidget frame14;
static swidget rowColumn3;
static swidget pushButtonGadget1;
static swidget pushButtonGadget2;
static swidget frame15;
static swidget rowColumn4;
static swidget frame16;
static swidget rowColumn5;
static swidget pushButtonGadget3;
static swidget pushButtonGadget4;
static swidget frame19;
static swidget rowColumn8;
static swidget frame20;
static swidget rowColumn9;
static swidget pushButtonGadget7;
static swidget pushButtonGadget8;
static swidget frame17;
static swidget rowColumn6;
static swidget frame18;
static swidget rowColumn7;
static swidget pushButtonGadget5;
static swidget pushButtonGadget6;
static swidget frame21;
static swidget rowColumn10;
static swidget frame22;
static swidget rowColumn11;
static swidget labelGadget1;
static swidget pushButtonGadget25;
static swidget frame23;
static swidget rowColumn12;
static swidget frame24;
static swidget rowColumn13;
static swidget pushButtonGadget9;
static swidget pushButtonGadget10;
static swidget separator2;
static swidget label21;
static swidget separator3;
static swidget UxParent;
#define CONTEXT_MACRO_ACCESS 1
#include "UncertainDatabase.h"
#undef CONTEXT_MACRO_ACCESS

/*---------------------------------------------------------------------------------------
**      The following are translation tables.
```

```
**-----------------------------------------------------------------------------------------*/

static char *transTable1 = "#augment\n\
<Btn1Down>(2):doubleclick()\n";

/*-----------------------------------------------------------------------------------------
**      The following are Action functions.
**-----------------------------------------------------------------------------------------*/

static void action_doubleclick( UxWidget, UxEvent, UxParams, p_UxNumParams )
        Widget UxWidget;
        XEvent *UxEvent;
        String *UxParams;
        Cardinal *p_UxNumParams;

{
        Cardinal UxNumParams = *p_UxNumParams;
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {printf("double click  using most left button. \n");}
}

/*-----------------------------------------------------------------------------------------
       The following are callback functions.
**-----------------------------------------------------------------------------------------*/
static void activateCB_pushButton1( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        extern swidget QueryEditor;
        create_QueryEditor(NO_PARENT);
        UxPopupInterface(QueryEditor, no_grab);

        }
}

static void activateCB_pushButton3( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /* this should be declared as external */
        /* since this is the interface function of */
        /* the interface to be poped up */
        extern swidget ScrolledText;
        /* create the Contest Structure of the called interface and pass the */
        /* required arguments to it provided that these arguments have the same */
        /* names in the Property Editor of the corresponding Widgets and the Interface*/
        /* function of the popedup interface. */
        /* this instruction ensures that the Context Structure is created but not
        /* displayed */
        create_ScrolledText(NO_PARENT, "Print ...", "Save ...");

        /* this instruction ensures that the Interface is displayed to the screen */
        UxPopupInterface(ScrolledText, no_grab);
        }
}

static void activateCB_pushButton7( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /* h is used by the UIM/X to identify the subprocess.*/
        /* UxCreateSubproc initializes UIM/X to handle the proc.*/
        /* first argument is the name of the application, the second argument */
        /* is the passed argument to the application NULL in this case, the third */
        /* argument is the function pointer to a function that handles output from the */
        /* subprocess. In this case the output is to be sent to a text widget */
        UxPutForeground(arrowButton1, "red");
        UxPutForeground(arrowButton2, "red");
        h = UxCreateSubproc("Terminal", NULL, UxAppendTo);
        /* the second call here identifies the text widget to which the UxAppendTo */
        /* will send the output from the subprocess. First argument is the handle h */
        /* of the subprocess created by the function call UcCreateSubproc function. */
        /* the second argument is the X widget pointer of the text widget */
        if (ERROR == UxSetSubprocClosure (h, UxGetWidget(scrolledText2))) {
      printf("Cannot Set subproc closure \n");
      return;
        }
        /* this function call Runs the Subprocess. Argument are the subprocess handle*/
        /* and the argument that we would like to pass to the subprocess */
        /* the function UxExecuteSubproc() could be used instead of UxRunSubproc(). */
        /* The difference is that the second function force the subprocess to */
        /* and start again in case his process was previously running */
        if (ERROR == UxRunSubproc(h, NULL)) {
      printf("Cannot start the application \n");
      return;
        }
        /* this code could be included in the final code of some interface and the */
        /* code would be executed as soon as tho interface is poped up in the screen */
        /* to do this we will include this code in the final code of that interface */

        UxPutSensitive(pushButton7, "false");

        }
}

static void activateCB_pushButton2( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /* this will release the handle given to the subprocess from the memory */
        /* For applications that might be terminated and restarted again many */
        /* times from the same interface, UxExecuteSubproc() function should be */
        /* used instead. In this case only a call to the UxRunSubproc() is needed */
        /* to restart it */
        UxDeleteSubproc(h);
        UxPutForeground(arrowButton1, "yellow");
        UxPutForeground(arrowButton2, "yellow");

        UxPutSensitive(pushButton4, "true");
        UxPutSensitive(pushButton7, "true");

        }
}

static void activateCB_pushButton4( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        char  s1[128];
        char  s2[128];
        /* if your process requires input from the terminal this is what you should do:*/
        /* create text widgets in the interface and fill in the appropriate values */
        /* for those text widgets as if you are entering these values from the terminal*/
        /* and send them to the process one after the other. This simulates entering */
        /* the values to through the terminal. */
        sprintf(s1, "%s", UxGetText(text1)); /* get the value from widget */
        /* used to send the command to the subprocess. In this case, the string sent */
        /* is formed by the command pop followed by the name of the country, which */
        /* is read from the text widget text1. Output would be handled by UxAppendTo()*/
        /* which would display the string in the Text widget text2 */
        UxSendSubproc(h, s1); /* send the value to the subprocess */
        UxPutForeground(arrowButton1, "green");
        /* after this line accept the other values from the widgets and send them to */
        /* the subprocess to be executed. */

        sprintf(s2, "%s", UxGetText(text2));
        UxSendSubproc(h, s2);
        UxPutForeground(arrowButton2, "green");

        UxPutSensitive(pushButton4, "false");


        }
}

static void activateCB_pushButtonGadget14( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /* h is used by the UIM/X to identify the subprocess.*/
        /* UxCreateSubproc initializes UIM/X to handle the proc.*/
        /* first argument is the name of the application, the second argument */
        /* is the passed argument to the application NULL in this case, the third */
        /* argument is the function pointer to a function that handles output from the */
        /* subprocess. In this case the output is to be sent to a text widget */
        UxPutForeground(arrowButton1, "red");
        UxPutForeground(arrowButton2, "red");
        h = UxCreateSubproc("Terminal", NULL, UxAppendTo);
        /* the second call here identifies the text widget to which the UxAppendTo */
        /* will send the output from the subprocess. First argument is the handle h */
        /* of the subprocess created by the function call UcCreateSubproc function. */
        /* the second argument is the X widget pointer of the text widget */
        if (ERROR == UxSetSubprocClosure (h, UxGetWidget(scrolledText2))) {
      printf("Cannot Set subproc closure \n");
      return;
        }
        /* this function call Runs the Subprocess. Argument are the subprocess handle*/
        /* and the argument that we would like to pass to the subprocess */
        /* the function UxExecuteSubproc() could be used instead of UxRunSubproc(). */
        /* The difference is that the second function force the subprocess to */
        /* and start again in case his process was previously running */
        if (ERROR == UxRunSubproc(h, NULL)) {
      printf("Cannot start the application \n");
      return;
        }
        /* this code could be included in the final code of some interface and the */
        /* code would be executed as soon as the interface is poped up in the screen */
        /* to do this we will include this code in the final code of that interface */

        UxPutSensitive(pushButton7, "false");

        }
}

static void activateCB_pushButtonGadget15( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /* this will release the handle given to the subprocess from the memory */
        /* For applications that might be terminated and restarted again many */
        /* times from the same interface, UxExecuteSubproc() function should be */
        /* used instead. In this case only a call to the UxRunSubproc() is needed */
        /* to restart it */
        UxDeleteSubproc(h);
        UxPutForeground(arrowButton1, "yellow");
        UxPutForeground(arrowButton2, "yellow");

        UxPutSensitive(pushButton4, "true");
        UxPutSensitive(pushButton7, "true");

        }
}

static void activateCB_pushButtonGadget22( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;
        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        char  s1[128];
        char  s2[128];
        /* if your process requires input from the terminal this is what you should do:*/
        /* create text widgets in the interface and fill in the appropriate values */
        /* for those text widgets as if you are entering these values from the terminal*/
        /* and send them to the process one after the other. This simulates entering */
        /* the values to through the command prompt. */
        sprintf(s1, "%s", UxGetText(text1)); /* get the value from widget */
        /* used to send the command to the subprocess. In this case, the string sent */
        /* is formed by the command pop followed by the name of the country, which */
        /* is read from the text widget text1. Output would be handled by UxAppendTo()*/
        /* which would display the string in the Text widget text2 */
        UxSendSubproc(h, s1); /* send the value to the subprocess */
        UxPutForeground(arrowButton1, "green");
        /* after this line accept the other values from the widgets and send them to */
        /* the subprocess to be executed. */

        sprintf(s2, "%s", UxGetText(text2));
        UxSendSubproc(h, s2);
        UxPutForeground(arrowButton2, "green");

        UxPutSensitive(pushButton4, "false");


        }
}

static void activateCB_pushButton12( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {

        exit(0);
        }
}

static void activateCB_pushButton13( UxWidget, UxClientData, UxCallbackArg )
        Widget          UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /*
        ** Prepare "options.rel" file to read in the reliabilities of information
        ** source vectors.
        */
        FILE *ifileptr;
        int relFlag;
        int rel1, rel2, rel3, rel4, rel5, rel6, rel7, rel8, rel9, rel10;

        ifileptr = fopen("options.rel", "r");
        fscanf(ifileptr, "%d", &relFlag); /* skip relFlag */
        fscanf(ifileptr, "%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",&rel1,&rel2,&rel3,&rel4,&rel5,
                         &rel6,&rel7,&rel8,&rel9,&rel10); /* read relFlag */
        fclose(ifileptr);

        /*
        ** Set the reliabilities of source vectors to the appropriate widgets
        */
        UxPutValue(scaleH2, rel1); /* S 1 */
        UxPutValue(scaleH3, rel2); /* S 2 */
        UxPutValue(scaleH5, rel3); /* S 3 */
        UxPutValue(scaleH4, rel4); /* s 4 */
        UxPutValue(scaleH9, rel5); /* s 5 */
        UxPutValue(scaleH8, rel6); /* s 6 */
        UxPutValue(scaleH6, rel7); /* s 7 */
        UxPutValue(scaleH7, rel8); /* s 8 */
        UxPutValue(scaleH10, rel9); /* s 9 */
        UxPutValue(scaleH11, rel10); /* s 10 */

        }
}

static void activateCB_pushButton14( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /*
        ** Prepare "options.rel" file to read in the reliabilities of information
        ** source vectors.
        */
        FILE *ifileptr;
        int relFlag;
        int rel1, rel2, rel3, rel4, rel5, rel6, rel7, rel8, rel9, rel10;
        /*
        ** Open "options.rel" file save the relflag (reliability Calculation Flag
        ** and close the "options.rel" again.
        */
        ifileptr = fopen("options.rel", "r");
        fscanf(ifileptr, "%d", &relFlag); /* skip relFlag */
        fclose(ifileptr);

         /*
        ** Set the reliabilities of source vectors to the appropriate widgets
        */
        rel1 = UxGetValue(scaleH2); /* S 1 */
        rel2 = UxGetValue(scaleH3); /* S 2 */
        rel3 = UxGetValue(scaleH5); /* S 3 */
        rel4 = UxGetValue(scaleH4); /* s 4 */
        rel5 = UxGetValue(scaleH9); /* s 5 */
        rel6 = UxGetValue(scaleH8); /* s 6 */
        rel7 = UxGetValue(scaleH6); /* s 7 */
        rel8 = UxGetValue(scaleH7); /* s 8 */
        rel9 = UxGetValue(scaleH10); /* s 9 */
        rel10 = UxGetValue(scaleH11); /* s 10 */

        /*
        ** Open the "options.rel" file and save the reliabilities of information
        ** sources set by the user interface. Do not forget to save the reliability
        ** calculation flag (relFlag) too.
        */
        ifileptr = fopen("options.rel", "w");
        fprintf(ifileptr, "%d\n", relFlag); /* do not change relFlag */
        fprintf(ifileptr, "%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",rel1,rel2,rel3,rel4,rel5,
                          rel6,rel7,rel8,rel9,rel10);

        fclose(ifileptr);

        }
}

static void createCB_toggleButton1( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {

XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {

      }
}

static void armCB_toggleButton1( UxWidget, UxClientData, UxCallbackArg )
        Widget UxWidget;
        XtPointer UxClientData, UxCallbackArg;

{
        swidget UxThisWidget;

        UxThisWidget = UxWidgetToSwidget( UxWidget );

        {
        /*
        ** pointer to file
        */
        FILE *ifileptr, ofileptr;
        /*
        ** relFlag = 1 means reliability calculation is set (allowed).
        ** relFlag = 0 means reliability calculation is not set.
        */
        int relFlag;
        int rel1, rel2, rel3, rel4, rel5, rel6, rel7, rel8, rel9, rel10;
        /*
        ** open "options.rel" file and switch the reliability
        ** option. If the option was 1 change it to zero and if the
        ** option was zero change it to 1.
        */
        relFlag = 0;
        ifileptr = fopen("options.rel", "r");
        fscanf(ifileptr, "%d", &relFlag); /* read relFlag */
        fscanf(ifileptr, "%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",&rel1,&rel2,&rel3,&rel4,&rel5,
                        &rel6,&rel7,&rel8,&rel9,&rel10); /* read relFlag */
        fclose(ifileptr);

        if ( relFlag == 0 )
        { /* change the relflag to 1 */
           relFlag = 1;}
        else
        { /* change the relFlag to 0 */
          relFlag = 0;
        }

        /*
        ** Save relFlag in the options file.
        */

        ifileptr = fopen("options.rel", "w");
        fprintf(ifileptr, "%d\n", relFlag); /* Save option */
        fprintf(ifileptr, "%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",rel1,rel2,rel3,rel4,rel5,
                         rel6,rel7,rel8,rel9,rel10);
        fclose(ifileptr);

        }
}

/*-------------------------------------------------------------------------------------------
**      The 'build_' function creates all the swidgets and X widgets,
**      and sets their properties to the values specified in the
**      Property Editor.
**------------------------------------------------------------------------------------------*/
static swidget _Uxbuild_UncertainDatabase()
{
        /* Create the swidgets */


        /* Creation of UncertainDatabase */
        UncertainDatabase = UxCreateBulletinBoard( "UncertainDatabase", UxParent );
        UxPutDefaultShell( UncertainDatabase, "transientShell" );

        UxPutResizePolicy( UncertainDatabase, "resize_none" );
        UxPutWidth( UncertainDatabase, 600 );
        UxPutHeight( UncertainDatabase, 900 );
        UxPutX( UncertainDatabase, 332 );
        UxPutY( UncertainDatabase, 416 );
        UxPutUnitType( UncertainDatabase, "pixels" );
        UxCreateWidget( UncertainDatabase );

        /* Creation of pushButton1 */
        pushButton1 = UxCreatePushButton( "pushButton1", UncertainDatabase );
        UxPutX( pushButton1, 12 );
        UxPutY( pushButton1, 428 );
        UxPutTranslations( pushButton1, transTable1 );
        UxPutLabelString( pushButton1, "  Popup QueryEditor  " );
        UxPutWidth( pushButton1, 188 );
        UxPutFontList( pushButton1, "-Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-
                                        ISO8859-1=FONTLIST_DEFAULT_TAG_STRING" );
```

```
UxCreateWidget( pushButton1 );

UxAddCallback( pushButton1, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton1,
 (XtPointer) NULL );

/* Creation of pushButton3 */
pushButton3 = UxCreatePushButton( "pushButton3", UncertainDatabase );
UxPutX( pushButton3, 200 );
UxPutY( pushButton3, 428 );
UxPutLabelString( pushButton3, "Popup Query Answers  " );
UxPutWidth( pushButton3, 192 );
UxCreateWidget( pushButton3 );

UxAddCallback( pushButton3, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton3,
 (XtPointer) NULL );

/* Creation of frame5 */
frame5 = UxCreateFrame( "frame5", UncertainDatabase );
UxPutWidth( frame5, 586 );
UxPutHeight( frame5, 238 );
UxPutX( frame5, 8 );
UxPutY( frame5, 458 );
UxCreateWidget( frame5 );


/* Creation of form3 */
form3 = UxCreateForm( "form3", frame5 );
UxPutWidth( form3, 578 );
UxPutHeight( form3, 234 );
UxPutResizePolicy( form3, "resize_none" );
UxPutX( form3, 4 );
UxPutY( form3, 2 );
UxCreateWidget( form3 );

/* Creation of pushButton7 */
pushButton7 = UxCreatePushButton( "pushButton7", form3 );
UxPutX( pushButton7, 4 );
UxPutY( pushButton7, 204 );
UxPutForeground( pushButton7, "black" );
UxPutLabelString( pushButton7, "    Connect to INGRES    " );
UxPutSensitive( pushButton7, "true" );
UxPutWidth( pushButton7, 191 );
UxCreateWidget( pushButton7 );

UxAddCallback( pushButton7, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton7,
 (XtPointer) NULL );


/* Creation of pushButton2 */
pushButton2 = UxCreatePushButton( "pushButton2", form3 );
UxPutX( pushButton2, 388 );
UxPutY( pushButton2, 204 );
UxPutForeground( pushButton2, "black" );
UxPutLabelString( pushButton2, "     Close Connection    " );
UxPutSensitive( pushButton2, "true" );
UxPutWidth( pushButton2, 191 );
UxCreateWidget( pushButton2 );

UxAddCallback( pushButton2, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton2,
 (XtPointer) NULL );
```

```
/* Creation of pushButton4 */
pushButton4 = UxCreatePushButton( "pushButton4", form3 );
UxPutX( pushButton4, 196 );
UxPutY( pushButton4, 204 );
UxPutForeground( pushButton4, "black" );
UxPutLabelString( pushButton4, "        Get Answers        " );
UxPutWidth( pushButton4, 192 );
UxCreateWidget( pushButton4 );

UxAddCallback( pushButton4, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton4,
 (XtPointer) NULL );


/* Creation of label3 */
label3 = UxCreateLabel( "label3", form3 );
UxPutX( label3, 108 );
UxPutY( label3, 4 );
UxPutForeground( label3, "black" );
UxPutLabelString( label3, "Database :" );
UxPutFontList( label3, "-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso8859-1" );
UxPutWidth( label3, 68 );
UxCreateWidget( label3 );

/* Creation of label4 */
label4 = UxCreateLabel( "label4", form3 );
UxPutX( label4, 88 );
UxPutY( label4, 120 );
UxPutForeground( label4, "black" );
UxPutLabelString( label4, "     Query :" );
UxPutFontList( label4, "-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso8859-1" );
UxPutWidth( label4, 68 );
UxPutHeight( label4, 15 );
UxCreateWidget( label4 );

/* Creation of text1 */
text1 = UxCreateText( "text1", form3 );
UxPutWidth( text1, 468 );
UxPutX( text1, 108 );
UxPutY( text1, 20 );
UxPutText( text1, "said" );
UxPutForeground( text1, "black" );
UxPutHighlightColor( text1, "black" );
UxPutHeight( text1, 47 );
UxPutFontList( text1, "-adobe-helvetica-bold-r-normal--14-140-75-75-p-82-iso8859-1" );
UxCreateWidget( text1 );

/* Creation of text2 */
text2 = UxCreateText( "text2", form3 );
UxPutWidth( text2, 468 );
UxPutX( text2, 108 );
UxPutY( text2, 72 );
UxPutText( text2, "select * from s;" );
UxPutHeight( text2, 47 );
UxPutFontList( text2, "-Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-
                ISO8859-1=FONTLIST_DEFAULT_TAG_STRING" );
UxCreateWidget( text2 );

/* Creation of arrowButton1 */
arrowButton1 = UxCreateArrowButton( "arrowButton1", form3 );
UxPutX( arrowButton1, 68 );
UxPutY( arrowButton1, 20 );
UxPutForeground( arrowButton1, "yellow" );
```

```
UxPutBorderColor( arrowButton1, "black" );
UxPutBorderWidth( arrowButton1, 0 );
UxPutHeight( arrowButton1, 48 );
UxPutWidth( arrowButton1, 40 );
UxCreateWidget( arrowButton1 );

/* Creation of arrowButton2 */
arrowButton2 = UxCreateArrowButton( "arrowButton2", form3 );
UxPutX( arrowButton2, 68 );
UxPutY( arrowButton2, 72 );
UxPutForeground( arrowButton2, "yellow" );
UxPutHeight( arrowButton2, 48 );
UxPutWidth( arrowButton2, 40 );
UxCreateWidget( arrowButton2 );

/* Creation of separator1 */
separator1 = UxCreateSeparator( "separator1", form3 );
UxPutWidth( separator1, 568 );
UxPutHeight( separator1, 16 );
UxPutX( separator1, 4 );
UxPutY( separator1, 132 );
UxCreateWidget( separator1 );

/* Creation of pushButtonGadget11 */
pushButtonGadget11 = UxCreatePushButtonGadget( "pushButtonGadget11", form3 );
UxPutX( pushButtonGadget11, 20 );
UxPutY( pushButtonGadget11, 20 );
UxPutLabelString( pushButtonGadget11, "" );
UxCreateWidget( pushButtonGadget11 );

/* Creation of pushButtonGadget14 */
pushButtonGadget14 = UxCreatePushButtonGadget( "pushButtonGadget14", form3 );
UxPutX( pushButtonGadget14, 68 );
UxPutY( pushButtonGadget14, 152 );
UxPutLabelString( pushButtonGadget14, "" );
UxCreateWidget( pushButtonGadget14 );

UxAddCallback( pushButtonGadget14, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButtonGadget14,
 (XtPointer) NULL );

/* Creation of pushButtonGadget15 */
pushButtonGadget15 = UxCreatePushButtonGadget( "pushButtonGadget15", form3 );
UxPutX( pushButtonGadget15, 460 );
UxPutY( pushButtonGadget15, 152 );
UxPutLabelString( pushButtonGadget15, "" );
UxPutWidth( pushButtonGadget15, 184 );
UxCreateWidget( pushButtonGadget15 );

UxAddCallback( pushButtonGadget15, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButtonGadget15,
 (XtPointer) NULL );

/* Creation of pushButtonGadget21 */
pushButtonGadget21 = UxCreatePushButtonGadget( "pushButtonGadget21", form3 );
UxPutX( pushButtonGadget21, 20 );
UxPutY( pushButtonGadget21, 72 );
UxPutLabelString( pushButtonGadget21, "" );
UxCreateWidget( pushButtonGadget21 );

/* Creation of pushButtonGadget22 */
pushButtonGadget22 = UxCreatePushButtonGadget( "pushButtonGadget22", form3 );
UxPutX( pushButtonGadget22, 264 );
UxPutY( pushButtonGadget22, 153 );
```

```
UxPutLabelString( pushButtonGadget22, "" );
UxCreateWidget( pushButtonGadget22 );

UxAddCallback( pushButtonGadget22, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButtonGadget22,
 (XtPointer) NULL );

/* Creation of pushButton12 */
pushButton12 = UxCreatePushButton( "pushButton12", UncertainDatabase );
UxPutX( pushButton12, 392 );
UxPutY( pushButton12, 428 );
UxPutWidth( pushButton12, 192 );
UxPutLabelString( pushButton12, "Quit ..." );
UxCreateWidget( pushButton12 );

UxAddCallback( pushButton12, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton12,
 (XtPointer) NULL );

/* Creation of frame7 */
frame7 = UxCreateFrame( "frame7", UncertainDatabase );
UxPutWidth( frame7, 578 );
UxPutHeight( frame7, 414 );
UxPutX( frame7, 10 );
UxPutY( frame7, 10 );
UxCreateWidget( frame7 );

/* Creation of form5 */
form5 = UxCreateForm( "form5", frame7 );
UxPutWidth( form5, 200 );
UxPutHeight( form5, 200 );
UxPutResizePolicy( form5, "resize_none" );
UxPutX( form5, 68 );
UxPutY( form5, 128 );
UxCreateWidget( form5 );

/* Creation of frame8 */
frame8 = UxCreateFrame( "frame8", form5 );
UxPutWidth( frame8, 572 );
UxPutHeight( frame8, 408 );
UxPutX( frame8, 0 );
UxPutY( frame8, 0 );
UxCreateWidget( frame8 );

/* Creation of scrolledWindowText2 */
scrolledWindowText2 = UxCreateScrolledWindow( "scrolledWindowText2", frame8 );
UxPutScrollingPolicy( scrolledWindowText2, "application_defined" );
UxPutVisualPolicy( scrolledWindowText2, "variable" );
UxPutScrollBarDisplayPolicy( scrolledWindowText2, "static" );
UxPutX( scrolledWindowText2, 28 );
UxPutY( scrolledWindowText2, 16 );
UxCreateWidget( scrolledWindowText2 );

/* Creation of scrolledText2 */
scrolledText2 = UxCreateScrolledText( "scrolledText2", scrolledWindowText2 );
UxPutWidth( scrolledText2, 200 );
UxPutHeight( scrolledText2, 200 );
UxPutEditMode( scrolledText2, "multi_line_edit" );
UxPutForeground( scrolledText2, "black" );
UxPutBackground( scrolledText2, "#7e88ab" );
UxCreateWidget( scrolledText2 );

/* Creation of frame9 */
frame9 = UxCreateFrame( "frame9", UncertainDatabase );
```

```
UxPutWidth( frame9, 520 );
UxPutHeight( frame9, 414 );
UxPutX( frame9, 608 );
UxPutY( frame9, 10 );
UxCreateWidget( frame9 );

/* Creation of form6 */
form6 = UxCreateForm( "form6", frame9 );
UxPutWidth( form6, 522 );
UxPutHeight( form6, 410 );
UxPutResizePolicy( form6, "resize_none" );
UxPutX( form6, -4 );
UxPutY( form6, 2 );
UxCreateWidget( form6 );

/* Creation of scaleH2 */
scaleH2 = UxCreateScale( "scaleH2", form6 );
UxPutWidth( scaleH2, 448 );
UxPutHeight( scaleH2, 36 );
UxPutOrientation( scaleH2, "horizontal" );
UxPutX( scaleH2, 58 );
UxPutY( scaleH2, 40 );
UxPutShowValue( scaleH2, "true" );
UxPutShadowThickness( scaleH2, 3 );
UxPutTitleString( scaleH2, "" );
UxPutForeground( scaleH2, "black" );
UxPutBottomShadowColor( scaleH2, "#43485a" );
UxPutValue( scaleH2, 100 );
UxCreateWidget( scaleH2 );

/* Creation of scaleH10 */
scaleH10 = UxCreateScale( "scaleH10", form6 );
UxPutWidth( scaleH10, 450 );
UxPutHeight( scaleH10, 36 );
UxPutOrientation( scaleH10, "horizontal" );
UxPutX( scaleH10, 58 );
UxPutY( scaleH10, 328 );
UxPutShowValue( scaleH10, "true" );
UxPutShadowThickness( scaleH10, 3 );
UxCreateWidget( scaleH10 );

/* Creation of scaleH7 */
scaleH7 = UxCreateScale( "scaleH7", form6 );
UxPutWidth( scaleH7, 448 );
UxPutHeight( scaleH7, 36 );
UxPutOrientation( scaleH7, "horizontal" );
UxPutX( scaleH7, 58 );
UxPutY( scaleH7, 292 );
UxPutShowValue( scaleH7, "true" );
UxPutShadowThickness( scaleH7, 3 );
UxCreateWidget( scaleH7 );

/* Creation of scaleH6 */
scaleH6 = UxCreateScale( "scaleH6", form6 );
UxPutWidth( scaleH6, 448 );
UxPutHeight( scaleH6, 40 );
UxPutOrientation( scaleH6, "horizontal" );
UxPutX( xPutX( scaleH6, 58 );
UxPutY( scaleH6, 252 );
UxPutShowValue( scaleH6, "true" );
UxPutShadowThickness( scaleH6, 3 );
UxCreateWidget( scaleH6 );

/* Creation of scaleH8 */
```

```
scaleH8 = UxCreateScale( "scaleH8", form6 );
UxPutWidth( scaleH8, 448 );
UxPutHeight( scaleH8, 36 );
UxPutOrientation( scaleH8, "horizontal" );
UxPutX( scaleH8, 58 );
UxPutY( scaleH8, 216 );
UxPutShowValue( scaleH8, "true" );
UxPutShadowThickness( scaleH8, 3 );
UxCreateWidget( scaleH8 );

/* Creation of scaleH9 */
scaleH9 = UxCreateScale( "scaleH9", form6 );
UxPutWidth( scaleH9, 448 );
UxPutHeight( scaleH9, 36 );
UxPutOrientation( scaleH9, "horizontal" );
UxPutX( scaleH9, 58 );
UxPutY( scaleH9, 180 );
UxPutShowValue( scaleH9, "true" );
UxPutShadowThickness( scaleH9, 3 );
UxCreateWidget( scaleH9 );

/* Creation of scaleH4 */
scaleH4 = UxCreateScale( "scaleH4", form6 );
UxPutWidth( scaleH4, 448 );
UxPutHeight( scaleH4, 36 );
UxPutOrientation( scaleH4, "horizontal" );
UxPutX( scaleH4, 58 );
UxPutY( scaleH4, 144 );
UxPutShowValue( scaleH4, "true" );
UxPutShadowThickness( scaleH4, 3 );
UxCreateWidget( scaleH4 );

/* Creation of scaleH5 */
scaleH5 = UxCreateScale( "scaleH5", form6 );
UxPutWidth( scaleH5, 448 );
UxPutHeight( scaleH5, 36 );
UxPutOrientation( scaleH5, "horizontal" );
UxPutX( scaleH5, 58 );
UxPutY( scaleH5, 108 );
UxPutShowValue( scaleH5, "true" );
UxPutShadowThickness( scaleH5, 3 );
UxCreateWidget( scaleH5 );

/* Creation of scaleH3 */
scaleH3 = UxCreateScale( "scaleH3", form6 );
UxPutWidth( scaleH3, 448 );
UxPutheight( scaleH3, 32 );
UxPutOrientation( scaleH3, "horizontal" );
UxPutX( scaleH3, 58 );
UxPutY( scaleH3, 76 );
UxPutShowValue( scaleH3, "true" );
UxPutShadowThickness( scaleH3, 3 );
UxCreateWidget( scaleH3 );

/* Creation of scaleH11 */
scaleH11 = UxCreateScale( "scaleH11", form6 );
UxPutWidth( scaleH11, 448 );
UxPutHeight( scaleH11, 36 );
UxPutOrientation( scaleH11, "horizontal" );
UxPutX( scaleH11, 58 );
UxPutY( scaleH11, 364 );
UxPutForeground( scaleH11, "black" );
UxPutShowValue( scaleH11, "true" );
UxPutShadowThickness( scaleH11, 3 );
```

```
UxCreateWidget( scaleH11 );

/* Creation of label9 */
label9 = UxCreateLabel( "label9", form6 );
UxPutX( label9, 56 );
UxPutY( label9, 12 );
UxPutWidth( label9, 448 );
UxPutHeight( label9, 25 );
UxPutLabelString( label9, "Reliability of Information Sources" );
UxPutFontList( label9, "-adobe-courier-bold-o-normal--17-120-100-100-m-100-iso8859-1" );
UxCreateWidget( label9 );

/* Creation of label10 */
label10 = UxCreateLabel( "label10", form6 );
UxPutX( label10, 8 );
UxPutY( label10, 40 );
UxPutHeight( label10, 36 );
UxPutLabelStrirg( label10, "S 1" );
UxCreateWidget( label10 );

/* Creation of label11 */
label11 = UxCreateLabel( "label11", form6 );
UxPutX( label11, 8 );
UxPutY( label11, 76 );
UxPutHeight( label11, 32 );
UxPutLabelString( label11, "S 2" );
UxCreateWidget( label11 );

/* Creation of label12 */
label12 = UxCreateLabel( "label12", form6 );
UxPutX( label12, 8 );
UxPutY( label12, 112 );
UxPutHeight( label12, 32 );
UxPutLabelString( label12, "S 3" );
UxCreateWidget( label12 );

/* Creation of label13 */
label13 = UxCreateLabel( "label13", form6 );
UxPutX( label13, 12 );
UxPutY( label13, 148 );
UxPutHeight( label13, 32 );
UxPutLabelString( label13, "S 4" );
UxCreateWidget( label13 );

/* Creation of label14 */
label14 = UxCreateLabel( "label14", form6 );
UxPutX( label14, 12 );
UxPutY( label14, 184 );
UxPutHeight( label14, 32 );
UxPutLabelString( label14, "S 5" );
UxCreateWidget( label14 );

/* Creation of label15 */
label15 = UxCreateLabel( "label15", form6 );
UxPutX( label15, 12 );
UxPutY( label15, 220 );
UxPutHeight( label15, 32 );
UxPutLabelString( label15, "S 6" );
UxCreateWidget( label15 );

/* Creation of label16 */
label16 = UxCreateLabel( "label16", form6 );
UxPutX( label16, 12 );
UxPutY( label16, 260 );
```

```
UxPutHeight( label16, 32 );
UxPutLabelString( label16, "S 7" );
UxCreateWidget( label16 );

/* Creation of label17 */
label17 = UxCreateLabel( "label17", form6 );
UxPutX( label17, 12 );
UxPutY( label17, 296 );
UxPutHeight( label17, 32 );
UxPutLabelString( label17, "S 8" );
UxCreateWidget( label17 );

/* Creation of label18 */
label18 = UxCreateLabel( "label18", form6 );
UxPutX( label18, 12 );
UxPutY( label18, 332 );
UxPutHeight( label18, 32 );
UxPutLabelString( label18, "S 9" );
UxCreateWidget( label18 );

/* Creation of label19 */
label19 = UxCreateLabel( "label19", form6 );
UxPutX( label19, 12 );
UxPutY( label19, 368 );
UxPutHeight( label19, 32 );
UxPutLabelString( label19, "S 10" );
UxCreateWidget( label19 );

/* Creation of pushButton13 */
pushButton13 = UxCreatePushButton( "pushButton13", UncertainDatabase );
UxPutX( pushButton13, 612 );
UxPutY( pushButton13, 428 );
UxPutWidth( pushButton13, 172 );
UxPutLabelString( pushButton13, "Load Reliabilities" );
UxCreateWidget( pushButton13 );

UxAddCallback( pushButton13, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton13,
 (XtPointer) NULL );


/* Creation of pushButton14 */
pushButton14 = UxCreatePushButton( "pushButton14", UncertainDatabase );
UxPutX( pushButton14, 784 );
UxPutY( pushButton14, 428 );
UxPutWidth( pushButton14, 172 );
UxPutLabelString( pushButton14, "Save Reliabilities" );
UxCreateWidget( pushButton14 );

UxAddCallback( pushButton14, XmNactivateCallback,
 (XtCallbackProc) activateCB_pushButton14,
 (XtPointer) NULL );

/* Creation of pushButton15 */
pushButton15 = UxCreatePushButton( "pushButton15", UncertainDatabase );
UxPutX( pushButton15, 956 );
UxPutY( pushButton15, 428 );
UxPutWidth( pushButton15, 168 );
UxPutLabelString( pushButton15, "Help ..." );
UxCreateWidget( pushButton15 );

/* Creation of frame10 */
frame10 = UxCreateFrame( "frame10", UncertainDatabase );
UxPutWidth( frame10, 582 );
```

```
UxPutHeight( frame10, 146 );
UxPutX( frame10, 10 );
UxPutY( frame10, 715 );
UxCreateWidget( frame10 );

/* Creation of form7 */
form7 = UxCreateForm( "form7", frame10 );
UxPutWidth( form7, 578 );
UxPutHeight( form7, 142 );
UxPutResizePolicy( form7, "resize_none" );
UxPutX( form7, 2 );
UxPutY( form7, 3 );
UxCreateWidget( form7 );

/* Creation of pushButton16 */
pushButton16 = UxCreatePushButton( "pushButton16", form7 );
UxPutX( pushButton16, 192 );
UxPutY( pushButton16, 112 );
UxPutWidth( pushButton16, 188 );
UxPutLabelString( pushButton16, "Palette: Colors" );
UxCreateWidget( pushButton16 );

/* Creation of frame12 */
frame12 = UxCreateFrame( "frame12", form7 );
UxPutWidth( frame12, 572 );
UxPutHeight( frame12, 40 );
UxPutX( frame12, 4 );
UxPutY( frame12, 4 );
UxCreateWidget( frame12 );

/* Creation of label20 */
label20 = UxCreateLabel( "label20", frame12 );
UxPutX( label20, 12 );
UxPutY( label20, 12 );
UxPutFontList( label20, "-adobe-courier-bold-o-normal--14-140-75-75-m-90-iso8859-1" );
UxPutLabelString( label20, "Options" );
UxPutWidth( label20, 552 );
UxCreateWidget( label20 );

/* Creation of pushButton17 */
pushButton17 = UxCreatePushButton( "pushButton17", form7 );
UxPutX( pushButton17, 4 );
UxPutY( pushButton17, 112 );
UxPutWidth( pushButton17, 188 );
UxPutLabelString( pushButton17, "Text Font List" );
UxCreateWidget( pushButton17 );

/* Creation of toggleButton1 */
toggleButton1 = UxCreateToggleButton( "toggleButton1", form7 );
UxPutX( toggleButton1, 396 );
UxPutY( toggleButton1, 112 );
UxPutLabelString( toggleButton1, "Reliability Calculation" );
UxCreateWidget( toggleButton1 );
createCB_toggleButton1( UxGetWidget( toggleButton1 ),
 (XtPointer) NULL, (XtPointer) NULL );

UxAddCallback( toggleButton1, XmNarmCallback,
 (XtCallbackProc) armCB_toggleButton1,
 (XtPointer) NULL );

/* Creation of pushButtonGadget12 */
pushButtonGadget12 = UxCreatePushButtonGadget( "pushButtonGadget12", form7 );
UxPutX( pushButtonGadget12, 260 );
UxPutY( pushButtonGadget12, 56 );
```

```
UxPutLabelString( pushButtonGadget12, "" );
UxCreateWidget( pushButtonGadget12 );

/* Creation of pushButtonGadget16 */
pushButtonGadget16 = UxCreatePushButtonGadget( "pushButtonGadget16", form7 );
UxPutX( pushButtonGadget16, 68 );
UxPutY( pushButtonGadget16, 56 );
UxPutLabelString( pushButtonGadget16, "" );
UxCreateWidget( pushButtonGadget16 );

/* Creation of pushButtonGadget26 */
pushButtonGadget26 = UxCreatePushButtonGadget( "pushButtonGadget26", form7 );
UxPutX( pushButtonGadget26, 452 );
UxPutY( pushButtonGadget26, 56 );
UxPutLabelString( pushButtonGadget26, "" );
UxCreateWidget( pushButtonGadget26 );

/* Creation of frame11 */
frame11 = UxCreateFrame( "frame11", UncertainDatabase );
UxPutWidth( frame11, 520 );
UxPutHeight( frame11, 404 );
UxPutX( frame11, 608 );
UxPutY( frame11, 456 );
UxCreateWidget( frame11 );

/* Creation of form8 */
form8 = UxCreateForm( "form8", frame11 );
UxPutWidth( form8, 200 );
UxPutHeight( form8, 200 );
UxPutResizePolicy( form8, "resize_none" );
UxPutX( form8, 152 );
UxPutY( form8, 112 );
UxCreateWidget( form8 );

/* Creation of scrolledWindow2 */
scrolledWindow2 = UxCreateScrolledWindow( "scrolledWindow2", form8 );
UxPutScrollingPolicy( scrolledWindow2, "automatic" );
UxPutWidth( scrolledWindow2, 510 );
UxPutHeight( scrolledWindow2, 351 );
UxPutX( scrolledWindow2, 2 );
UxPutY( scrolledWindow2, 34 );
UxCreateWidget( scrolledWindow2 );

/* Creation of form9 */
form9 = UxCreateForm( "form9", scrolledWindow2 );
UxPutWidth( form9, 508 );
UxPutHeight( form9, 348 );
UxPutResizePolicy( form9, "resize_none" );
UxPutX( form9, 4 );
UxPutY( form9, 7 );
UxCreateWidget( form9 );

/* Creation of frame13 */
frame13 = UxCreateFrame( "frame13", form9 );
UxPutWidth( frame13, 110 );
UxPutHeight( frame13, 98 );
UxPutX( frame13, 5 );
UxPutY( frame13, 134 );
UxPutShadowThickness( frame13, 3 );
UxPutShadowType( frame13, "shadow_out" );
UxCreateWidget( frame13 );

/* Creation of rowColumn2 */
rowColumn2 = UxCreateRowColumn( "rowColumn2", frame13 );
```

```
UxPutWidth( rowColumn2, 106 );
UxPutHeight( rowColumn2, 94 );
UxPutX( rowColumn2, 2 );
UxPutY( rowColumn2, 2 );
UxCreateWidget( rowColumn2 );

/* Creation of frame14 */
frame14 = UxCreateFrame( "frame14", rowColumn2 );
UxPutWidth( frame14, 98 );
UxPutHeight( frame14, 85 );
UxPutX( frame14, 3 );
UxPutY( frame14, 3 );
UxPutShadowType( frame14, "shadow_out" );
UxPutShadowThickness( frame14, 3 );
UxCreateWidget( frame14 );

/* Creation of rowColumn3 */
rowColumn3 = UxCreateRowColumn( "rowColumn3", frame14 );
UxPutWidth( rowColumn3, 64 );
UxPutHeight( rowColumn3, 79 );
UxPutX( rowColumn3, 12 );
UxPutY( rowColumn3, 0 );
UxCreateWidget( rowColumn3 );

/* Creation of pushButtonGadget1 */
pushButtonGadget1 = UxCreatePushButtonGadget( "pushButtonGadget1", rowColumn3 );
UxPutX( pushButtonGadget1, 0 );
UxPutY( pushButtonGadget1, 0 );
UxPutHeight( pushButtonGadget1, 46 );
UxPutShadowThickness( pushButtonGadget1, 1 );
UxPutAlignment( pushButtonGadget1, "alignment_center" );
UxPutWidth( pushButtonGadget1, 469 );
UxPutLabelString( pushButtonGadget1, "" );
UxCreateWidget( pushButtonGadget1 );

/* Creation of pushButtonGadget2 */
pushButtonGadget2 = UxCreatePushButtonGadget( "pushButtonGadget2", rowColumn3 );
UxPutX( pushButtonGadget2, 3 );
UxPutY( pushButtonGadget2, 52 );
UxPutLabelString( pushButtonGadget2, "   Tables   " );
UxPutWidth( pushButtonGadget2, 44 );
UxCreateWidget( pushButtonGadget2 );

/* Creation of frame15 */
frame15 = UxCreateFrame( "frame15", form9 );
UxPutWidth( frame15, 110 );
UxPutHeight( frame15, 98 );
UxPutX( frame15, 122 );
UxPutY( frame15, 135 );
UxPutShadowThickness( frame15, 3 );
UxPutShadowType( frame15, "shadow_out" );
UxCreateWidget( frame15 );

/* Creation of rowColumn4 */
rowColumn4 = UxCreateRowColumn( "rowColumn4", frame15 );
UxPutWidth( rowColumn4, 106 );
UxPutHeight( rowColumn4, 94 );
UxPutX( rowColumn4, 2 );
UxPutY( rowColumn4, 2 );
UxCreateWidget( rowColumn4 );

/* Creation of frame16 */
frame16 = UxCreateFrame( "frame16", rowColumn4 );
UxPutWidth( frame16, 98 );
```

```
UxPutHeight( frame16, 85 );
UxPutX( frame16, 3 );
UxPutY( frame16, 3 );
UxPutShadowType( frame16, "shadow_out" );
UxPutShadowThickness( frame16, 3 );
UxCreateWidget( frame16 );

/* Creation of rowColumn5 */
rowColumn5 = UxCreateRowColumn( "rowColumn5", frame16 );
UxPutWidth( rowColumn5, 64 );
UxPutHeight( rowColumn5, 79 );
UxPutX( rowColumn5, 12 );
UxPutY( rowColumn5, 0 );
UxCreateWidget( rowColumn5 );


/* Creation of pushButtonGadget3 */
pushButtonGadget3 = UxCreatePushButtonGadget( "pushButtonGadget3", rowColumn5 );
UxPutX( pushButtonGadget3, 0 );
UxPutY( pushButtonGadget3, 0 );
UxPutHeight( pushButtonGadget3, 46 );
UxPutShadowThickness( pushButtonGadget3, 1 );
UxPutAlignment( pushButtonGadget3, "alignment_end" );
UxPutWidth( pushButtonGadget3, 469 );
UxPutLabelString( pushButtonGadget3, "" );
UxCreateWidget( pushButtonGadget3 );

/* Creation of pushButtonGadget4 */
pushButtonGadget4 = UxCreatePushButtonGadget( "pushButtonGadget4", rowColumn5 );
UxPutX( pushButtonGadget4, 3 );
UxPutY( pushButtonGadget4, 52 );
UxPutLabelString( pushButtonGadget4, "   Reports  " );
UxPutWidth( pushButtonGadget4, 86 );
UxPutHeight( pushButtonGadget4, 32 );
UxCreateWidget( pushButtonGadget4 );

/* Creation of frame19 */
frame19 = UxCreateFrame( "frame19", form9 );
UxPutWidth( frame19, 110 );
UxPutHeight( frame19, 98 );
UxPutX( frame19, 241 );
UxPutY( frame19, 134 );
UxPutShadowThick ess( frame19, 3 );
UxPutShadowType( frame19, "shadow_out" );
UxCreateWidget( frame19 );

/* Creation of rowColumn8 */
rowColumn8 = UxCreateRowColumn( "rowColumn8", frame19 );
UxPutWidth( rowColumn8, 106 );
UxPutHeight( rowColumn8, 94 );
UxPutX( rowColumn8, 2 );
UxPutY( rowColumn8, 2 );
UxCreateWidget( rowColumn8 );

/* Creation of frame20 */
frame20 = UxCreateFrame( "frame20", rowColumn8 );
UxPutWidth( frame20, 98 );
UxPutHeight( frame20, 85 );
UxPutX( frame20, 3 );
UxPutY( frame20, 3 );
UxPutShadowType( frame20, "shadow_out" );
UxPutShadowThickness( frame20, 3 );
UxCreateWidget( frame20 );
```

```
/* Creation of rowColumn9 */
rowColumn9 = UxCreateRowColumn( "rowColumn9", frame20 );
UxPutWidth( rowColumn9, 64 );
UxPutHeight( rowColumn9, 79 );
UxPutX( rowColumn9, 12 );
UxPutY( rowColumn9, 0 );
UxCreateWidget( rowColumn9 );

/* Creation of pushButtonGadget7 */
pushButtonGadget7 = UxCreatePushButtonGadget( "pushButtonGadget7", rowColumn9 );
UxPutX( pushButtonGadget7, 0 );
UxPutY( pushButtonGadget7, 0 );
UxPutHeight( pushButtonGadget7, 46 );
UxPutShadowThickness( pushButtonGadget7, 1 );
UxPutAlignment( pushButtonGadget7, "alignment_end" );
UxPutWidth( pushButtonGadget7, 469 );
UxPutLabelString( pushButtonGadget7, "" );
UxCreateWidget( pushButtonGadget7 );

/* Creation of pushButtonGadget8 */
pushButtonGadget8 = UxCreatePushButtonGadget( "pushButtonGadget8", rowColumn9 );
UxPutX( pushButtonGadget8, 3 );
UxPutY( pushButtonGadget8, 52 );
UxPutLabelString( pushButtonGadget8, "    Forms " );
UxPutWidth( pushButtonGadget8, 44 );
UxCreateWidget( pushButtonGadget8 );

/* Creation of frame17 */
frame17 = UxCreateFrame( "frame17", form9 );
UxPutWidth( frame17, 110 );
UxPutHeight( frame17, 98 );
UxPutX( frame17, 357 );
UxPutY( frame17, 134 );
UxPutShadowThickness( frame17, 3 );
UxPutShadowType( frame17, "shadow_out" );
UxCreateWidget( frame17 );

/* Creation of rowColumn6 */
rowColumn6 = UxCreateRowColumn( "rowColumn6", frame17 );
UxPutWidth( rowColumn6, 104 );
UxPutHeight( rowColumn6, 97 );
UxPutX( rowColumn6, 3 );
UxPutY( rowColumn6, 3 );
UxCreateWidget( rowColumn6 );

/* Creation of frame18 */
frame18 = UxCreateFrame( "frame18", rowColumn6 );
UxPutWidth( frame18, 98 );
UxPutHeight( frame18, 85 );
UxPutX( frame18, 3 );
UxPutY( frame18, 3 );
UxPutShadowType( frame18, "shadow_out" );
UxPutShadowThickness( frame18, 3 );
UxCreateWidget( frame18 );

/* Creation of rowColumn7 */
rowColumn7 = UxCreateRowColumn( "rowColumn7", frame18 );
UxPutWidth( rowColumn7, 64 );
UxPutHeight( rowColumn7, 79 );
UxPutX( rowColumn7, 12 );
UxPutY( rowColumn7, 0 );
UxCreateWidget( rowColumn7 );

/* Creation of pushButtonGadget5 */
```

```
pushButtonGadget5 = UxCreatePushButtonGadget( "pushButtonGadget5", rowColumn7 );
UxPutX( pushButtonGadget5, 0 );
UxPutY( pushButtonGadget5, 0 );
UxPutHeight( pushButtonGadget5, 46 );
UxPutShadowThickness( pushButtonGadget5, 1 );
UxPutAlignment( pushButtonGadget5, "alignment_end" );
UxPutWidth( pushButtonGadget5, 469 );
UxPutLabelString( pushButtonGadget5, "" );
UxCreateWidget( pushButtonGadget5 );

/* Creation of pushButtonGadget6 */
pushButtonG dget6 = UxCreatePushButtonGadget( "pushButtonGadget6", rowColumn7 );
UxPutX( pushButtonGadget6, 3 );
UxPutY( pushButtonGadget6, 52 );
UxPutLabelString( pushButtonGadget6, "   Graphs" );
UxPutWidth( pushButtonGadget6, 86 );
UxPutHeight( pushButtonGadget6, 32 );
UxCreateWidget( pushButtonGadget6 );

/* Creation of frame21 */
frame21 = UxCreateFrame( "frame21", form9 );
UxPutWidth( frame21, 470 );
UxPutHeight( frame21, 115 );
UxPutX( frame21, 2 );
UxPutY( frame21, 6 );
UxPutShadowThickness( frame21, 3 );
UxPutShadowType( frame21, "shadow_out" );
UxCreateWidget( frame21 );

/* Creation of rowColumn10 */
rowColumn10 = UxCreateRowColumn( "rowColumn10", frame21 );
UxPutWidth( rowColumn10, 464 );
UxPutHeight( rowColumn10, 109 );
UxPutX( rowColumn10, 3 );
UxPutY( rowColumn10, 4 );
UxCreateWidget( rowColumn10 );

/* Creation of frame22 */
frame22 = UxCreateFrame( "frame22", rowColumn10 );
UxPutWidth( frame22, 458 );
UxPutHeight( frame22, 99 );
UxPutX( frame22, 3 );
UxPutY( frame22, 3 );
UxPutShadowType( frame22, "shadow_out" );
UxPutShadowThickness( frame22, 3 );
UxCreateWidget( frame22 );

/* Creation of rowColumn11 */
rowColumn11 = UxCreateRowColumn( "rowColumn11", frame22 );
UxPutWidth( rowColumn11, 452 );
UxPutHeight( rowColumn11, 93 );
UxPutX( rowColumn11, 3 );
UxPutY( rowColumn11, 3 );
UxCreateWidget( rowColumn11 );

/* Creation of labelGadget1 */
labelGadget1 = UxCreateLabelGadget( "labelGadget1", rowColumn11 );
UxPutX( labelGadget1, 3 );
UxPutY( labelGadget1, 3 );
UxPutLabelString( labelGadget1, "\n          SOFTEKS RESEARCH LABORATORY \n
                  Software Engineering and Knowledge-base Systems \n
                        Department of Computer Science \n
                              Concordia University \n\n
                                     Implemented \n
```

```
                                    by \n"
                                Joseph Said. \n");
UxPutFontList( labelGadget1, "-adobe-courier-bold-o-normal--14-140-75-75-m-90-iso8859-1" );
UxPutHeight( labelGadget1, 99 );
UxPutAlignment( labelGadget1, "alignment_center" );
UxCreateWidget( labelGadget1 );

/* Creation of pushButtonGadget25 */
pushButtonGadget25 = UxCreatePushButtonGadget( "pushButtonGadget25", rowColumn11 );
UxPutX( pushButtonGadget25, 5 );
UxPutY( pushButtonGadget25, 30 );
UxPutLabelString( pushButtonGadget25, "" );
UxCreateWidget( pushButtonGadget25 );

/* Creation of frame23 */
frame23 = UxCreateFrame( "frame23", form9 );
UxPutWidth( frame23, 110 );
UxPutHeight( frame23, 98 );
UxPutX( frame23, 6 );
UxPutY( frame23, 243 );
UxPutShadowThickness( frame23, 3 );
UxPutShadowType( frame23, "shadow_out" );
UxCreateWidget( frame23 );

/* Creation of rowColumn12 */
rowColumn12 = UxCreateRowColumn( "rowColumn12", frame23 );
UxPutWidth( rowColumn12, 104 );
UxPutHeight( rowColumn12, 97 );
UxPutX( rowColumn12, 3 );
UxPutY( rowColumn12, 3 );
UxCreateWidget( rowColumn12 );

/* Creation of frame24 */
frame24 = UxCreateFrame( "frame24", rowColumn12 );
UxPutWidth( frame24, 98 );
UxPutHeight( frame24, 85 );
UxPutX( frame24, 3 );
UxPutY( frame24, 3 );
UxPutShadowType( frame24, "shadow_out" );
UxPutShadowThickness( frame24, 3 );
UxCreateWidget( frame24 );

/* Creation of rowColumn13 */
rowColumn13 = UxCreateRowColumn( "rowColumn13", frame24 );
UxPutWidth( rowColumn13, 64 );
UxPutHeight( rowColumn13, 79 );
UxPutX( rowColumn13, 12 );
UxPutY( rowColumn13, 0 );
UxCreateWidget( rowColumn13 );

/* Creation of pushButtonGadget9 */
pushButtonGadget9 = UxCreatePushButtonGadget( "pushButtonGadget9", rowColumn13 );
UxPutX( pushButtonGadget9, 0 );
UxPutY( pushButtonGadget9, 0 );
UxPutHeight( pushButtonGadget9, 46 );
UxPutShadowThickness( pushButtonGadget9, 1 );
UxPutAlignment( pushButtonGadget9, "alignment_end" );
UxPutWidth( pushButtonGadget9, 469 );
UxPutLabelString( pushButtonGadget9, "" );
UxCreateWidget( pushButtonGadget9 );

/* Creation of pushButtonGadget10 */
pushButtonGadget10 = UxCreatePushButtonGadget( "pushButtonGadget10", rowColumn13 );
UxPutX( pushButtonGadget10, 3 );
```

```
UxPutY( pushButtonGadget10, 52 );
UxPutLabelString( pushButtonGadget10, "   Help..." );
UxPutWidth( pushButtonGadget10, 86 );
UxPutHeight( pushButtonGadget10, 32 );
UxCreateWidget( pushButtonGadget10 );

/* Creation of separator2 */
separator2 = UxCreateSeparator( "separator2", form8 );
UxPutWidth( separator2, 220 );
UxPutHeight( separator2, 20 );
UxPutX( separator2, 288 );
UxPutY( separator2, 8 );
UxCreateWidget( separator2 );

/* Creation of label21 */
label21 = UxCreateLabel( "label21", form8 );
UxPutX( label21, 64 );
UxPutY( label21, 8 );
UxPutLabelString( label21, "Connection to INGRES ..." );
UxPutFontList( label21, "-adobe-courier-bold-r-normal--14-100-100-100-m-90-iso8859-1" );
UxCreateWidget( label21 );

/* Creation of separator3 */
separator3 = UxCreateSeparator( "separator3", form8 );
UxPutWidth( separator3, 56 );
UxPutHeight( separator3, 20 );
UxPutX( separator3, 4 );
UxPutY( separator3, 8 );
UxCreateWidget( separator3 );

UxDelayUpdate( pushButtonGadget11 );
UxPutArmPixmap( pushButtonGadget11, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget11, "udbms.xpm" );
UxPutLabelType( pushButtonGadget11, "pixmap" );
UxUpdate( pushButtonGadget11 );
UxDelayUpdate( pushButtonGadget14 );
UxPutArmPixmap( pushButtonGadget14, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget14, "connect.xpm" );
UxPutLabelType( pushButtonGadget14, "pixmap" );
UxUpdate( pushButtonGadget14 );
UxDelayUpdate( pushButtonGadget15 );
UxPutArmPixmap( pushButtonGadget15, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget15, "connect.xpm" );
UxPutLabelType( pushButtonGadget15, "pixmap" );
UxUpdate( pushButtonGadget15 );
UxDelayUpdate( pushButtonGadget21 );
UxPutArmPixmap( pushButtonGadget21, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget21, "query.xpm" );
UxPutLabelType( pushButtonGadget21, "pixmap" );
UxUpdate( pushButtonGadget21 );
UxDelayUpdate( pushButtonGadget22 );
UxPutArmPixmap( pushButtonGadget22, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget22, "scrltext.xpm" );
UxPutLabelType( pushButtonGadget22, "pixmap" );
UxUpdate( pushButtonGadget22 );
UxDelayUpdate( pushButtonGadget12 );
UxPutArmPixmap( pushButtonGadget12, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget12, "cview.xpm" );
UxPutLabelType( pushButtonGadget12, "pixmap" );
UxUpdate( pushButtonGadget12 );
UxDelayUpdate( pushButtonGadget16 );
UxPutArmPixmap( pushButtonGadget16, "workingD.xpm" );
UxPutLabelPixmap( pushButtonGadget16, "fview.xpm" );
UxPutLabelType( pushButtonGadget16, "pixmap" );
```

```
        UxUpdate( pushButtonGadget16 );
        UxDelayUpdate( pushButtonGadget26 );
        UxPutArmPixmap( pushButtonGadget26, "workingD.xpm" );
        UxPutLabelPixmap( pushButtonGadget26, "relcal.xpm" );
        UxPutLabelType( pushButtonGadget26, "pixmap" );
        UxUpdate( pushButtonGadget26 );
        UxDelayUpdate( pushButtonGadget1 );
        UxPutArmPixmap( pushButtonGadget1, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget1, "pixmap" );
        UxPutLabelPixmap( pushButtonGadget1, "action.xpm" );
        UxUpdate( pushButtonGadget1 );
        UxDelayUpdate( pushButtonGadget3 );
        UxPutArmPixmap( pushButtonGadget3, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget3, "pixmap" );
        UxPutLabelPixmap( pushButtonGadget3, "bboard.xpm" );
        UxUpdate( pushButtonGadget3 );
        UxDelayUpdate( pushButtonGadget7 );
        UxPutArmPixmap( pushButtonGadget7, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget7, "pixmap" );
        UxPutLabelPixmap( pushButtonGadget7, "text.xpm" );
        UxUpdate( pushButtonGadget7 );
        UxDelayUpdate( pushButtonGadget5 );
        UxPutArmPixmap( pushButtonGadget5, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget5, "pixmap" );
        UxPutLabelPixmap( pushButtonGadget5, "unknown.xpm" );
        UxUpdate( pushButtonGadget5 );
        UxDelayUpdate( pushButtonGadget25 );
        UxPutArmPixmap( pushButtonGadget25, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget25, "pixmap" );
        UxUpdate( pushButtonGadget25 );
        UxDelayUpdate( pushButtonGadget9 );
        UxPutArmPixmap( pushButtonGadget9, "workingD.xpm" );
        UxPutLabelType( pushButtonGadget9, "pixmap" );
        UxPutLabelPixmap( pushButtonGadget9, "informD.xpm" );
        UxUpdate( pushButtonGadget9 );

        /* UxRealizeInterface creates the X windows for the widgets above. */

        UxRealizeInterface( UncertainDatabase );

        return ( UncertainDatabase );
}

/*-----------------------------------------------------------------------------------------
**      The following is the 'Interface function' which is the
**      external entry point for creating this interface.
**      This function should be called from your application or from
**      a callback function.
**-----------------------------------------------------------------------------------------*/

swidget create_UncertainDatabase( _UxUxParent )
        swidget _UxUxParent;
{
        swidget                 rtrn;
        static int _Uxinit = 0;

        UxParent = _UxUxParent;

        if ( ! _Uxinit )
        {
         static XtActionsRec _Uxactions[] = {
         { "doubleclick", (XtActionProc) action_doubleclick }};

         XtAppAddActions( UxAppContext,
          _Uxactions,
          XtNumber(_Uxactions) );

          _Uxinit = 1;
         }

         {
          printf(" I am in the initial code of the bulletinboard popup...\n");
          rtrn = _Uxbuild_UncertainDatabase();

          printf("I am in the Final code of the bulletinboard popup \n");
          return(rtrn);
         }
}

/*******************************************************************************
       END OF FILE
*******************************************************************************/



/*******************************************************************************
** QueryEditor.c
**
**      Associated Header file: QueryEditor.h
*******************************************************************************/

#include <stdio.h>
#include "UxLib.h"
#include "UxCascB.h"
#include "UxSep.h"
#include "UxPushB.h"
#include "UxRowCol.h"
#include "UxLabel.h"
#include "UxComm.h"
#include "UxFsBox.h"
#include "UxMainW.h"



static swidget fileSelectionBox1;
static swidget command1;
static swidget label1;
static swidget menu1;
static swidget menu1_p1;
static swidget menu1_p1_b1;
static swidget menu1_p1_b2;
static swidget menu1_p1_b4;
static swidget menu1_p1_b5;
static swidget menu1_p1_b6;
static swidget menu1_p1_b7;
static swidget menu1_p1_b9;
static swidget menu1_top_b1;
static swidget UxParent;

#define CONTEXT_MACRO_ACCESS 1
#include "QueryEditor.h"
#undef CONTEXT_MACRO_ACCESS

swidget QueryEditor;

/*------------------------------------------------------------------------------
       The following are callback functions.
------------------------------------------------------------------------------*/
```

```
static void activateCB_menu1_p1_b5( UxWidget, UxClientData, UxCallbackArg )
Widget UxWidget;
XtPointer UxClientData, UxCallbackArg;

{
swidget UxThisWidget;

UxThisWidget = UxWidgetToSwidget( UxWidget );

{
   FILE *myfileptr;
   FILE *newfileptr;
   int inchar;

   myfileptr = fopen("myfile.qry", "r");
   newfileptr = fopen("newfile", "w");
   while ((inchar = getc(myfileptr)) != EOF)
   {
    putc(inchar, newfileptr);
   }
  fclose(myfileptr);
  fclose(newfileptr);
}
}

static void activateCB_menu1_p1_b6( UxWidget, UxClientData, UxCallbackArg )
Widget UxWidget;
XtPointer UxClientData, UxCallbackArg;

{
swidget UxThisWidget;

UxThisWidget = UxWidgetToSwidget( UxWidget );

{
   FILE *myfileptr;
   FILE *newfileptr;
   int inchar, i, n;
   char qry[500];

   myfileptr = fopen("myfile.qry", "r");
   newfileptr = fopen("newfile", "w");
   i = 0;
   while ((inchar = getc(myfileptr)) != EOF)
   {
    if ( inchar !=  '\n')
      qry[i++] = tolower(inchar);
    else
     {
      qry[i++] = tolower(',');
      qry[i++] = tolower(' ');
     }
    putc(inchar, newfileptr);
   }
  qry[i-2] = tolower('\0');
  n = i-2;
  for ( i = 0; i <= n; i++)
    printf("%c", qry[i]);
  UxPutListItemCount(command1, 2);
  UxPutListItems(command1, qry);

  fclose(myfileptr);
  fclose(newfileptr);
}
}

static void activateCB_menu1_p1_b9( UxWidget, UxClientData, UxCallbackArg )
Widget UxWidget;
XtPointer UxClientData, UxCallbackArg;

{
swidget UxThisWidget;

UxThisWidget = UxWidgetToSwidget( UxWidget );

{
extern swidget QueryEditor;
UxPopdownInterface(QueryEditor, no_grab);

}
}

/*------------------------------------------------------------------------------------------
**      The 'build_' function creates all the swidgets and X widgets,
**      and sets their properties to the values specified in the
**      Property Editor.
**------------------------------------------------------------------------------------------*/


static swidget _Uxbuild_QueryEditor()
{
/* Create the swidgets */


/* Creation of QueryEditor */
QueryEditor = UxCreateMainWindow( "QueryEditor", UxParent );
UxPutDefaultShell( QueryEditor, "transientShell" );

UxPutWidth( QueryEditor, 400 );
UxPutHeight( QueryEditor, 700 );
UxPutX( QueryEditor, 589 );
UxPutY( QueryEditor, 175 );
UxPutUnitType( QueryEditor, "pixels" );
UxPutForeground( QueryEditor, "white" );
UxCreateWidget( QueryEditor );


/* Creation of fileSelectionBox1 */
fileSelectionBox1 = UxCreateFileSelectionBox( "fileSelectionBox1", QueryEditor );
UxPutApplyLabelString( fileSelectionBox1, "Filter Files" );
UxPutButtonFontList( fileSelectionBox1,
                                "-adobe-helvetica-bold-o-normal--13-180-75-75-p-104-iso8859-1" );
UxPutCancelLabelString( fileSelectionBox1, "Cancel Selection" );
UxPutDirListLabelString( fileSelectionBox1, "Directories Available" );
UxPutFileListLabelString( fileSelectionBox1, "Query Files" );
UxPutForeground( fileSelectionBox1, "white" );
UxPutHelpLabelString( fileSelectionBox1, "Help ..." );
UxPutListLabelString( fileSelectionBox1, "Query Files" );
UxPutNoMatchString( fileSelectionBox1, " [ NO MATCH ] " );
UxPutOkLabelString( fileSelectionBox1, "OK Selection " );
UxPutTextFontList( fileSelectionBox1,
                        "-misc-fixed-medium-r-semicondensed--13-100-100-100-c-60-iso8859-1" );
UxPutWidth( fileSelectionBox1, 500 );
UxPutLabelFontList( fileSelectionBox1,
                              "-adobe-helvetica-bold-o-normal--17-120-100-100-p-92-iso8859-1" );
UxPutDirMask( fileSelectionBox1, "*.qry" );
UxCreateWidget( fileSelectionBox1 );
```

```
/* Creation of command1 */
command1 = UxCreateCommand( "command1", QueryEditor );
UxPutButtonFontList( command1,
                        "-adobe-helvetica-bold-o-normal--18-180-75-75-p-104-iso8859-1" );
UxPutForeground( command1, "white" );
UxPutLabelFontList( command1,
                        "-adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859-1" );
UxPutPromptString( command1, ">Enter a New Query or Select a Previous One ..." );
UxPutTextFontList( command1, "-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1" );
UxPutListItemCount( command1, 2 );
UxPutTextString( command1, "" );
UxPutHistoryItemCount( command1, 0 );
UxPutListLabelString( command1, "" );
UxPutCommand( command1, "" );
UxPutListItems( command1, "Item 1, Item 2" );
UxCreateWidget( command1 );


/* Creation of label1 */
label1 = UxCreateLabel( "label1", QueryEditor );
UxCreateWidget( label1 );


/* Creation of menu1 */
menu1 = UxCreateRowColumn( "menu1", QueryEditor );
UxPutRowColumnType( menu1, "menu_bar" );
UxPutMenuAccelerator( menu1, "<KeyUp>F10" );
UxCreateWidget( menu1 );


/* Creation of menu1_p1 */
menu1_p1 = UxCreateRowColumn( "menu1_p1", menu1 );
UxPutRowColumnType( menu1_p1, "menu_pulldown" );
UxCreateWidget( menu1_p1 );


/* Creation of menu1_p1_b1 */
menu1_p1_b1 = UxCreatePushButton( "menu1_p1_b1", menu1_p1 );
UxPutLabelString( menu1_p1_b1, "New Query File" );
UxPutMnemonic( menu1_p1_b1, "N" );
UxPutForeground( menu1_p1_b1, "white" );
UxCreateWidget( menu1_p1_b1 );

/* Creation of menu1_p1_b2 */
menu1_p1_b2 = UxCreatePushButton( "menu1_p1_b2", menu1_p1 );
UxPutLabelString( menu1_p1_b2, "Open Query File" );
UxPutMnemonic( menu1_p1_b2, "O" );
UxPutForeground( menu1_p1_b2, "white" );
UxCreateWidget( menu1_p1_b2 );


/* Creation of menu1_p1_b4 */
menu1_p1_b4 = UxCreateSeparator( "menu1_p1_b4", menu1_p1 );
UxPutForeground( menu1_p1_b4, "white" );
UxCreateWidget( menu1_p1_b4 );


/* Creation of menu1_p1_b5 */
menu1_p1_b5 = UxCreatePushButton( "menu1_p1_b5", menu1_p1 );
UxPutLabelString( menu1_p1_b5, "Save Query File" );
UxPutMnemonic( menu1_p1_b5, "S" );
UxPutForeground( menu1_p1_b5, "white" );
UxCreateWidget( menu1_p1_b5 );
```

```
UxAddCallback( menu1_p1_b5, XmNactivateCallback,
(XtCallbackProc) activateCB_menu1_p1_b5,
(XtPointer) NULL );


/* Creation of menu1_p1_b6 */
menu1_p1_b6 = UxCreatePushButton( "menu1_p1_b6", menu1_p1 );
UxPutLabelString( menu1_p1_b6, "Save Query File As ..." );
UxPutMnemonic( menu1_p1_b6, "A" );
UxPutForeground( menu1_p1_b6, "white" );
UxCreateWidget( menu1_p1_b6 );

UxAddCallback( menu1_p1_b6, XmNactivateCallback,
(XtCallbackProc) activateCB_menu1_p1_b6,
(XtPointer) NULL );


/* Creation of menu1_p1_b7 */
menu1_p1_b7 = UxCreateSeparator( "menu1_p1_b7", menu1_p1 );
UxPutForeground( menu1_p1_b7, "white" );
UxCreateWidget( menu1_p1_b7 );

/* Creation of menu1_p1_b9 */
menu1_p1_b9 = UxCreatePushButton( "menu1_p1_b9", menu1_p1 );
UxPutLabelString( menu1_p1_b9, "Exit Query Editor" );
UxPutMnemonic( menu1_p1_b9, "E" );
UxPutForeground( menu1_p1_b9, "white" );
UxCreateWidget( menu1_p1_b9 );

UxAddCallback( menu1_p1_b9, XmNactivateCallback,
(XtCallbackProc) activateCB_menu1_p1_b9,
(XtPointer) NULL );


/* Creation of menu1_top_b1 */
menu1_top_b1 = UxCreateCascadeButton( "menu1_top_b1", menu1 );
UxPutLabelString( menu1_top_b1, "QueryFiles" );
UxPutMnemonic( menu1_top_b1, "F" );
UxPutForeground( menu1_top_b1, "white" );
UxPutSubMenuId( menu1_top_b1, "menu1_p1" );
UxCreateWidget( menu1_top_b1 );


UxMainWindowSetAreas( QueryEditor, menu1, command1,
NULL_SWIDGET, NULL_SWIDGET, fileSelectionBox1 );
UxMainWindowSetMessageWindow( QueryEditor. label1 );
return ( QueryEditor );
}

/*---------------------------------------------------------------------------------------
**      The following is the 'Interface function' which is the
**      external entry point for creating this interface.
**      This function should be called from your application or from
**      a callback function.
**-------------------------------------------------------------------------------------*/

swidget create_QueryEditor( _UxUxParent )
swidget _UxUxParent;
{
swidget                 rtrn;

UxParent = _UxUxParent;
```

```
rtrn = _Uxbuild_QueryEditor();

return(rtrn);
}

/*------------------------------------------------------------------------------------------
**      END OF FILE
**-----------------------------------------------------------------------------------------*/




/*******************************************************************************************
**
**      ScrolledText.c
**
**      Associated Header file: ScrolledText.h
********************************************************************************************/


#include <stdio.h>
#include "UxLib.h"
#include "UxPushB.h"
#include "UxRowCol.h"
#include "UxScText.h"
#include "UxScrW.h"
#include "UxLabel.h"
#include "UxFrame.h"
#include "UxForm.h"
#include "UxApplSh.h"



static swidget form1;
static swidget form2;
static swidget frame1;
static swidget label2;
static swidget frame2;
static swidget frame3;
static swidget scrolledWindowText1;
static swidget scrolledText1;
static swidget frame4;
static swidget rowColumn1;
static swidget pushButton9;
static swidget pushButton5;
static swidget pushButton6;
static swidget UxParent;
static char *label1;

#define CONTEXT_MACRO_ACCESS 1
#include "ScrolledText.h"
#undef CONTEXT_MACRO_ACCESS

swidget ScrolledText;

/*------------------------------------------------------------------------------------------
        The following are callback functions.
**-----------------------------------------------------------------------------------------*/

static void activateCB_pushButton6( UxWidget, UxClientData, UxCallbackArg )
Widget UxWidget;
XtPointer UxClientData, UxCallbackArg;
```

```
{
swidget UxThisWidget;

UxThisWidget = UxWidgetToSwidget( UxWidget );

{
extern swidget ScrolledText;
UxPopdownInterface(ScrolledText, no_grab);

}
}

/*--------------------------------------------------------------------------------------------
**      The 'build_' function creates all the swidgets and X widgets,
**      and sets their properties to the values specified in the
**      Property Editor.
**-------------------------------------------------------------------------------------------*/

static swidget _Uxbuild_ScrolledText()
{
/* Create the swidgets */


/* Creation of ScrolledText */
ScrolledText = UxCreateApplicationShell( "ScrolledText", UxParent );

UxPutWidth( ScrolledText, 583 );
UxPutHeight( ScrolledText, 550 );
UxPutX( ScrolledText, 320 );
UxPutY( ScrolledText, 16 );
UxCreateWidget( ScrolledText );


/* Creation of form1 */
form1 = UxCreateForm( "form1", ScrolledText );
UxPutWidth( form1, 581 );
UxPutHeight( form1, 446 );
UxPutResizePolicy( form1, "resize_none" );
UxPutX( form1, 28 );
UxPutY( form1, 36 );
UxPutUnitType( form1, "pixels" );
UxCreateWidget( form1 );


/* Creation of form2 */
form2 = UxCreateForm( "form2", form1 );
UxPutWidth( form2, 575 );
UxPutHeight( form2, 33 );
UxPutResizePolicy( form2, "resize_none" );
UxPutX( form2, 4 );
UxPutY( form2, 6 );
UxCreateWidget( form2 );

/* Creation of frame1 */
frame1 = UxCreateFrame( "frame1", form2 );
UxPutWidth( frame1, 575 );
UxPutHeight( frame1, 31 );
UxPutX( frame1, 0 );
UxPutY( frame1, 1 );
UxCreateWidget( frame1 );


/* Creation of label2 */
label2 = UxCreateLabel( "label2", frame1 );
```

```
UxPutX( label2, 2 );
UxPutY( label2, 2 );
UxPutHeight( label2, 27 );
UxPutWidth( label2, 572 );
UxPutForeground( label2, "white" );
UxPutLabelString( label2, "Query Answers" );
UxCreateWidget( label2 );


/* Creation of frame2 */
frame2 = UxCreateFrame( "frame2", form1 );
UxPutWidth( frame2, 576 );
UxPutHeight( frame2, 408 );
UxPutX( frame2, 4 );
UxPutY( frame2, 40 );
UxCreateWidget( frame2 );


/* Creation of frame3 */
fram[illegible]ateFrame( "frame3", frame2 );
UxPutWidth( frame3, 200 );
UxPutHeight( frame3, 192 );
UxPutX( frame3, 36 );
UxPutY( frame3, 312 );
UxCreateWidget( frame3 );


/* Creation of scrolledWindowText1 */
scrolledWindowText1 = UxCreateScrolledWindow( "scrolledWindowText1", frame3 );
UxPutScrollingPolicy( scrolledWindowText1, "application_defined" );
UxPutVisualPolicy( scrolledWindowText1, "variable" );
UxPutScrollBarDisplayPolicy( scrolledWindowText1, "static" );
UxPutX( scrolledWindowText1, 8 );
UxPutY( scrolledWindowText1, 8 );
UxCreateWidget( scrolledWindowText1 );

/* Creation of scrolledText1 */
scrolledText1 = UxCreateScrolledText( "scrolledText1", scrolledWindowText1 );
UxPutWidth( scrolledText1, 200 );
UxPutHeight( scrolledText1, 200 );
UxPutEditMode( scrolledText1, "multi_line_edit" );
UxPutText( scrolledText1, "Hello \nAnother Hello" );
UxCreateWidget( scrolledText1 );


/* Creation of frame4 */
frame4 = UxCreateFrame( "frame4", form1 );
UxPutWidth( frame4, 576 );
UxPutHeight( frame4, 96 );
UxPutX( frame4, 4 );
UxPutY( frame4, 452 );
UxCreateWidget( frame4 );


/* Creation of rowColumn1 */
rowColumn1 = UxCreateRowColumn( "rowColumn1", frame4 );
UxPutWidth( rowColumn1, 554 );
UxPutHeight( rowColumn1, 92 );
UxPutX( rowColumn1, 2 );
UxPutY( rowColumn1, 2 );
UxCreateWidget( rowColumn1 );


/* Creation of pushButton9 */
```

```
pushButton9 = UxCreatePushButton( "pushButton9", rowColumn1 );
UxPutX( pushButton9, 66 );
UxPutY( pushButton9, 2 );
UxPutForeground( pushButton9, "white" );
UxPutLabelString( pushButton9, label1 );
UxCreateWidget( pushButton9 );


/* Creation of pushButton5 */
pushButton5 = UxCreatePushButton( "pushButton5", rowColumn1 );
UxPutX( pushButton5, 66 );
UxPutY( pushButton5, 30 );
UxPutForeground( pushButton5, "white" );
UxPutLabelString( pushButton5, "pushbutton5" );
UxPutHeight( pushButton5, 25 );
UxCreateWidget( pushButton5 );


/* Creation of pushButton6 */
pushButton6 = UxCreatePushButton( "pushButton6", rowColumn1 );
UxPutX( pushButton6, 4 );
UxPutY( pushButton6, 60 );
UxPutLabelString( pushButton6, "Exit" );
UxPutForeground( pushButton6, "white" );
UxPutHeight( pushButton6, 30 );
UxCreateWidget( pushButton6 );

UxAddCallback( pushButton6, XmNactivateCallback,
(XtCallbackProc) activateCB_pushButton6,
(XtPointer) NULL );

UxDelayUpdate( pushButton5 );
UxPutPositionIndex( pushButton5, 1 );
UxUpdate( pushButton5 );


/* UxRealizeInterface creates the X windows for the widgets above. */

UxRealizeInterface( ScrolledText );

return ( ScrolledText );
}

/*------------------------------------------------------------------------------------------
**      The following is the 'Interface function' which is the
**      external entry point for creating this interface.
**      This function should be called from your application or from
**      a callback function.
**-----------------------------------------------------------------------------------------*/
swidget create_ScrolledText( _UxUxParent, _Uxlabel1 )
swidget _UxUxParent;
char *_Uxlabel1;
{
swidget                 rtrn;

UxParent = _UxUxParent;
label1 = _Uxlabel1;

rtrn = _Uxbuild_ScrolledText();

/* to print in the text field you have to concatenate strings together */
UxPutText(scrolledText1, " ");
/* you can not say multi Uxputs */
/* UxPutText(scrolledText1, "\n This is the second line entered using UxPutText...\n"); */
```

```
return(rtrn);
}

/*******************************************************************************************************
      END OF FILE
*******************************************************************************************************/
```

8.2 Include files

```
/*******************************************************************************
**      QueryEditor.h
**      This header file is included by QueryEditor.c
**
*******************************************************************************/

#ifndef _QUERYEDITOR_INCLUDED
#define _QUERYEDITOR_INCLUDED


#include <stdio.h>
#include "UxLib.h"
#include "UxCascB.h"
#include "UxSep.h"
#include "UxPushB.h"
#include "UxRowCol.h"
#include "UxLabel.h"
#include "UxComm.h"
#include "UxFsBox.h"
#include "UxMainW.h"

extern swidget QueryEditor;

/*******************************************************************************
**      Declarations of global functions.
*******************************************************************************/

swidget create_QueryEditor();

#endif /* _QUERYEDITOR_INCLUDED */


/*******************************************************************************
**      ScrolledText.h
**      This header file is included by ScrolledText.c
**
*******************************************************************************/

#ifndef _SCROLLEDTEXT_INCLUDED
#define _SCROLLEDTEXT_INCLUDED


#include <stdio.h>
#include "UxLib.h"
#include "UxPushB.h"
#include "UxRowCol.h"
#include "UxScText.h"
#include "UxScrW.h"
#include "UxLabel.h"
#include "UxFrame.h"
#include "UxForm.h"
#include "UxApplSh.h"
```

```
extern swidget ScrolledText;

/*******************************************************************************
**      Declarations of global functions.
*******************************************************************************/

swidget create_ScrolledText();

#endif /* _SCROLLEDTEXT_INCLUDED */


/*******************************************************************************
**      UncertainDatabase.h
**      This header file is included by UncertainDatabase.c
**
*******************************************************************************/

#ifndef _UNCERTAINDATABASE_INCLUDED
#define _UNCERTAINDATABASE_INCLUDED


#include <stdio.h>
#include "UxLib.h"
#include "UxLabelG.h"
#include "UxRowCol.h"
#include "UxTogB.h"
#include "UxScText.h"
#include "UxScrW.h"
#include "UxScale.h"
#include "UxPushBG.h"
#include "UxSep.h"
#include "UxArrB.h"
#include "UxText.h"
#include "UxLabel.h"
#include "UxForm.h"
#include "UxFrame.h"
#include "UxPushB.h"
#include "UxBboard.h"

/*******************************************************************************
**      Declarations of global functions.
*******************************************************************************/

swidget create_UncertainDatabase();

#endif /* _UNCERTAINDATABASE_INCLUDED */
```

8.3 ESQL code

```
#include <stdio.h>
#include <ctype.h>

/*
** defined values for Structured Query Template
*/
#define MAXATTR   200                 /* maximum number of regular attributes         */
#define ATTWIDTH  20                  /* maximum width of an attribute name           */
```

```
#define MAXTABLES 200                     /* maximum number of tables                      */
#define TABWIDTH  20                      /* maximum table width name                      */
#define MAXCONJ   200                     /* maximum number of source attributes           */
#define CONJWIDTH 100                     /* maximum width of a source attribute name      */
#define MAXVEC    3                       /* maximum number of source vectors for Algorithm 1 */
#define MAXSOURCES 10                     /* Maximum number of sources in the prototype    */
#define NROWS     100                     /* number of tuples in an answer to a query      */
#define WIDTH     50                      /* width of an attribute value                   */
#define MAXDIN    50                      /* maximum number of disjunctions                */
#define MAXCON    50                      /* maximum number of conjuncts                    */
#define MAXLINE   300                     /* maximum line width                            */
#define MAXTUPLES 1000                    /* Maximum number of tuples to sort              */



/*
** Declare the SQLCA structure and the SQLDA typedef
*/

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

EXEC SQL DECLARE stmt STATEMENT; /* Dynamic SQL Statement */
EXEC SQL DECLARE csr CURSOR FOR stmt; /* Cursor for dynamic AQL statement */

/*
** Declare of result columns for a dynamic SELECT. If a SELECT
** statement returns more than DEF_ELEMS, then a new SQLDA will be allocated.
*/

# define DEF_ELEMS 5
# define DATE_SIZE 25                     /* Size of a DATE string variable                 */
# define SQL_NOTFOUND 100                 /* The SQL code for the NOT FOUND condition      */
# define DBNAME_MAX 50                    /* Maximum Database name                         */
# define INPUT_SIZE 256                   /* Max input line size                           */
# define STMT_MAX 1000                    /* Max SQL statement length                      */

/*
** Global SQL variables
*/
IISQLDA  *sqlda = (IISQLDA *)0;           /* Pointer to the SQL dynamic area               */

/*
** Result storage buffer for dynamic SELECT statement
*/
struct {
    int   res_length; /* Size of meme_data */
    char  *res_data;  /* Pointer to allocated result buffer */
} res_buf = {0, NULL};

/*
** Structured Query Template
*/
 struct  {
    int   nattr;                                         /* number of attributes read */
    char  attr[MAXATTR][ATTWIDTH];                       /* attribute list            */
    int   ntbls;                                         /* number of tables read     */
    char  tbls[MAXTABLES][TABWIDTH];                     /* tables list               */
    int   ncnd;                                          /* number of conjuncts       */
    char  cnd[MAXCONJ][CONJWIDTH];                       /* conjuncts list            */
    int   qtype;                                         /* is the query type         */
 } qry ;
```

```
/*
** array that defines the type of each attribute name.
*/
int t[MAXATTR];

/*
** count the number of duplicate rows in any set of query answers.
*/
int drows;

/*
** Reliability of source vectors
*/
float re[MAXSOURCES];



/*
** Define the Query Syntax
*/
char  slct[7];                                    /* select                              */
char    fm[5];                                    /* form                                */
char   whr[6];                                    /* where                               */
char    un[6];                                    /* union                               */

/*
** functions and procedure needed
*/
void   Run_Application();                         /* Run SQL Monitor                     */
void   Init_Sqlda();                              /* Initialize SQLDA                    */
void   Print_Scheme();                            /* Print SELECT column header          */
void   Print_Tuples();                            /* Print SELECT Row values             */
void   Print_Error();                             /* Print a user error                  */
void   Conjunction();                             /* w = u /\ v                          */
void   ExtendedSelection();                       /* Extended Selection                  */
void   ExtendedProjection();                      /* Extended Projection                 */
void   ExtendedUnion();                           /* Extended Union                      */
void   ExtendedProduct();                         /* Extended Cartesian Product          */
void   ExtendedJoin();                            /* Extended Join                       */
void   ExtendedIntersection();                    /* Extended Intersection               */
void   ExtendedDifference();                      /* Extended Set Difference             */
void   RelCalculation();                          /* Reliability of single source vector */
void   SortFile();                                /* Sort lines in file in increasing order */
char   *Read_Query();                             /* Read statement from terminal        */
char   *Alloc_Mem();                              /* Allocate memory                     */
char   *calloc();                                 /* C allocation routine                */
char   *FillStructuredQueryTemplate();            /* Read statement from terminal        */
char   *QueryTransformation();                    /* Extended model Query Transformation */
int    CmpSourceVectors();                        /* Compares two source vectors         */
int    EqualStrings();                            /* Test if two strings are equal       */
int    Exectute_Query();                          /* Execute Dynamic Select              */
float  Algorithm1();                              /* Reliability Calculation algorithm 1 */
float  SourceVectorReliability();                 /* reliability of a single source vector */
double str2flt();                                 /* String to float conversion          */


/*------------------------------------------------------------------------------------
** Procedure : main
** Purpose:    Main body of SQL Monitor application. Prompts for database
**             name and connect to the database. Run the monitor and
**             disconnect from the database. Before disconnecting roll
**             back any pending updates.
** Parameters: None
**------------------------------------------------------------------------------------------*/
main()
{
  /*
  ** pointer to the "options.rel" file that contains the information
  ** sources reliability.
  */
  FILE *ifileptr;
  /*
  ** In the 'options.rel" file there is the reliability calculation flag
  ** integer variable and we would like to have it unchanged. by this
  ** procedure. Skip this variable and read the information source
  ** reliabilities.
  */
  int relFlag, i;
  int rel1,rel2,rel3,rel4,rel5,rel6,rel7,rel8,rel9,rel10;

  EXEC SQL BEGIN DECLARE SECTION;
     char dbname[DBNAME_MAX +1];                      /* Database name                       */
  EXEC SQL END DECLARE SECTION;

  /*
  ** Initialization of query Syntax (Select ... From ... Where ...)
  */
  slct[1] = 's'; slct[2] = 'e'; slct[3] = 'l'; slct[4] = 'e';
  slct[5] = 'c'; slct[6] = 't'; slct[7] = '\0';

  fm[1] = 'f'; fm[2] = 'r'; fm[3] = 'o'; fm[4] = 'm'; fm[5] = '\0';

  whr[1] = 'w'; whr[2] = 'h'; whr[3] = 'e';
  whr[4] = 'r'; whr[5] = 'e'; whr[6] = '\0';

  un[1] = 'u'; un[2] = 'n'; un[3] = 'i';
  un[4] = 'o'; un[5] = 'n'; un[6] = '\0';

 /*
 ** Open the 'options.rel' file and read the reliability calculation
 ** flag as well as the reliability of information sources.
 */
 ifileptr = fopen("options.rel", "r");

 /*
 ** skip relFlag (reliability calculation flag).
 */
 fscanf(ifileptr, "%d", &relFlag);
 fscanf(ifileptr,"%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",&rel1,&rel2,&rel3,&rel4,&rel5,
                                                &rel6,&rel7,&rel8,&rel9,&rel10);
 fclose(ifileptr);
 re[1] = rel1 * 0.01; re[2] = rel2 * 0.01; re[3] = rel3 * 0.01;
 re[4] = rel4 * 0.01; re[5] = rel5 * 0.01; re[6] = rel6 * 0.01;
 re[7] = rel7 * 0.01; re[8] = rel8 * 0.01; re[9] = rel9 * 0.01;
 re[10] = rel10 * 0.01;

 /*
 ** Prompt the user for database name - could be command line parameter
 ** printf("SQL Database name:");
 ** fgets : reads the next input line (including the newline) from the stdin
 ** into the character array dbname declared before. at most
 ** DBNAME_MAX characters will be read. The resulting line is terminated by
 ** a NULL string. On end of file or error, fgets returns NULL.
 */
 if (fgets(dbname, DBNAME_MAX, stdin) == NULL)
     exit(1);
 /*
 ** Treat connection errors as fatal.
 */
 EXEC SQL WHENEVER SQLERROR STOP;
 EXEC SQL CONNECT :dbname;

 /*
 ** Call Run_Application() which accepts all typed queries by the users
 ** and sends them to INGRES.
 */
 Run_Application();

 EXEC SQL WHENEVER SQLERROR CONTINUE;
 printf("SQL: Exiting program.\n");

 EXEC SQL ROLLBACK;
 EXEC SQL DISCONNECT;
} /* End main */



/*-----------------------------------------------------------------------------------------------
** Procedure : Run_Application
** Purpose:    Run the SQL monitor Initialize the first SQLDA with the
**             default size (DEF_ELEMS 'sqlvar' elements). Loop while
**             prompting the user for input, and processing the statement.
**             if it is not a SELECT statement then execute it, otherwise
**             open a cursor to process a dynamic SELECT statement.
** Parameters: None
**---------------------------------------------------------------------------------------------*/
void Run_Application()
{

  /*
  ** ESQL declarations
  */
  EXEC SQL BEGIN DECLARE SECTION;
  char stmt_buf[STMT_MAX + 1];                /* SQL statement input buffer       */
  char trans_buf[STMT_MAX + 1];               /* SQL statement input buffer       */
  EXEC SQL END DECLARE SECTION;

  int   stmt_num;                             /* SQL statement number             */
  int   rows;                                 /* Rows affected                    */

  /*
  ** Allocate a new SQLDA
  */
  Init_Sqlda(DEF_ELEMS);

  /*
  ** Accept input until a Cntrl+D is pressed. Cntrl+D signals the
  ** end of file.
  */
  for (stmt_num = 1 ;; stmt_num++)
  {

   drows = 0;

   /
   ** Prompt and read the next statement. If Read_Query
   ** returns NULL then end-of-file = Cntrl+D was detected.
   ** Send the statement number and the statement buffer declared by the
** ESQL at the beginning of this procedure to be filled up by the Read_Query()
** function. STMT_MAX in the maximum characters that could be accepted into the
** declared buffer area stmt_buf. The function will exit if the returned type was
** NULL = Cntrl + D is entered by the user. Calling this way, by using the if
** statement, is better than calling the Read_Query() alone since we will control
** errors in a better way.
*/
if (Read_Query(stmt_num, stmt_buf, STMT_MAX) == NULL)
  break;

/*
** After reading the statement call QueryIdentification() to identify which
** Relational Algebra Operation (RAO) is needed.
*/
FillStructuredQueryTemplate(stmt_num, stmt_buf, STMT_MAX, trans_buf);

/*
** Given the Structured Query Template, we can, now, decide which
** relational algebra operation is required. An RAO is identified and
** transformed into another query depending on the query typed by the user.
** The transformed query is stored in trans_buf and the typed in query
** is stored in stmt_buf.
*/
QueryTransformation(stmt_num, stmt_buf, STMT_MAX, trans_buf);

/*
** Errors are non Fatal from here on out since the returned value by the
** Read_Query() was not NULL. So we are ready to send the buffer stmt_buf to
** INGRES and get the answers to the queries. Before we send the buffer to
** INGRES we have to prepare the stmt FROM :stmt_buf; and describe the stmt
** into the sqlda area (sql dynamic area).
*/
EXEC SQL WHENEVER SQLERROR GOTO Stmt_Err;

/*
** Prepare and describe the statement. If we cannot fully describe
** the statement (our SQLDA is too small) then allocate a new one
** and redescribe the statement.
*/
EXEC SQL PREPARE stmt FROM :trans_buf;
EXEC SQL DESCRIBE stmt INTO :sqlda;
if (sqlda->sqld > sqlda->sqln)
{
  Init_Sqlda(sqlda->sqld);
  /*
  ** redescribe the statement again into :sqlda; so that
  ** the accepted statement can fit into the new :sqlda.
  */
  EXEC SQL DESCRIBE stmt INTO :sqlda;
}

if (sqlda->sqld == 0 )
{
  EXEC SQL EXECUTE stmt;
  rows = sqlca.sqlerrd[2];
}
else /* SELECT */
{
  rows = Exectute_Query();
  /*
  ** After the selection is executed, we will read the "query.ans" file and
  ** perform the correct processing based on the query entered by the user.
  ** At this point we have the answers to the query with source vectors
  ** properly manipulated. We will use the reliability calculation algorithm 1
```

```
      ** to find the reliability of answers to users queries bases on the reliability
      ** of the information sources contributing to the tuple as a whole. In this
      ** prototype we will calculate the reliability of the whole tuple, i.e.,
      ** when the information is complete. Reliability of partial information is
      ** not implemented in this prototype.
      */
      switch (qry.qtype)
      { /* begin switch on query type */
         case 0:
         case 1:
                 ExtendedSelection();
                 break;
         case 2:
                 ExtendedProjection();
                 break;
         case 3:
                 ExtendedUnion();
                 break;
         case 4:
                 ExtendedProduct();
                 break;
         case 5:
                 ExtendedJoin();
                 break;
         case 6:
                 ExtendedIntersection();
                 break;
         case 7:
                 ExtendedDifference();
                 break;
         default:
                printf();
      } /* end switch on query type */
    }
    printf("[%d row(s)]\n", rows - drows);
    /*
    ** skip error handler and return to accept another query from the prompt
    ** after displaying how many rows were affected with the executed statement
    */
    continue;

Stmt_Err:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    /*
    ** Print error message and continue
    */
    Print_Error();
    } /* For each statement */
} /* Run_Application */



/*------------------------------------------------------------------------------
** Procedure : Init_Sqlda
** Purpose:    Initialize SQLDA. Free any old SQLDA's and allocate a new
**             one. Set the number of 'sqlvar' elements.
** Parameters: num_elems   - Number of elements.
**----------------------------------------------------------------------------*/
void Init_Sqlda(num_elems)
int num_elems;
{
  /*
  ** Free the old SQLDA
  */
  if (sqlda)
    free((char *)sqlda);

  /*
  ** Allocate a new SQLDA
  */
  sqlda = (IISQLDA *) Alloc_Mem(IISQDA_HEAD_SIZE + (num_elems * IISQDA_VAR_SIZE), "new SQLDA");
  sqlda->sqln = num_elems;   /* set the size */
} /* Init_Sqlda */



/*-----------------------------------------------------------------------------------------------------
** Procedure : Exectute_Query
**    Purpose: Run a dynamic SELECT statement. The SQLDA has already been
**             described. Accumulate the number of rows processed.
** Parameters: non
**    Returns: Number of rows processed.
**-------------------------------------------------------------------------------------------------*/
int Exectute_Query()
{
  int rows;
  FILE *fileptr;                                 /* pointer to file                          */
  /*
  ** Print the result column names, allocate the result variables.
  ** and setup the types. Save the returned answers in a file for
  ** future manipulation.
  */
  fileptr = fopen("query.ans", "w");

  /*
  ** Display the attribute names and source attributes associated with
  ** the corresponding attributes.
  */
  Print_Scheme(fileptr);

  /*
  ** In case of errors
  */
  EXEC SQL WHENEVER SQLERROR GOTO Close_Csr;

  /*
  ** Open the dynamic cursor
  */
  EXEC SQL OPEN csr FOR READONLY;

  /*
  ** Fetch and print each row
  */
  rows = 0;
  while (sqlca.sqlcode == 0)
  {
     EXEC SQL FETCH csr USING DESCRIPTOR :sqlda;
     if (sqlca.sqlcode == 0)
     {
        rows++;                                  /* count the rows                           */
        Print_Tuples(fileptr);
     }
  } /* while there are more rows */
  fclose(fileptr);

 Close_Csr:
    /*
    ** print the error message
    */
    if (sqlca.sqlcode < 0 )
        Print_Error();

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL CLOSE csr;

    return rows;
} /* Execute Select */



/*------------------------------------------------------------------------------------------
** Procedure : Print_Scheme
** Purpose:    A statement has been defined to set up the SQLDA for
**             result processing. Print all the column names and allocate
**             a result buffer for retrieving data. The result buffer is
**             one buffer (whose size is determined by adding up the results
**             column sizes).
** Parameters: none
**-----------------------------------------------------------------------------------------*/
void Print_Scheme(fileptr)
FILE *fileptr;
{
  int   i;                                  /* Index into 'sqlvar'                      */
  IISQLVAR *sqv;                            /* Pointer to 'sqlvar'                      */
  int base_type;                            /* Base type w/o nullability                */
  int res_cur_size;                         /* Result size required                     */
  int round;                                /* For alignment                           */

  fprintf(fileptr, "%d\n", sqlda->sqld);
  /*
  ** sqlda->sqld: is the number of attributes involved in the
  ** select statement. If the query is select *... then
  ** sqlda->sqld = number of attributes in the extended relation.
  ** IN this case we will save the involved number of attributes
  ** of the given query in a file to treat it later.
  */

  /*
  ** Initialize the array that contains the type of each attribute
  ** in the query. We will assume the integer, long integer, money, and
  ** float point (types of ingres ) have the type code of '0'.
  ** The array t[] will work as attributes type.
  ** This is needed since when we match the disjunct of a selection we
  ** we should know if we are comparing integers or strings.
  */
  for ( i = 0; i < MAXATTR ; i++) { t[i] = 0;}

  for (res_cur_size = 0, i = 0; i < sqlda->sqld; i++)
  {
   /*
   ** Print each column name and its number
   */
   sqv = &sqlda->sqlvar[i];
   printf("[%d] %.*s ", i+1, sqv->sqlname.sqlnamel, sqv->sqlname.sqlnamec);
   fprintf(fileptr, "%.*s~",sqv->sqlname.sqlnamel,sqv->sqlname.sqlnamec);
   /*
   ** Find the base-type of the result (non-nullable)
   */
   if ((base_type = sqv->sqltype) < 0)
       base_type = -base_type;
   /*
   ** Collapse different types into INT, FLOAT or CHAR
```

```
*/
switch (base_type)
{
   case  IISQ_INT_TYPE:
      /*
      ** Always retrieve into a long integer
      */
      res_cur_size += sizeof(long);
      sqv->sqllen = sizeof(long);
      /*
      ** If qry.type = 1, then we have a selection. Store
      ** the type of each attribute name of the scheme of
      ** the extended relation.
      */
      t[i] = 0;
      break;

   case IISQ_MNY_TYPE:
      /*
      ** Always retrieve into a double floating point
      */
      if (sqv->sqltype < 0 )
          sqv->sqltype = -IISQ_FLT_TYPE;
      else
          sqv->sqltype = IISQ_FLT_TYPE;
      res_cur_size += sizeof(double);
      sqv->sqllen = sizeof(double);
      t[i] = 0;
      break;

   case IISQ_FLT_TYPE:
      /*
      ** Always retrieve into a double floating point
      */
      res_cur_size += sizeof(double);
      sqv->sqllen = sizeof(double);
      t[i] = 0;
      break;

   case IISQ_DTE_TYPE:
      sqv->sqllen = DATE_SIZE;
      /* Fall through the handle like CHAR */

   case IISQ_CHA_TYPE:
   case IISQ_VCH_TYPE:
      /*
      ** Assume no binary data is returned from the CHAR type.
      ** Also allocate one extra byte for the null terminator.
      */
      res_cur_size += sqv->sqllen + 1;
      /*
      ** Always round off to aligned data boundary
      */
      if (round = res_cur_size % 4)
          res_cur_size += 4 - round;
      if (sqv->sqltype < 0)
          sqv->sqltype = -IISQ_CHA_TYPE;
      else
          sqv->sqltype = IISQ_CHA_TYPE;
      t[i] = 1;
      break;
} /* switch on base type */
  /*
  ** save away space for the null indicator
```

```
      */
      if (sqv->sqltype <0)
         res_cur_size += sizeof(short);
   } /* for each column */

   fprintf(fileptr, "\n");
   printf("\n");
   /*printf("\n<%d>\n", res_cur_size );*/


/*
** At this point we've printed all column headers and converted
** all types to one of INT, CHAR or FLOAT, Now we allocate
** single result header buffer, and point all the result column data
** areas into it.
**
** If we have an old result data area that is not large enough then
** free it and allocate a new one. Otherwise we can reuse the last
** one that was created when the last statement (SQL query) was
** entered by the user.
*/
if (res_buf.res_length > 0 && res_buf.res_length < res_cur_size)
{
   free(res_buf.res_data);
   res_buf.res_length = 0;
}
/*
** the first time we run this program the res_buf.res_length
** has the value zero. So the coming lines are executed
*/
if (res_buf.res_length == 0)
{ /*
   ** res_cur_size contains the size of the area to be created.
   ** calculated this while we were printing the headers that were
   ** involved in the query given to the system.
   */
   res_buf.res_data = Alloc_Mem(res_cur_size, "result data storage area");
   res_buf.res_length = res_cur_size;
}


/*
** After processing the values of sqltype and sqllen, you should allocate
** storage for the variables that will contain the result of the select
** statement. Do this by pointing the  sqldata at a host language variable
** that will contain the result data.
** Now for each column now assign the result address (sqldata) and
** indicator address (sqlind) from the result data area.
*/
for (res_cur_size = 0, i = 0; i < sqlda->sqld; i++)
{ /*
   ** here if the number of columns involved in the
   ** query is k, then this for loops k times to allocate
   ** space for the k columns attribute values
   */
   sqv = &sqlda->sqlvar[i];

   /*
   ** Find the base-type of the result (non-nullable)
   */
   if ((base_type = sqv->sqltype) < 0)
        base_type = -base_type;

   /*
```

```
        ** Current data points at current offset
        */
        sqv->sqldata = (char *)&res_buf.res_data[res_cur_size];
        res_cur_size += sqv->sqllen;

        if (base_type == IISQ_CHA_TYPE)
        {
          res_cur_size++;                           /* Add one for null                     */
          if (round = res_cur_size % 4)             /* Round off to aligned boundary        */
              res_cur_size += 4 - round;
        }
         /*
         ** Point at result indicator variable
         */
         if (sqv->sqltype < 0)
         {
            sqv->sqlind = (short *)&res_buf.res_data[res_cur_size];
            res_cur_size += sizeof(short);
         }
         else
         {
            sqv->sqlind = (short *)0;
         } /* if type is nullable */
        } /* for each column */
      } /* Print_Scheme */


/*-----------------------------------------------------------------------------------------
** Procedure : Print_Tuples
** Purpose:    For each element inside the SQLDA, print the values. Print
**             its column number too in order to identify it with a column
**             name printed earlier. If the value is NULL print 'N/A".
** Parameters: none
**---------------------------------------------------------------------------------------*/
void Print_Tuples(fileptr)
FILE *fileptr;
{
   int        i;                              /* Index into 'sqlvar'                  */
   IISQLVAR   *sqv;                           /* Pointer to 'sqlvar'                  */
   int base_type;                             /* Base type w/o nullability            */

 /*
 ** For each column, print the column number and the data.
 ** NULL column print as "N/A".
 */
 for (i = 0; i < sqlda->sqld; i++)
 {
    /*
    ** Print each column value with its number. the variable i tells us the
    ** column number being processed.
    */
    sqv = &sqlda->sqlvar[i];
    printf("[%d] ", i+1);

    if (sqv->sqlind && *sqv->sqlind < 0)
    {
       printf(" N/A ");
    }
    else                                   /* Either not nullable, or nullable but not null */
    {
       /*
       ** Find the base-type of the result (non-nullable)
       */
```

```
        if ((base_type = sqv->sqltype) < 0)
             base_type = -base_type;
        switch (base_type)
        {
          case IISQ_INT_TYPE:
            /*
            ** All integers were retrieved into long integers
            */
            printf("%d ", *(long *)sqv->sqldata);
            fprintf(fileptr, "%d", *(long *)sqv->sqldata);
            break;

          case IISQ_FLT_TYPE:
            /*
            ** All floats were retrieved into doubles
            */
            printf("%g ", *(double *)sqv->sqldata);
            fprintf(fileptr, "%g", *(double *)sqv->sqldata);
            break;

          case IISQ_CHA_TYPE:
            /*
            ** All characters were null terminated
            */
            printf("%s ", (char *)sqv->sqldata);
            fprintf(fileptr, "%s", (char *)sqv->sqldata);
            break;
        } /* switch on base type */
    } /* if not null */
    fprintf(fileptr, "~");
 } /* for each column */
 printf(" \n");
 fprintf(fileptr, "\n");
} /* Print_Tuples */


/*-------------------------------------------------------------------------------------------
** Procedure : Print_Error
**    Purpose: SQLCA error detected. Retrieve the error message and
**             print it.
** Parameters: none
**------------------------------------------------------------------------------------------*/
void Print_Error()
{
   /*
   ** Embedded SQL declaration
   */
   EXEC SQL BEGIN DECLARE SECTION;
        char error_buf[150];                    /* For error text retrieval                 */
   EXEC SQL END DECLARE SECTION;

   EXEC SQL INQUIRE_SQL (:error_buf = ERRORTEXT);
   printf("\nSQL Error:\n   %s\n", error_buf);
} /* Print_Error */


/*-------------------------------------------------------------------------------------------
** Procedure : Read_Query
** Purpose:    Reads a statement from standard input. This routine prompts
**             the user for input (using a statement number) and scans input
**             tokens for the statement delimiter (semicolon).
**             - Continues over new-lines.
**            - Uses SQL string literal rules.
** Parameters:
**            stmt_num - Statement number for prompt.
**            stmt_buf - Buffer to fill for input.
**            stmt_max - Max size of statement.
** Returns:
**            A pointer to the input buffer. If NULL then end-of-file was
**            typed in; EOF = Cntrl+D.
**-----------------------------------------------------------------------------------------*/
char *Read_Query(stmt_num, stmt_buf, stmt_max)
int stmt_num;                                 /* current statement number              */
char stmt_buf[];                              /* pointer to buffer for input statement */
int stmt_max;                                 /* maximum size of input statement       */
{
   char input_buf[INPUT_SIZE + 1];            /* Terminal input buffer                 */
   char *icp;                                 /* scans input buffer                    */
   char *ocp;                                 /* to output (stmt_buff)                 */
   int in_string;                             /* For string handling                   */

   ocp = stmt_buf;                            /* Pointer operation                     */
   in_string = 0;
   /*
   ** fgets reads a line and the new line from stdin into characters array input_buf
   ** at most INPUT_SIZE characters are read into input_buf resulting line is terminated
   ** with a '\0' NULL character. Returns input_buf, or NULL when EOF or error
   */
   while (fgets(input_buf, INPUT_SIZE, stdin) != NULL)
   {
      for (icp = input_buf; *icp && (ocp - stmt_buf < stmt_max); icp++)
      { /*
        ** read all characters in buffer and filter them out
        ** Not in string - check for delimiters (";") and new lines
        */
        if (!in_string)
         {
          if (*icp == ';') /* done since we read all the statement */
          {    /*
               ** insert a NULL character at the end of the
               ** read string and return the read statement
               ** to the caller. The returned value by this
               ** procedure is a pointer to a buffer of characters
               ** that contains the filtered statement to be
               ** described and executed.
               */
               *ocp = '\0';                   /* terminate the string with a null character */
               return stmt_buf;               /* return the pointer to stmt_buf        */
          }
          else if (*icp == '\n')              /* a new line is encountered             */
          {
               /*
               ** New line outside of string is replaced with blank
               ** and the pointer is incremented after wards so that
               ** ocp will point to the next location in stmt_buf to
               ** fill it up with the appropriate values.
               */
               *ocp++ = ' ';                  /* replace a \n by a space               */
               break;                         /* exit for loop and read next line      */
          }
          else if (*icp == '\'')              /* Entering string                       */
          {
               in_string++;
          }
          /*
          ** transform one character at a time from the input buffer
        ** that were used by the fgets() to the buffer (stmt_buf),
        ** pointed to by ocp, and was passed as an argument from the
        ** calling procedure. Increment the pointer of the stmt_buf
        ** so that it points to the next location in stmt_buf. Note that
        ** icp is not incremented since this is taken care by the for loop
        */
        *ocp++ = *icp;
      }
      else
      {
        if (*icp == '\n')
        {   /*
            ** New-line entered in string is ignored;
            ** break from the for loop and read the
            ** second line input.
            */
            break;
        }
        else if (*icp == '\'')
        {
           if (*(icp+1) == '\'')                  /* Escaped Code ?                          */
               *ocp++ = *icp++;
           else
               in_string--;
        }
        /*
        ** transform one character at a time from the input buffer
        ** that were used by the fgets() to the buffer (stmt_buf),
        ** pointed to by ocp, and was passed as an argument from the
        ** calling procedure. Increment the pointer of the stmt_buf
        ** so that it points to the next location in stmt_buf. Note that
        ** icp is not incremented since this is taken care by the for loop
        ** *ocp is incremented after the assignment statement is done.
        */
        *ocp++ = *icp;
      } /* if instring */
    } /* for all character in buffer */

    if (ocp - stmt_buf >= stmt_max)
    {
      /*
      ** Statement is too large; ignore it and try again
      ** until the users enters a line whose length is less
      ** than or equal to stmt_max
      */
      printf(" SQL Error: Maximum statement length (%d) exceeded. \n",
            stmt_max);
      ocp = stmt_buf;
      in_string = 0;
    }
    else
    { /*
      ** Break on new line - print continue sign
      ** and read another line input after going
      ** back to the while loop
      */
      printf(">");
    }

 } /* while reading from standard input */
 /*
 ** since users pressed Cntrl+D = EOF marker.
 */
 return NULL;
```

```
} /* Read_Query() ends here */


/*------------------------------------------------------------------------------------------
** Procedure : Alloc_Mem
** Purpose:    General purpose memory allocator. If it can not allocate
**             enough space. It prints a fatal error and aborts any
**             pending updates.
**
** Parameters:
**            mem_size      - size of space requested.
**            error_string  - Error message to print if failure.
** Returns:
**            Pointer to newly allocated space.
**--------------------------------------------------------------------------------------*/
char *Alloc_Mem(mem_size, error_string)
int mem_size;
char *error_string;
{
    char  *mem;

    mem = calloc(1, mem_size);
    if (mem)
        return mem;
    /*
    ** Print an error and roll back any updates
    */
    printf("SQL Fatal Error: Cannot allocate %s (%d bytes).\n",error_string, mem_size);
    printf("Any pending updates are being rolled back.\n");
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    EXEC SQL DISCONNECT;
    exit(-1);
} /* Alloc_Mem */



/*
** Procedure: Conjunction()
**   Purpose: Performs the conjunction between two source vectors
**            and returns the answer as an argument to the calling
**            procedure.
**    return: the result of conjunction.
*/
void Conjunction(u1, v1, w1)
char u1[MAXSOURCES + 1];                    /* neutral source vector              */
char v1[MAXSOURCES + 1];                    /* the read source vector             */
char w1[MAXSOURCES + 1];                    /* the result source vector           */
{

int c;
int i, j;

/*
** Calculate w1 = u1 /\ v1.
*/
for (i = 0; i < MAXSOURCES + 1; i++)
{
 if (u1[i] == '\0')
 {
  if ( v1[i] == '\0') {
    w1[i] = '\0'; break; }
  else
  {
    for (j = i; j < MAXSOURCES + 1; j++)
      wi[j] = vi[j]; break;
  }
 }
 else
 {
  if (vi[i] == '\0')
  {
    for (j = i; j < MAXSOURCES + 1; j++)
      wi[j] = ui[j]; break;
  }
  else
  {
    if ( (ui[i] == '1') && (vi[i] == '1') ) {
      wi[i] = '1'; continue;}
    if ( (ui[i] == '1') && (vi[i] == '0') ) {
      wi[i] = '1'; continue;}
    if ( (ui[i] == '1') && (vi[i] == '-') ) {
      wi[i] = 'T'; wi[1] = '\0'; break;}
    if ( (ui[i] == '1') && (vi[i] == 'T') ) {
      wi[0] = 'T'; wi[1] = '\0'; break;}

    if ( (ui[i] == '0') && (vi[i] == '1') ) {
      wi[i] = '1'; continue;}
    if ( (ui[i] == '0') && (vi[i] == '0') ) {
      wi[i] = '0'; continue;}
    if ( (ui[i] == '0') && (vi[i] == '-') ) {
      wi[i] = '-'; continue;}
    if ( (ui[i] == '0') && (vi[i] == 'T') ) {
      wi[i] = 'T'; wi[1] = '\0'; break;}

    if ( (ui[i] == '-') && (vi[i] == '1') ) {
      wi[i] = 'T'; wi[1] = '\0'; break;}
    if ( (ui[i] == '-') && (vi[i] == '0') ) {
      wi[i] = '-'; continue;}
    if ( (ui[i] == '-') && (vi[i] == '-') ) {
      wi[i] = '-'; continue;}
    if ( (ui[i] == '-') && (vi[i] == 'T') ) {
      wi[0] = 'T'; wi[1] = '\0'; break;}

    if (ui[i] == 'T') {
      wi[i] = 'T'; wi[1] = '\0'; break;}
  }
 }
} /* for */

}


/*------------------------------------------------------------------------------------------
** Procedure : FillStructuredQueryTemplate()
** Purpose:    Fill up the Structured Query Template by the number of
**             attributes, attribute names, number of tables, table names,
**             selection condition (conjuncts).
**             Relational algebra operations are: Selection, projection,
**             union, cartesian product, join, intersection, set difference.
** Parameters:
**            stmt_num - Statement number for prompt.
**            stmt_buf - Buffer that contains the query to be identified.
**            stmt_max - Max size of statement.
** Returns:   return the Structured Query Template information in qry
**
```

```
**-----------------------------------------------------------------------------------------*/
char *FillStructuredQueryTemplate(stmt_num, stmt_buf, stmt_max, trans_buf)
int stmt_num;                                   /* current statement number                 */
char stmt_buf[];                                /* pointer to buffer of input statement     */
char trans_buf[];                               /* pointer to buffer of transformed statement */
int stmt_max;                                   /* maximum size of input statement          */
{
 char *icp;                                     /* scans the stmt_buf full buffer           */
 char *ocp;                                     /* to output (trans_buf) Empty buffer       */
 int in_string, i, j;                           /* For string handling                      */
 int flg;

 /*
 ** Initialization of all required variables, pointers
 ** and structures. ClearStructuredQueryTemplate ...
 */

 ocp = trans_buf;                               /* Pointer to empty buffer trans_buf         */
 in_string = 0;
 flg = 0;

 qry.nattr = 0;
 qry.ntbls = 0;
 qry.ncnd = 0;
 qry.qtype = 0;

 for ( i = 0; i < MAXATTR ; i++)
 {
   for (j = 0; j < ATTWIDTH; j++)
   {
     qry.attr[i][j] = '\0';
   }
 }

 for ( i = 0; i < MAXTABLES ; i++)
 {
   for (j = 0; j < TABWIDTH; j++)
   {
     qry.tbls[i][j] = '\0';
   }
 }

 for ( i = 0; i < MAXCONJ; i++)
 {
   for (j = 0; j < CONJWIDTH; j++)
   {
     qry.cnd[i][j] = '\0';
   }
 }

 /*
 ** Clear the buffer before use.
 */
 ocp = trans_buf;
 for (ocp = trans_buf; *ocp && (ocp - trans_buf < stmt_max); ocp++)
 {
   *ocp = '\0';
 }
 ocp = trans_buf;
 for (icp = stmt_buf; *icp && (ocp - trans_buf < stmt_max); icp++)
 {
   *ocp++ = *icp;
 } /* for all characters in stmt_buf */
 /*
```

```
** Now, we have a copy of the query entered by the user we will scan it
** and identify the RAO. Once the RAO is identified we will transform the
** Query into a form that will include the source vectors associated with the
** tuples and the required attributes based on the RAO.
*/

for (icp = trans_buf ; *icp; icp++)
{
  j = 0;
  /*
  ** Read the 'select'
  */
  if ( (*icp++ == slct[1]) && (*icp++ == slct[2]) && (*icp++ == slct[3]) &&
       (*icp++ == slct[4]) && (*icp++ == slct[5]) && (*icp++ == slct[6]) )
  {
  while (*icp == ' '){
    *icp++;}
  /*
  ** at this point we have the attribute names
  */
  while ( 1 ){ /* read all attributes */
    while ( !(*icp == ',') && !(*icp == ' ') ){
      /*
      ** get attribute after attribute
      */
      qry.attr[qry.nattr][j++] = *icp;
      *icp++;
    } /* while get attribute after attribute*/
    /*
    ** put NULL string at the end of attributes
    */
    qry.attr[qry.nattr][j++] = '\0';
    qry.nattr++;                               /* add number of attributes             */
    j = 0;
    if (*icp == ' ') {
      *icp++;
      break; }                                 /* means finish reading attributes      */
    /* when you exit because of ',' you still expect attributes */
    *icp++;
    /*
    ** skip spaces
    */
    while (*icp == ' '){*icp++;}
  }/* read all attributes */

  /*
  ** read the 'from'
  */
  if ( (*icp++ == fm[1]) && (*icp++ == fm[2]) &&
       (*icp++ == fm[3]) && (*icp++ == fm[4]) )
  {
  j = 0;
  while (*icp == ' '){ *icp++;}
  while ( 1 ){ /* read all table names */
    while ( !(*icp == ',') && !(*icp == ' ') && !(*icp == '\0') ){
      /*
      ** save the tables names in the array of tables
      */
      qry.tbls[qry.ntbls][j++] = *icp;
      *icp++;
    } /* while get table after table */
    /*
    ** put NULL string at the end of attributes
    */
    qry.tbls[qry.ntbls][j++] = '\0';
    qry.ntbls++;                                    /* add number of attributes              */
    j = 0;
    /*
    ** end of query
    */
    if (*icp == '\0') {break;}
    /*
    ** expect to read the 'where' selection condition
    */
    if (*icp == ' ') {
      *icp++;
      break; }
    /*
    ** when you exit because of ',' you still
    ** expect more table names
    */
    *icp++;
    while (*icp == ' '){ *icp++;}
  } /* read all tables */
  } /* read from */

  /*
  ** at this point we have the attributes and the tables in the query
  */

  j = 0;
  /*
  ** read the 'where'
  */
  if ( (*icp++ == whr[1]) && (*icp++ == whr[2]) &&
       (*icp++ == whr[3]) && (*icp++ == whr[4]) &&
       (*icp++ == whr[5]) )
  {
  while (*icp == ' '){ *icp++;}                    /* discard spaces                        */
  while (*icp == '('){ *icp++;}                    /* every conjunct should start with a '('  */
  while (*icp == ' '){ *icp++;}                    /* discard spaces                        */

  /*
  ** read all conjuncts (F1 /\ ... /\ F1)
  */
  while ( 1 ){
    while ( !(*icp == ')') ){
      qry.cnd[qry.ncnd][j++] = *icp;
      *icp++;
    } /* while get one conjunct after the other */
    *icp++;
    /*
    ** discard spaces
    */
    while (*icp == ' '){ *icp++;}
    if (*icp++ == 'o')
    {
      if (*icp++ == 'r')
      {
       qry.cnd[qry.ncnd][j++] = '\0';              /* put NULL string at the end of attributes */
       qry.ncnd++;                                 /* increment the number of conjuncts       */
       j = 0;
       while (*icp == ' '){ *icp++;}
       while (*icp == '('){ *icp++;}               /* every conjunct should start with a '('   */
       continue;
      }
      else
      {
          /*
          ** Syntax error 'r' is missed abort operation
          */
         }
       }
       else
       { /*
         ** test for semicolon to terminate
         */
        if (*icp == '\0')
        { /* terminate */
          qry.ncnd++;                              /* increment the number of conjuncts          */
          break;
        }
       }
     } /* read all conjuncts */
     } /* read where */

     } /* read select */
   } /* for end of buffer */
 }



/*------------------------------------------------------------------------------------------
** Procedure : QueryTransformation()
** Purpose:    Use the Structured Query Template to identify which relational
**             algebra operation is required.
**             Relational algebra operations are: Selection, projection,
**             union, cartesian product, join, intersection, set difference.
** Parameters:
**            stmt_num - Statement number for prompt.
**            stmt_buf - Buffer that contains the query to be identified.
**            stmt_max - Max size of statement.
**            qry - is the Structured Query Template.
** Returns:   pointer to a character string
**            trans_buf - pointer to transformed query.
**-----------------------------------------------------------------------------------------*/
char *QueryTransformation(stmt_num, stmt_buf, stmt_max, trans_buf)
int stmt_num;                                    /* current statement number                 */
char stmt_buf[];                                 /* pointer to buffer for input statement    */
char trans_buf[];                                /* pointer to buffer for transformed statement */
int stmt_max;                                    /* maximum size of input statement          */
{

  int i, j;
  char *ocp;                                     /* to output (trans_buf) Empty buffer       */

  FILE *ifileptr;

  int relFlag;
  int rel1,rel2,rel3,rel4,rel5,rel6,rel7,rel8,rel9,rel10;

/*
** Print the query typed by the user before transforming it to another
** form printing the answers.
*/
printf("Query: '%s'\n", stmt_buf);

/*
** Open the file that contains the reliabilities of information sources
** read the reliability of information sources and print to the users
** the reliability of information sources. This is done for each query
** typed in by the user.
```

```
*/
 ifileptr = fopen("options.rel", "r");
 fscanf(ifileptr, "%d", &relFlag);
 fscanf(ifileptr,"%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",&rel1,&rel2,&rel3,&rel4,&rel5,
                                                  &rel6,&rel7,&rel8,&rel9,&rel10);
 fclose(ifileptr);
 if ( relFlag != 0 )
 { /*
   ** Reliability calculation is allowed
   */
    re[1] = rel1 * 0.01; re[2] = rel2 * 0.01; re[3] = rel3 * 0.01;
    re[4] = rel4 * 0.01; re[5] = rel5 * 0.01; re[6] = rel6 * 0.01;
    re[7] = rel7 * 0.01; re[8] = rel8 * 0.01; re[9] = rel9 * 0.01;
    re[10] = rel10 * 0.01;
    printf("Information Sources Reliabilities:\n");
    printf("P(S1 ) = %6.4f, P(S2 ) = %6.4f, P(S3 ) = %6.4f, P(S4 ) = %6.4f, P(S5 ) = %6.4f,",
                                              re[1], re[2], re[3], re[4], re[5]);
    printf("\n");
    printf("P(S6 ) = %6.4f, P(S7 ) = %6.4f, P(S8 ) = %6.4f, P(S9 ) = %6.4f, P(S10) = %6.4f.",
                                             re[6], re[7], re[8], re[9], re[10]);
    printf("\n");
 }

/*
** Testing for selection: '*' as an attribute value,
** one and only one extended relation, and zero or
** more conjuncts in the selection condition.
*/
if ( (qry.nattr == 1) &&
     (qry.attr[0][0] == '*') &&
     (qry.ntbls == 1) &&
     (qry.ncnd >= 0) )
{
  printf("The Query is a Selection...\n");
  qry.qtype = 1; /* selection type */
}

/*
** Testing for Projection: One or more attribute(s), one table name
** and no selection condition (no arguments for the 'where').
*/
if ( (qry.nattr >= 1) &&
     (qry.attr[0][0] != '*') &&
     (qry.ntbls == 1) &&
     (qry.ncnd == 0) )
{
  qry.qtype = 2;
  printf("The Query is a Projection...\n");
  /*
  ** Do the proper query transformation to include the
  ** corresponding source vectors associated with the
  ** named attributes. Replace the attribute names
  ** with a '*' to retrieve all attributes.
  */
  ocp = trans_buf;                         /* point to beginning of trans_buf               */
  for (ocp = trans_buf; *ocp && (ocp - trans_buf < stmt_max); ocp++)
  {
    j = 0;
    if ( (*ocp++ == slct[1]) && (*ocp++ == slct[2]) && (*ocp++ == slct[3]) &&
         (*ocp++ == slct[4]) && (*ocp++ == slct[5]) && (*ocp++ == slct[6]) )
    {
      while (*ocp == ' '){*ocp++;}
      /*
      ** insert a '*' instead of the attribute names
```

```
        */
        *ocp++ = '*';
        /*
        ** at this point we have the attribute names
        */
        while ( 1 )
        { /*
          ** replace attributes by a '*'
          */
          while ( !(*ocp == ',') && !(*ocp == ' ') )
          { /*
            ** get attribute after attribute
            */
            *ocp++ = ' ';
          } /* while get attribute after attribute*/
          if (*ocp == ' ')
          {
            *ocp++;
            break;
          } /* means finish inserting ' ' instead of attributes */
          /*
          ** when you exit because of replace the comma by a space and
          ** keep on replacing attributes by spaces
          */
          *ocp++ = ' ';
          while (*ocp == ' '){*ocp++;}
        } /* replace attributes by a '*' */
        /*
        ** when you exit from here you keep reading until you expect another
        ** select statement. This is helpful when we have a UNION. In this
        ** prototype we have the UNION is treated under selection.
        */
      }
    }
  }

  /*
  ** Testing for a Union: In this model the Union is exactly
  ** the same as the union in the regular case. So to perform
  ** a Union the query is sent directly to INGRES and
  ** use the reliability calculation algorithms to find
  ** the reliability of answers to user's queries.
  */


  /*
  ** Testing for a Cartesian Product: Cartesian Product is
  ** identified by having non zero numbers of attributes,
  ** more than two extended relations (tables) and no
  ** selection condition (no arguments for the 'where'.
  */
  if ( (qry.nattr == 1) &&
       (qry.attr[0][0] == '*') &&
       (qry.ntbls >= 2) &&
       (qry.ncnd == 0) )
  {
    printf("The Query is a Cartesian Product...\n");
    qry.qtype = 4;
  }

  /*
  ** Testing for a Join: Join is identified by having
  ** non zero number of attributes,more than two extended
  ** relations (tables) and arguments for the 'where'
```

```
** clause. Note that the 'where' have EXACTLY one
** conjunct.
*/
if ( (qry.nattr >= 1) &&
     (qry.attr[0][0] != '*')&&
     (qry.ntbls >= 2) &&
     (qry.ncnd != 0) )
{
  printf("The Query is a Join...\n");
  qry.qtype = 5;

  /*
  ** Replacing attributes by '*' to get all the source attributes
  ** associated with the corresponding regular attribute values.
  */
  ocp = trans_buf;                              /* point to beginning of trans_buf              */
  for (ocp = trans_buf; *ocp && (ocp - trans_buf < stmt_max); ocp++)
  {
    j = 0;
    if ( (*ocp++ == slct[1]) && (*ocp++ == slct[2]) && (*ocp++ == slct[3]) &&
         (*ocp++ == slct[4]) && (*ocp++ == slct[5]) && (*ocp++ == slct[6]) )
    {
      while (*ocp == ' '){*ocp++;}              /* skip all spaces                              */
      /*
      ** insert a '*' instead of the attribute names
      */
      *ocp++ = '*';
      /*
      ** at this point we have the attribute names
      */
      while ( 1 )
      { /*
        ** replace attributes by a '*'
        */
        while ( !(*ocp == ',') && !(*ocp == ' ') )
        { /* get attribute after attribute */
          *ocp++ = ' ';
        } /* while get attribute after attribute*/
        if (*ocp == ' ')
        {
          *ocp++;
          break;
        } /* means finish inserting ' ' instead of attributes */
        /*
        ** when you exit because of a comma, replace the comma by a
        ** space and keep on replacing attributes by spaces
        */
        *ocp++ = ' ';
        while (*ocp == ' '){ /* skip spaces */
        *ocp++;}
      }/* replace attributes by a '*' */
    }
  }
}

/*
** Testing for Intersection: Intersection is a special Join
** where the join attributes form a set that is equal to the
** schemes of both extended relation given in the query.
*/

/*
** Testing for Set Difference:
**
```

```
*/
/*
if (
    (qry.ntbls == 2) &&
    (qry.ncnd == 0) )
{
  printf("The Query is a Set Difference...\n");
  qry.qtype = 7;
}
*/

}



/*----------------------------------------------------------------------------------------
** Procedure: ExtendedSelection();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**--------------------------------------------------------------------------------------*/
void  ExtendedSelection()
{

 typedef struct conjunct {
  char op1[ATTWIDTH];                          /* attribute                          */
  char op[3];                                  /* operation                          */
  char op2[ATTWIDTH];                          /* attribute                          */
 } con;

 FILE *ifileptr, *ofileptr, *relfileptr;
 int  inchar, j, i, i1, k, l, relFlag;
 int  numattr, regattr, srcattr;
 char u[MAXSOURCES + 1];                       /* neutral source vector              */
 char u1[MAXSOURCES + 1];
 char v[MAXATTR][MAXSOURCES + 1];              /* the read source vector             */
 char c[WIDTH];                                /* current read value                 */
 char h[MAXATTR][ATTWIDTH];                    /* names of regular attributes        */
 char r[MAXATTR][WIDTH];                       /* attribute values                   */
 con  F[MAXDIS][MAXCON];                       /* holds the selection condition      */
 int  pos[MAXATTR];                            /* needed to keep temporarily flags   */
 int  p[MAXATTR];                              /* needed to keep flags for all conjuncts */
 int  simple;                                  /* flag                               */
 char opr1[WIDTH];                             /* First Operand for a conjunct       */
 char opr2[WIDTH];                             /* Second Operand for a conjunct      */
 int satisfaction;
 int opr1type, opr2type;
 float rel;

 /*
 ** Initialize the arrays of source vectors, attribute names, and
 ** attribute values.
 */
 for (i = 0; i <= MAXSOURCES + 1; i++) {u[i] ='\0'; v[i][0] = '\0';u1[i] ='\0';}
 for (i = 0; i <= WIDTH ; i++) {c[i] ='\0'; opr1[i] = '\0'; opr2[i] = '\0';}
 for (i = 0; i <= MAXATTR ; i++) {h[i][0] ='\0'; r[i][0] = '\0'; pos[i] = 0; p[i] = 0;}

 /*
 ** Initialize the two dimensional array that represents the selection
 ** condition. Every row in this array is a conjunct.
 */
 for (i = 0; i <= MAXDIS ; i++)
 {
        for (j = 0;  j <= MAXCON; j++)
        {
          F[i][j].op[0] = '\0';
          F[i][j].op[1] = '\0';
          F[i][j].op[2] = '\0';
          F[i][j].op[3] = '\0';
          for (k = 0; k <= ATTWIDTH; k++)
          {
            F[i][j].op1[k] = '\0';
            F[i][j].op2[k] = '\0';
          }
        }
      }
      i = 0; j = 0;  k = 0; i1 = 0, l = 0;

      relFlag = 0;
      satisfaction = 0;
      opr1type = 0;                            /* by default the operand type is an integer  */
      opr2type = 0;                            /* integer by default                          */

      ifileptr = fopen("query.ans", "r");
      ofileptr = fopen("query.rel", "w");

      fscanf(ifileptr, "%d", &numattr);        /* read number of attributes                   */
      fprintf(ofileptr, "%d\n", numattr);
      regattr = (numattr - 1)/ 2;              /* Calculate the number of regular attributes */
      inchar = getc(ifileptr);                 /* to skip the <cr> characters                  */
      inchar = getc(ifileptr);
      /*
      ** print the regular attributes names only
      */
      j = 1;
      i =0;
      printf("[%d] ",j);
      while ( j <= regattr )
      {
        if (inchar != '~')
          {
           h[j-1][i++] = inchar;
          }
          else
          { /*
           ** increment j towards the number of regular attributes
           */
           j++;
           h[j-2][i] = '\0';
           fprintf(ofileptr, "%s~", h[j-2]);
           printf("%s ", h[j-2]);
           i = 0;
           if (j <= regattr) {printf(" [%d] ",j);}
          }
        inchar = getc(ifileptr);
      }

      /*
      ** Skip the name of source attributes by reading
      ** until a '\n' is reached.
      */
      while ( (inchar = getc(ifileptr) != '\n') ) {;}
      fprintf(ofileptr, "\n");

      i = 0;

      /*
```

```
** Read the Structured Query Template and store the
** disjuncts into an array. The rows of this array
** represent the disjunction of the selection condition.
** the columns of this array represent conjuncts.
** where each conjunct is a simple formula (explained
** in the thesis).
*/
i = 0;                         /* runs over the attribute names, operations, and attribute values */
j = 0;                         /* runs over the number of disjuncts                               */
k = 0;                         /* runs over the number of conjuncts in one disjunct               */
while (qry.cnd[j][i] != '\0')
{ /* while not end of conjunct */
 while (1)
 {
  /*
  ** read the first attribute name, operations, and second
  ** attribute name (the second operand might be a constant).
  */
  l = 0; /* runs over the first operand */
  while (qry.cnd[j][i] != ' ')
  { /*
    ** while there is no space read and save attribute name
    */
    F[j][k].op1[l] = qry.cnd[j][i];
    i++;
    l++;
  }
  F[j][k].op1[l] =  '\0';
  /*
  ** skip spaces
  */
   while (qry.cnd[j][i] == ' ') {i++;}

 /*
 ** Read the operations <, >, =, <=, >=, !=.
 */
 l = 0; /* runs over the operation */
 while (qry.cnd[j][i] != ' ')
 {
   F[j][k].op[l] = qry.cnd[j][i];
   i++;
   l++;
 }
 F[j][k].op[l] = '\0';


 /*
 ** Skip spaces.
 */
 while (qry.cnd[j][i] == ' ') {i++;}

 /*
 ** Read second operand that could be an attribute or a
 ** a constant.
 */
 l = 0;
 while ( (qry.cnd[j][i] != ' ') && (qry.cnd[j][i] != '\0') )
 { /* while there is no space read save attribute name */
   F[j][k].op2[l] = qry.cnd[j][i];
   i++;
   l++;
 }
 F[j][k].op2[l] = '\0';
```

```
   if (qry.cnd[j][i] == '\0') {break;}

   /*
   ** Skip spaces
   */
   while (qry.cnd[j][i] == ' ') {i++;}

   /*
   ** At this point we expect to continue reading the conjunct or we expect
   ** to read '\0' which means that the conjunct is terminated. In case
   ** there are more attributes and operators to read in the same
   ** conjunct we will loop again until a ')' is read.
   */
   if (qry.cnd[j][i] == '\0')
   { /* read another conjunct expect to read an 'or' or a ';'*/
     j++;
     break;
   }
   else
   {
     /*
     ** Skip the 'and' logical operator and prepare another
     ** storage for a new conjunct.
     */
     i = i+3;
     /*
     ** Skip spaces
     */
     while (qry.cnd[j][i] == ' ') {i++;}
     k++;
     continue;
   }
  }
 j++; /* get another conjunct */
 i = 0; /* set the beginning of the conjunct */
 k = 0;
 }

 /*
 ** print the regular attribute values and
 ** perform the reliability calculation for
 ** the whole tuple.
 */
 inchar = getc(ifileptr);
 while ( inchar != EOF )
 { /* not EOF */
  while (inchar != '\n')
  { /* '\n' */
   /*
   ** clear the array that contains the attribute values
   */
   j = 1;
   i = 0;
   while ( j <= numattr )
   { /*
     ** save regular attribute values and their
     ** corresponding source vectors
     */
     if (inchar != '^')
     {
       r[j-1][i++] = inchar;
     }
     else
```

```
      { /*
        ** increment j towards the number of regular attributes
        */
        j++;
        r[j-2][i] = '\0';
        i = 0;
      }
      inchar = getc(ifileptr);
    } /* print regular attribute values */

  /*
  ** Given h[][], the array of regular attribute names,
  ** r[][], the array of attribute values with the corresponding
  ** source vectors, and the F[][], the array that represents
  ** the selection condition (disjunctive normal form), we want
  ** to calculate the source vector associated with each tuple.
  */
   j = 0;
   k = 0;
   while  ( ( !(F[j][k].op1[0] == '\0') &&
              !(F[j][k].op[0] == '\0') &&
              !(F[j][k].op2[0] == '\0') )  ||  (qry.ncnd == 0)  )
   { /*
     ** Read one conjunct at a time and test if the tuple stored
     ** in r[][] satisfy any conjunct. For every conjunct satisfied
     ** by a tuple in r[][] perform the appropriate conjunction operation
     ** between the source vector associated with the tuple in r[][]
     ** and the source vectors associated with the attributes found in the
     ** conjunct satisfied by the tuple r[][]
     */

     i = 0;
     k = 0;
     while  ( ( !(F[j][k].op1[0] == '\0') &&
                !(F[j][k].op[0] == '\0') &&
                !(F[j][k].op2[0] == '\0') )  ||  (qry.ncnd == 0)  )
     {
       /*
       ** read every simple condition in a given conjunct and test if
       ** it is satisfied by the tuple in r[][]. If the simple condition
       ** is not satisfied by the tuple, then there is no need to continue
       ** testing other simple conditions.
       */
       i = 0;
       while ( !(h[i][0] == '\0') )
       {
         if ( !(EqualStrings(h[i], F[j][k].op1)) )
         { /*
           ** the first operand in the selection condition is equal to
           ** one of the attribute names in the array h[][]. Save the
           ** position of that attribute in the array pos[].
           **
           */
           pos[i] = 1;
         }
         i++;
       }
       /*
       ** Consider the second operand in the current conjunct. Search
       ** in h[][] and try to find a match. If a match was not found
       ** in h[][], then the second operand is a constant. If a match
       ** was found, then the second operand is an attribute and in this
       ** case set the appropriate flag pointer in pos[].
       */
```

```
i = 0;
while ( !(h[i][0] == '\0') )
{
  if ( !(EqualStrings(h[i], F[j][k].op2)) )
  {
    /* printf("%s = %s ? %s \n", h[i], r[i], F[j][k].op2); */
    pos[i] = 1;
    simple = 1;                    /* case when there are two attributes in a conjunct */
  }
  i++;
}
/*
** Based on the 'simple' flag we will test if the conjunct is
** satisfied by the tuple in r[][]. In both cases read the attribute
** value of the first operand.
** Get the attribute value to opr1[] after initializing it.
*/
i = 0;
while ( !(h[i][0] == '\0') )
{
   if ( pos[i] == 1 )
   {
      /*
      ** save the attribute type
      */
      opr1type = t[i];
      l = 0;
      while ( !(r[i][l] == '\0' ) )
      {
         opr1[l] = r[i][l];
         l++;
      }
      opr1[l] = '\0';
      i++;
      break;
   }
   i++;
}

if (simple == 1)
{
  /* printf("simple = %d ",simple);*/
  /*
  ** get the attribute value for the second operand
  */
  while ( !(h[i][0] == '\0') )
  {
    if (pos[i] == 1)
    {
      l = 0;
      while ( !(r[i][l] == '\0') )
      {
        opr2[l] = r[i][l];
        l++;
      }
      opr2[l] = '\0';
    }
  i++;
  }
}
else
{ /*
  ** the second operand is a constant and should be read from
  ** F[j][k].op[] and stored in opr2[]
  */
  i = 0;
  while ( !(F[j][k].op2[i] == '\0') )
  {
    opr2[i] = F[j][k].op2[i];
    i++;
  }
  opr2[i] = '\0';
}

/*
** read the operation and perform the appropriate test
*/

/*
** Testing '<'
*/
if ( (F[j][k].op[0] == '<') && (F[j][k].op[1] != '=') )
{ /* Test if operation is '<' */
  if ( (opr1type == 1) )
  {  /* begin if opr1type */
    for ( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
    if ( (opr1[i] == '\0') && (opr2[i] == '\0') )
    { /* no chance to have one less than the other */
      satisfaction = 0;
    }
    else
    { /* test if less by subtracting one from the other */
      if (opr1[i] - opr2[i] < 0)
      { /* then opr1[] is less than opr2[] */
          satisfaction = 1;
      }
      else
      {
          satisfaction = 0;
      }
    }
  } /* end if opr1type */
  else
  {
    if ( (str2flt(opr1) < str2flt(opr2)) )
       /* operand 1 and operand 2 are equal */
       satisfaction = 1;
    else
       satisfaction = 0;
  }
} /* end test '<' */

/*
** Testing '>'
*/
if ( (F[j][k].op[0] == '>') && (F[j][k].op[1] != '=') )
{ /* Test if operation is '>' */
  if ( (opr1type == 1) )
  {  /* begin if opr1type */
    /*
    ** compare the two operands opr1[] and opr2[] to see if
    ** tuple r[][] satisfy the operation.
    */
    for( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
    if  ( (opr1[i] == '\0') && (opr2[i] == '\0')  )
    { /* no chance to have one greater than the other */
      satisfaction = 0;
    }
```

```
      else
      { /* test if greater by subtracting one from the other */
        if (opr1[i] - opr2[i] > 0)
        { /* then opr1[i] > opr2[i] and satisfaction is 1 (true ) */
          satisfaction = 1;
        }
        else
        {
          satisfaction = 0;
        }
      }
    }
    else
    {
      if ( (str2flt(opr1) > str2flt(opr2)) )
         /* operand 1 and operand 2 are equal */
         satisfaction = 1;
      else
         satisfaction = 0;
    } /* end if opr1type */
  }

  /*
  ** Testing '='
  */
if (F[j][k].op[0] == '=')
{
  if ( (opr1type == 1) )
  { /* begin if opr1type */
    for( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
    if  ( (opr1[i] == '\0') && (opr2[i] == '\0') )
      satisfaction = 1;
    else
      satisfaction = 0;
  }
  else
  {
    if ( !(str2flt(opr1) - str2flt(opr2)) )
       /* operand 1 and operand 2 are equal */
       satisfaction = 1;
    else
       satisfaction = 0;
  } /* end if opr1type */
}

/*
** Testing '<='
*/
if ( (F[j][k].op[0] == '<') && (F[j][k].op[1] == '=') )
{ /* Test if operation is '<' */
  if ( (opr1type == 1) )
  {  /* begin if opr1type */
    for ( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
    if ( (opr1[i] == '\0')  )
    { /* they are equal */
      satisfaction = 1;
    }
    else
    { /* test if opr1[i] is less opr2[i] by subtracting one from the other */
      if ( (opr1[i] - opr2[i] <= 0 )  )
      { /* then opr1[] is less than opr2[] */
         satisfaction = 1;
      }
      else
```

```
        {
           satisfaction = 0;
        }
      }
    }
    else
    {
      if ( (str2flt(opr1) <= str2flt(opr2)) )
         /* operand 1 and operand 2 are equal */
         satisfaction = 1;
      else
         satisfaction = 0;
    } /* end if opr1type is true */
  } /* end test '<' */

  /*
  ** Testing '>='
  */
  if ( (F[j][k].op[0] == '>') && (F[j][k].op[1] == '=') )
  { /* Test if operation is '>' */
    if ( (opr1type == 1) )
    {  /* begin if opr1type */
     for ( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
     if ( (opr2[i] == '\0')   )
     { /* they are equal */
       satisfactior = 1;
     }
     else
     { /* test if opr1[i] is less opr2[i] by subtracting one from the other */
       if ( (opr1[i] - opr2[i] >= 0 )   )
       { /* then opr1[] is less than opr2[] */
          satisfaction = 1;
       }
       else
       {
          satisfaction = 0;
       }
     }
    }
    else
    {
      if ( (str2flt(opr1) >= str2flt(opr2)) )
         /* operand 1 and operand 2 are equal */
         satisfaction = 1;
      else
         satisfaction = 0;
    } /* end test opr1type */
  } /* end test '<' */

 /*
 ** Testing '!='
 */
 if ( (F[j][k].op[0] == '!') && (F[j][k].op[1] == '=') )
 { /* Test if operation is '>' */
   if ( (opr1type == 1) )
   {  /* begin if opr1type */
      for ( i = 0; (opr1[i] == opr2[i]) && (opr1[i] != '\0') && (opr2[i] != '\0'); i++);
      if ( (opr1[i] == '\0') && (opr2[i] == '\0')  )
      { /* they are equal */
        satisfaction = 0;
      }
      else
      { /* test if opr1[i] is less opr2[i] by subtracting one from the other */
        if ( (opr1[i] - opr2[i] != 0 )   )
```

```
            { /* then opr1[] is less than opr2[] */
               satisfaction = 1;
            }
            else
            {
               satisfaction = 0;
            }
          }
        }
        else
        {
          if ( (str2flt(opr1) != str2flt(opr2)) )
             /* operand 1 and operand 2 are equal */
             satisfaction = 1;
          else
             satisfaction = 0;
        }/* end test opritype */
      } /* end test '<' */
      if (qry.ncnd == 0) {break;}
      k++;
      for ( i = 0 ; i < MAXATTR ; i++)
      {
        if (p[i] == 0)
        { /* this prevents overwriting the '1' values of p[] */
          p[i] = pos[i];                       /* array that keeps track of all attributes
                                                  read in one conjunct                           */
        }
        pos[i] = 0; /* temporary array to keep flags */
      }
    } /* finish reading conjuncts */
  /*
  ** At this point we have the tuple stored in the array r[][].
  ** We need to calculate the source vector associated with the
  ** tuple in the answer to t he query. We calculate the source
  ** by taking the conjunction between the source vectors associated
  ** with the attribute values that satisfy one of the conjuncts
  ** from the array F[][].
  */
  if ( (satisfaction == 1) )
  { /*
    ** then the tuple satisfies the selection condition
    ** that we stored in F[][]. Based on that we will
    ** calculate the conjunction of those source vectors
    ** Initialize the source vector u[]
    */
    for (i = 0; i < MAXSOURCES + 1; i++) { u[i] = '\0';}

    for (i = 0; i < numattr; i++)
    { /* for begins */
      if (p[i] == 1)
      {
         Conjunction(u, r[i+regattr+1], u);
      }
    } /* for ends */
    Conjunction(u, r[regattr], u);
    /*
    ** We now can display the values of the regular attributes
    ** and the source vectors associated with those attribute
    ** values.
    */
    printf("\n");
    for (i = 0; i < regattr; i++)
    { /*
      ** print  regular attributes
      */
      printf("[%d] %s ", i+1, r[i]);
      fprintf(ofileptr, "%s~", r[i]);
    }
    printf("[%d] %s ", i+1, u);
    fprintf(ofileptr, "%s~", u);

    /*
    ** Print the source vectors associated with the corresponding
    ** attribute value
    */
    for (i = 0; i < MAXSOURCES + 1; i++) { u1[i] = '\0';}
    for (i = regattr + 1; i < numattr; i++)
    { /*
      ** print  regular attributes
      */
      Conjunction(u, r[i], u1);
      printf("[%d] %s ", i+1, u1);
      fprintf(ofileptr, "%s~", u1);
      for (i1 = 0; i1 < MAXSOURCES + 1; i1++) { u1[i1] = '\0';}
    }
    fprintf(ofileptr, "\n");
  }
  /*
  ** If the selection condition has zero attributes, then save
  ** the tuples in a file and call the Reliability Calculation
  ** procedure to sort the data, eliminate duplications, and
  ** calculate the reliability of tuples.
  */
  if (qry.ncnd == 0)
  {
    /*
    ** read the attribute values from the array r[]
    ** and print the attribute values as well as the
    ** the source vectors associated with those attributes.
    */
    printf("\n");
    for (i = 0; i < numattr; i++)
    { /*
      ** print  regular attributes
      */
      printf("[%d] %s ", i+1, r[i]);
      fprintf(ofileptr, "%s~", r[i]);
    }
    fprintf(ofileptr, "\n");
    break;
  }
  j++;
  k = 0;
  for ( i = 0 ; i < MAXATTR ; i++) { p[i] = 0;}
  } /* finish reading disjuncts */
  /*
  ** Initialize the source vector again
  */
  i = 0;
  while ( u[i] != '\0' ) {u[i++] = '\0';}
}
 inchar = getc(ifileptr);
} /* end EOF */
printf("\n");
fclose(ifileptr);
fclose(ofileptr);
ifileptr = fopen("options.rel", "r");
fscanf(ifileptr, "%d", &relFlag); /* read relFlag */
```

```
 fclose(ifileptr);
 if ( relFlag != 0 )
 { /*
   ** Reliability calculation is allowed
   */
   RelCalculation();
 }
}



/*-----------------------------------------------------------------------------------------
** Procedure: ExtendedProjection();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**---------------------------------------------------------------------------------------*/
void  ExtendedProjection()
{
 FILE *ifileptr, *ofileptr;
 int  inchar, j, i, k, relFlag;
 int  numattr, regattr, srcattr;
 char u[MAXSOURCES + 1];             /* neutral source vector                                */
 char v[MAXSOURCES + 1];             /* the read source vector                               */
 char a[MAXATTR][ATTWIDTH];          /* names of regular attributes                          */
 int  pos[MAXATTR];                  /* array of flags indicating that attributes a[i] are needed */
 char c[ATTWIDTH];                   /* contains the attribute value read                    */

 float rel;
 /*
 ** Initialize the arrays of source vectors
 */
 for (i = 0; i <= MAXSOURCES + 1; i++) {u[i] ='\0'; v[i] = '\0';}
 i = 0; j = 0;
 /*
 ** set all positions to zero and then whenever you find out that the
 ** attributes in the structures query template are found among the
 ** attributes read in the array a[][], set the corresponding position
 ** to one. This is needed to manipulate the source vectors later.
 */
 for (i = 0; i <= MAXATTR; i++) {pos[i] = 0;}

 ifileptr = fopen("query.ans", "r");
 ofileptr = fopen("query.rel", "w");

 fscanf(ifileptr, "%d", &numattr);                /* read number of attributes             */
 fprintf(ofileptr, "%d\n", (qry.nattr * 2) + 1);
 regattr = (numattr - 1)/ 2;                      /* Calculate the number of regular attributes */
 inchar = getc(ifileptr);                         /* to skip the <cr> characters            */
 inchar = getc(ifileptr);
 /*
 ** print the regular attributes names that were typed in by the user
 ** when entering the query. The attribute names are found in the Structured
 ** Query Template.
 */
 j = 1; i = 0;
 while ( j <= regattr )
 {
   if (inchar != '^')
     { /*
       ** fill an array that represents the regular scheme of qry.tbls[]
       */
       a[j-1][i++] = inchar;
```

```
    }
    else
    { /*
      ** save the attribute name
      */
      a[j-1][i] = '\0';
      i = 0;
      j++;                          /* increment j towards the number of regular attributes */
    }
  inchar = getc(ifileptr);
}
a[j-1][0] = '\0';
/*
** exit from the previous loop with a[][] filled
** with attribute names of the extended relation
** used for projection.
*/
printf("\n");

/*
** Given the array a[][] of all regular attributes
** read the Structured Query Template and select the
** appropriate attributes needed from the array a[][].
*/
j = 0;
k = 0;
while ( j < qry.nattr )
{
  i = 0;
  while ( a[i][0] != '\0')
  { /* compare and set positions */
    if ( !(EqualStrings(qry.attr[j], a[i])) )
    { /*
      ** the needed attribute is found save its position
      */
      pos[i] = 1;                     /* flag indicating that qry.attr[j] = a[i] is needed */
      printf("[%d] %s ", i+1, a[i] );
      fprintf(ofileptr, "%s~", a[i]);
      /*
      ** whenever an attribute name is needed, assume that the
      ** the attribute names are unique.
      */
      i++;
      break;                          /* and do not look for other attribute names         */
    }
    i++;
  } /* compare and set positions */
  j++;
}

/* printf("\n"); */
/*
** Skip the name of source attributes by reading
** until a '\n' is reached.
*/
while ( (inchar = getc(ifileptr) != '\n') ) {;}
fprintf(ofileptr,"\n");                    /* skip a line to print attribute values.      */
printf("\n");                              /* skip a line on the screen                   */
inchar = getc(ifileptr);                   /* skip the '\n'                               */

/*
** print the regular attribute values and
** perform the reliability calculation for
** the whole tuple.
```

```
*/
while ( inchar != EOF )
{ /* not EOF */
 while (inchar != '\n')
 { /* '\n' */
  j = 1;
  i = 0;
  c[i] = '\0';
  i = 0;
  while ( j <= regattr )
  { /*
    ** print regular attribute values corresponding to
    ** to the attributes typed in by the user. Th. attributes
    ** typed in by the user are found in the Structured Query
    ** Template.
    */
    if (inchar != '~')
    { /* print the characters */
      c[i] = inchar;
      i++;
    }
    else
    { /*
      ** if position of i in array pos[] is 1 then this attribute
      ** value is needed to be printed. Print it and increment
      ** to read another attribute value.
      */
      if (pos[j-1] == 1)
      {
        c[i] = '\0';
        printf("[%d] %s ",j, c);
        fprintf(ofileptr, "%s~", c);
      }
      i = 0;
      j++;                           /* increment j towards the number of regular attributes */
    }
    inchar = getc(ifileptr);
 } /* print regular attribute values */
 /*
 ** Read the source vectors for every tuple
 ** Do the source vector conjunction for all
 ** source vectors whose attribute names are
 ** needed by the user (Projection).  To know
 ** these sources we will use the pos[] array
 ** with a displacement to get the sources.
 */
 j = 1;
 i = 0;
 srcattr = numattr - regattr;
 while ( j <= srcattr )
 { /* start reading source vectors */
    if (inchar != '~')
    { /* print the characters
      putc(inchar, ofileptr);*/
      /*
      ** store the source vector bit by bit, then increment i
      */
      v[i++] = inchar;
      /* putchar(v[i++]);*/
    }
    else
    {
      j++;                           /* increment j towards the number of regular attributes */
      /*
```

```
        ** Call the conjunction procedure passing u[], and v[]
        ** the answer should be returned in u[]
        */
        if ( (j >= 3) && (pos[j-3] == 1) || (j == 2) )
        { /*
          ** the source vector is needed based on pos[j-2]
          */
          Conjunction(u, v, u);
          fprintf(ofileptr, "%s~", v);
          printf("%s ", v);
        }
        for (i = 0; i <= MAXSOURCES + 1; i++) {v[i] = '\0';}
        i = 0; /* prepare for the second source vector */
      }
      inchar = getc(ifileptr);
    } /* end reading source vectors */
  } /* '\n' */
  printf("     ");

  /*
  ** print the source vector that show the
  ** contributing source vectors to the whole tuple
  */
  putchar('\n');
  putc('\n', ofileptr);
  inchar = getc(ifileptr);
  /*
  ** Initialize the source vector again
  */
  i = 0;
  while ( u[i] != '\0' ) {u[i++] = '\0';}
 } /* not EOF */
 fclose(ifileptr);
 fclose(ofileptr);

 relFlag = 0;
 ifileptr = fopen("options.rel", "r");
 fscanf(ifileptr, "%d", &relFlag); /* read relFlag */
 fclose(ifileptr);
 if ( relFlag != 0 )
 { /* Reliability calculation is allowed */
   RelCalculation();
 }

}


/*-----------------------------------------------------------------------------------------
** Procedure: ExtendedUnion();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**---------------------------------------------------------------------------------------*/
void  ExtendedUnion()
{
  /*
  ** Not Separately implemented. Union is treated is selection.
  */
}
```

```
/*---------------------------------------------------------------------------------------
** Procedure: ExtendedProduct();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**--------------------------------------------------------------------------------------*/
void  ExtendedProduct()
{
/*
** Open the query.ans file name and read in the result of the query
** typed by the user. isolate the values for regular attributes
** and source if each relation. r[][] is the array that represents
** the first extended relation and s[][] is the array that represents
** the second extended relation.
*/
 FILE *ifileptr, *ofileptr;        /* pointers to input and output files              */
 int  inchar, j, i, k, relFlag;
 int  numattr, regattr, srcattr;
 char u[MAXSOURCES + 1];           /* neutral source vector                           */
 char v[MAXSOURCES + 1];           /* the read source vector                          */
 char a[MAXATTR][ATTWIDTH];        /* names of regular attributes                     */
 int  pos[MAXATTR];                /* array of flags indicating that attributes a[i] are needed */
 char c[ATTWIDTH];                 /* contains the attribute value read temporarily   */

 char r[NROWS][WIDTH];
 char s[NROWS][WIDTH];
 char h[NROWS][WIDTH];

 int hitsrc,numsrcvec;
 int srcattr1, regattr1, nattrib1;
 int srcattr2, regattr2, nattrib2;

 float rel;
 relFlag = 0;
 rel = 0; k = 0; numattr = 0; regattr = 0; srcattr = 0;
 hitsrc = 0; numsrcvec = 0;
 srcattr1 = 0; regattr1 = 0; nattrib1 = 0;
 srcattr2 = 0; regattr2 = 0; nattrib2 = 0;

 /*
 ** Initialize the arrays of source vectors to have null values
 */
 for (i = 0; i <= MAXSOURCES + 1; i++) {u[i] ='\0'; v[i] = '\0';}
 i = 0; j = 0;
 /*
 ** set all positions to zero and then whenever you find out that the
 ** attributes in the structures query template are found among the
 ** attributes read in the array a[][], set the corresponding position
 ** to one. This is needed to manipulate the source vectors later.
 */
 for (i = 0; i <= MAXATTR; i++) {pos[i] = 0;}

 /*
 ** Initialize the arrays that should contain the tuples and sources
 */
 for (i = 0; i <= NROWS; i++)
 {
   r[NROWS][0] = '\0';
   s[NROWS][0] = '\0';
   h[NROWS][0] = '\0';
 }
 /*
 ** Open files of input/output
```

```
*/
ifileptr = fopen("query.ans", "r");
ofileptr = fopen("query.rel", "w");

/*
** get the number of columns in the answer to the query
*/
fscanf(ifileptr, "%d", &numattr);               /* read number of attributes         */
inchar = getc(ifileptr);                        /* skip the '\n' character           */
/*
** read the attribute names until the end of line
*/
j = 0;
i = 0;
inchar = getc(ifileptr);
while ( j < numattr )
{
    if (inchar != '~')
    {
      h[j][i++] = inchar;
    }
    else
    {
      h[j][i] = '\0';
      i = 0;
      j++;                          /* increment j towards the number of regular attributes */
    }
  inchar = getc(ifileptr);
}

/*
** Now, it is the time to read the attribute values and the source vectors
** to ONE array r[][]. During the reading process we will gather information
** about the number of attributes of the first extended relation and the
** second extended relation.
*/
j = 0;
i = 0;
inchar = getc(ifileptr);                        /* skip '\n' character               */
while ( j < numattr )
{
    if (inchar != '~')
    {
      r[j][i++] = inchar;
      if (inchar == '0' || inchar == '1' || inchar == '-' )
      {
        hitsrc++;
      }
    }
    else
    {
      regattr1++;
      r[j][i] = '\0';
      if ( hitsrc == MAXSOURCES )
      {
        regattr1--;
        hitsrc = 0;
        i = 0;
        j++;
        break;
      }
      hitsrc = 0;
      i = 0;
      j++; /* increment j towards the number of regular attributes */
```

```
     }
   inchar = getc(ifileptr);
  }

  /*
  ** When break is executed we have to read till the end of the
  ** tuple. The values for regular attributes of the first extended
  ** relation and the source vector associated with the tuple are
  ** read and stored in r[][]. Read the rest.
  */
  inchar = getc(ifileptr); /* skip '\n' character */
  while ( j < numattr )
  {
      if (inchar != '^')
      {
        r[j][i++] = inchar;
      }
      else
      {
        r[j][i] = '\0';
        i = 0;
        j++;                              /* increment j towards the number of regular attributes */
      }
    inchar = getc(ifileptr);
  }

  /*
  ** At this point we have the number of regular attributes of the first
  ** extended relation and the number of source attributes of the first
  ** extended relation too. We will calculate the number of regular attributes
  ** and the number of source attributes of second relation.
  */
  srcattr1 = regattr1 + 1;
  nattrib1 = regattr1 + srcattr1;
  nattrib2 = numattr - nattrib1;
  regattr2 = (nattrib2 - 1) / 2;
  srcattr2 = regattr2 + 1;

  /*
  ** save the number of attributes and the attribute names
  */
  fprintf(ofileptr, "%d\n", nattrib1 + nattrib2 - 1);

  /*
  ** print the attribute names in the file
  */
  for (i = 1; i <= regattr1; i++)
  {
    fprintf(ofileptr, "%s^", h[i-1]);
    printf("[%d] %s ", i, h[i-1]);
  }

  for (i = nattrib1; i < nattrib1 + regattr2; i++)
  {
    fprintf(ofileptr, "%s^", h[i]);
    printf("[%d] %s ", i - srcattr1 + 1, h[i]);
  }

  fprintf(ofileptr, "\n");
  printf("\n");

 while (inchar != EOF)
 {
  /*
```

```
** print the tuple and the source vectors
*/
for (i = 1; i <= regattr1; i++)
{
  printf("[%d] %s ", i, r[i-1]);
  fprintf(ofileptr, "%s~", r[i-1]);
}

for (i = nattrib1; i < nattrib1 + regattr2; i++)
{
  printf("[%d] %s ", i - srcattr1 + 1, r[i]);
  fprintf(ofileptr, "%s~", r[i]);
}

/*
** Perform the conjunction between the source vectors associated with
** the tuples in each extended relation. The answer will be in u[].
*/
Conjunction(r[regattr1], r[nattrib1+regattr2], u);
printf("%s ", u);
fprintf(ofileptr, "%s~", u);

/*
** Print source vectors associated with the attribute values of
** the first extended relation.
*/
for (i = regattr1 + 1; i <= nattrib1 - 1; i++)
{
  Conjunction(u, r[i], v);
  printf("%s ", v);
  fprintf(ofileptr, "%s~", v);
}

/*
** Print source vectors associated with the attribute values of
** the second extended relation.
*/
for (i = nattrib1 + regattr2 + 1; i <= nattrib1 + nattrib2 - 1; i++)
{
  Conjunction(u, r[i], v);
  printf("%s ", v);
  fprintf(ofileptr, "%s~", v);
}
/*
** read another tuple from the file
*/
j = 0;
i = 0;
inchar = getc(ifileptr); /* skip '\n' character */
while ( j < numattr )
{
    if (inchar != '~')
    {
      r[j][i++] = inchar;
    }
    else
    {
      r[j][i] = '\0';
      i = 0;
      j++;                          /* increment j towards the number of regular attributes */
    }
  inchar = getc(ifileptr);
}
j = 0;
```

```
 i = 0;
 fprintf(ofileptr, "\n");
 printf("\n");
} /* read, manipulate, and print tuples */

fclose(ifileptr);
fclose(ofileptr);
 /*
 ** Open "options.rel" file and read the relFlag.
 ** if the relFlag = 1, then calculate the reliability
 ** of answers to users queries.
 */
 relFlag = 0;
 ifileptr = fopen("options.rel", "r");
 fscanf(ifileptr, "%d", &relFlag); /* read relFlag */
 fclose(ifileptr);
 if ( relFlag != 0 )
 { /*
   ** Reliability calculation is allowed
   */
   RelCalculation();
 }
}



/*------------------------------------------------------------------------------------------
** Procedure: ExtendedJoin();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**-----------------------------------------------------------------------------------------*/
void  ExtendedJoin()
{
/*
** Open the query.ans file name and read in the result of the query
** typed by the user. isolate the values for regular attributes
** and source attributes of each relation. r[][] is the array that represents
** the first extended relation and s[][] is the array that represents
** the second extended relation.
*/
 FILE *ifileptr, *ofileptr;                        /* pointers to input and output files        */
 int  inchar, j, i, k, relFlag, equalFlag;
 int  numattr, regattr, srcattr, numattrinx;
 char u[MAXSOURCES + 1];                           /* neutral source vector                     */
 char v[MAXSOURCES + 1];                           /* the read source vector                    */
 char a[MAXATTR][ATTWIDTH];                        /* names of regular attributes               */
 int  pos[MAXATTR];              /* array of flags indicating that attributes a[i] are needed */
 char c[ATTWIDTH];                           /* contains the attribute value read temporarily */
 char x[MAXATTR][ATTWIDTH];                          /* is the set of common attributes         */

 char r[NROWS][WIDTH];
 char s[NROWS][WIDTH];
 char h[NROWS][WIDTH];                             /* array for the heading names               */

 int hitsrc, numsrcvec;
 int srcattr1, regattr1, nattrib1;
 int srcattr2, regattr2, nattrib2;

 float rel;
 equalFlag = 0;
 numattrinx = 0;
 relFlag = 0;
```

```
rel = 0; k = 0; numattr = 0; regattr = 0; srcattr = 0;
hitsrc = 0; numsrcvec = 0;
srcattr1 = 0; regattr1 = 0; nattrib1 = 0;
srcattr2 = 0; regattr2 = 0; nattrib2 = 0;

/*
** Initialize the arrays of source vectors to have null values
*/
for (i = 0; i <= MAXSOURCES + 1; i++) {u[i] ='\0'; v[i] = '\0';}
i = 0; j = 0;

/*
** set all positions to zero and then whenever you find out that the
** attributes in the structured query template are found among the
** attributes read in the array a[][], set the corresponding position
** to one. This is needed to manipulate the source vectors later.
*/
for (i = 0; i <= MAXATTR; i++) {pos[i] = 0; x[i][0] = '\0';}

/*
** Initialize the arrays that should contain the tuples and sources
*/
for (i = 0; i <= NROWS; i++)
{
  r[NROWS][0] = '\0';
  s[NROWS][0] = '\0';
  h[NROWS][0] = '\0';
}

/*
** Open files for input/output
*/
ifileptr = fopen("query.ans", "r");
ofileptr = fopen("query.rel", "w");

/*
** get number of columns in the answer to the query
*/
fscanf(ifileptr, "%d", &numattr); /* read number of attributes */
inchar = getc(ifileptr); /* skip the '\n' character */

/*
** Find the set of joining attributes
*/
k = 0;
i = 0;
while (1)
{ /* Finding the set of joining attributes */
  /*
  ** Since the extended Join requires common attributes. We will read the
  ** common attributes from the Structured Query Template and isolate the
  ** attributes that form the Join. Read until an equal sign is reached.
  */
  while ( (qry.cnd[0][i] != '=') && (qry.cnd[0][i] != '\0') ){i++;}
  if (qry.cnd[0][i++] == '\0'){break;}

  /*
  ** When an equal sign is reached, read until a dot '.' is reached
  */
  while (qry.cnd[0][i++] != '.'){;}

  /*
  ** At this point, the first character of the attribute needed is found
  ** right after the dot '.' found. Read the attribute and store its name
  ** in an array for future use. The attribute name will end as soon as
  ** a space is reached.
  */
  j = 0;
  while ( (qry.cnd[0][i] != ' ') && (qry.cnd[0][i] != '\0'))
  { /*
    ** save the attribute name in x[][]
    */
    x[k][j] = qry.cnd[0][i];
    i++;
    j++;
 }
  numattrinx++;
  x[k][j] = '\0';

  /*
  ** Read until an '=' is reached or an '\0' is reached.
  ** if an end of line is reached there are no more attributes
  ** to abstract. If an '=' is reached loop again to abstract
  ** another attribute and store it in x[][].
  */
  k++;
  printf("set of common attributes: <%s>\n", x[k-1]);
} /* end finding the set of common attributes */

/*
** read the attribute names from the file pointed to by ifileptr
** until the end of line is reached.
*/
j = 0;
i = 0;
inchar = getc(ifileptr);
while ( j < numattr )
{
    if (inchar != '~')
    {
      h[j][i++] = inchar;
    }
    else
    {
      h[j][i] = '\0';
      i = 0;
      j++;                              /* increment j towards the number of regular attributes */
    }
  inchar = getc(ifileptr);
}

/*
** Now, it is the time to read attribute values and the source vectors
** to ONE array r[][]. During the reading process we will gather information
** about the number of attributes of the first extended relation and the
** second extended relation.
*/
j = 0;
i = 0;
inchar = getc(ifileptr); /* skip '\n' character */
while ( j < numattr )
{
    if (inchar != '~')
    {
      r[j][i++] = inchar;
      if (inchar == '0' || inchar == '1' || inchar == '-' )
      {
        bitsrc++;
        }
      }
      else
      {
        regattri++;
        r[j][i] = '\0';
        if ( hitsrc == MAXSOURCES )
        {
          regattri--;
          hitsrc = 0;
          i = 0;
          j++;
          break;
        }
        hitsrc = 0;
        i = 0;
        j++;                            /* increment j towards the number of regular attributes */
      }
    inchar = getc(ifileptr);
  }

  /*
  ** When break is executed we have to read till the end of the
  ** tuple. The values for regular attributes of the first extended
  ** relation and the source vector associated with the tuple are
  ** read and stored in r[][]. Read the rest.
  */
  inchar = getc(ifileptr); /* skip '\n' character */
  while ( j < numattr )
  {
      if (inchar != '~')
      { /* print the characters
        putc(inchar, ofileptr);
        putchar(inchar);*/
        r[j][i++] = inchar;
      }
      else
      {
        r[j][i] = '\0';
        i = 0;
        j++;                            /* increment j towards the number of regular attributes */
      }
    inchar = getc(ifileptr);
  }

  /*
  ** At this point we have the number of regular attributes of the first
  ** extended relation and the number of source attributes of the first
  ** extended relation too. We will calculate the number of regular attributes
  ** and the number of source attributes of second relation.
  */
  srcattri = regattri + 1;                    /* source attributes in 1st relation         */
  nattrib1 = regattri + srcattri;             /* attributes in first relation              */
  nattrib2 = numattr - nattrib1;              /* attributes of second relation             */
  regattr2 = (nattrib2 - 1) / 2;              /* regular attributes in second relation     */
  srcattr2 = regattr2 + 1;                    /* source attributes in second relation      */

  /*
  ** save the number of attributes and the attribute names
  */
  fprintf(ofileptr, "%d\n", nattrib1 + nattrib2 - numattrinx - numattrinx - 1);

  /*
  ** print and save the attribute names of the first extended relation
```

```
 */
 for (i = 1; i <= regattr1; i++)
 {
   fprintf(ofileptr, "%s~", h[i-1]);
   printf("[%d] %s ", i, h[i-1]);
 }

 /*
 ** Print and save the attribute names of the second extended relation.
 ** Remember that attribute names that are in the set of joining attributes
 ** should not be printed again.
 */
 for (i = nattrib1; i < nattrib1 + regattr2; i++)
 {
  /*
  ** before printing the attribute name we have to make sure that
  ** the attribute name is not in the set of joining attributes.
  */
  equalFlag = 0;
  k = 0;
  while (x[k][0] != '\0')
  {
     if ( !(EqualStrings(x[k], h[i])) )
     {
        equalFlag = 1;
        break;
     }
     k++;
  }
     if (equalFlag != 1)
     {
       fprintf(ofileptr, "%s~", h[i]);
       printf("[%d] %s ", i - srcattr1 + 1, h[i]);
     }
 }

 fprintf(ofileptr, "\n");
 printf("\n");

while (inchar != EOF)
{/* read, manipulate, and print tuples */
 /*
 ** print the tuple and the source vectors of the first
 ** extended relation. Some of the attributes are
 ** in the set of Joining attributes.
 */
 for (i = 1; i <= regattr1; i++)
 {
   printf("[%d] %s ", i, r[i-1]);
   fprintf(ofileptr, "%s~", r[i-1]);
 }

 /*
 ** Print and save the attribute values of the second
 ** extended relation.
 */
 for (i = nattrib1; i < nattrib1 + regattr2; i++)
 {
  /*
  ** before printing the attribute values we have to make sure that
  ** the attribute value is not in the set of joining attributes.
  */
  equalFlag = 0;
  k = 0;
 while (x[k][0] != '\0')
 {
    if ( !(EqualStrings(x[k], h[i])) )
    {
       equalFlag = 1;
       break;
    }
    k++;
 }
 if (equalFlag != 1)
 {
   printf("[%d] %s ", i - srcattr1 + 1, r[i]);
   fprintf(ofileptr, "%s~", r[i]);
 }
}

/*
** Perform the conjunction between the source vector associated with
** the tuple in the first extended relation, the source vector associated
** with the tuple of the second extended relation and the source vectors
** associated with the joining attributes. The answer will be in u[].
*/
Conjunction(r[regattr1], r[nattrib1+regattr2], u);

/*
** Look for source vectors associated with the joining attribute values
** of the first extended relation.
*/
for (i = 1; i <= regattr1; i++)
{
  equalFlag = 0;
  k = 0;
  while (x[k][0] != '\0')
  {
     if ( EqualStrings(x[k], h[i-1]) )
     {
        equalFlag = 1;
        break;
     }
     k++;
  }
  if (equalFlag != 1)
  { /* perform conjunction */
    Conjunction(u, r[regattr1 + i], u);
  }
}

/*
** Look for source vectors associated with the joining attribute values
** of the second extended relation.
*/
for(i = nattrib1; i < nattrib1 + regattr2; i++)
{
  equalFlag = 0;
  k = 0;
  while (x[k][0] != '\0')
  {
     if ( EqualStrings(x[k], h[i]) )
     {
        equalFlag = 1;
        break;
     }
     k++;
  }
```

```
      if (equalFlag != 1)
      { /* perform conjunction */
        Conjunction(u, r[regattr2 + i + 1], u);
      }
    }
    printf("%s ", u);
    fprintf(ofileptr, "%s~", u);

    /*
    ** Print source vectors associated with the attribute values of
    ** the first extended relation.
    */
    for (i = regattr1 + 1; i <= nattrib1 - 1; i++)
    {
      Conjunction(u, r[i], v);
      printf("%s ", v);
      fprintf(ofileptr, "%s~", v);
    }

    /*
    ** Print source vectors associated with the attribute values of
    ** the second extended relation. Remember to skip the source vectors
    ** that are associated with the values of the joining attributes.
    */
    for (i = nattrib1 + regattr2 + 1; i <= nattrib1 + nattrib2 - 1; i++)
    {
     equalFlag = 0;
      k = 0;
      while (x[k][0] != '\0')
      {
        if ( !(EqualStrings(x[k], h[i-regattr2-1])) )
        {
          equalFlag = 1;
          break;
        }
        k++;
      }
      if (equalFlag != 1)
      { /* perform conjunction */
        Conjunction(u, r[i], v);
        printf("%s ", v);
        fprintf(ofileptr, "%s~", v);
      }
    }

    /*
    ** read another tuple from the file
    */
    j = 0;
    i = 0;
    inchar = getc(ifileptr);                                        /* skip '\n' character */
    while ( j < numattr )
    {
        if (inchar != '~')
        {
          r[j][i++] = inchar;
        }
        else
        {
          r[j][i] = '\0';
          i = 0;
          j++;                         /* increment j towards the number of regular attributes */
        }
      inchar = getc(ifileptr);
```

```
  }
  j = 0;
  i = 0;
  fprintf(ofileptr, "\n");
  printf("\n");
 }/* read, manipulate, and print tuples */

 fclose(ifileptr);
 fclose(ofileptr);
  /*
  ** Open "options.rel" file and read the relFlag.
  ** if the relFlag = 1, then calculate the reliability
  ** of answers to users queries.
  */
  relFlag = 0;
  ifileptr = fopen("options.rel", "r");
  fscanf(ifileptr, "%d", &relFlag); /* read relFlag */
  fclose(ifileptr);
  if ( relFlag != 0 )
  { /* Reliability calculation is allowed */
    RelCalculation();
  }

 } /* end procedure ExtendedJoin() */


/*---------------------------------------------------------------------------------
** Procedure: ExtendedIntersection();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**-------------------------------------------------------------------------------*/
void  ExtendedIntersection()
{
 /* Not implemented separately. It is a special Join */
}


/*---------------------------------------------------------------------------------
** Procedure: ExtendedDifference();
**   Purpose: Reads a file and prints the heading,
**            attribute values, and the reliability
**            of each attribute.
**    return: Non
**-------------------------------------------------------------------------------*/
void  ExtendedDifference()
{
  /* Not Implemented */
}


/*---------------------------------------------------------------------------------
** Procedure: Algorithm1();
**   Purpose: pass an array of source vectors associated with a pure
**            tuple in the answer to a query. This Procedure will
**            calculate the reliability of the resulting array.
**            One restriction in this Algorithm is that we assume
**            no more than a fixed length array should be passed.
**    return: Non.
**-------------------------------------------------------------------------------*/
```

```
float Algorithm1(x)
 char x[MAXTUPLES][MAXSOURCES + 1];                  /* passed array of source vectors              */
{
 char ui[MAXSOURCES + 1];
 char uj[MAXSOURCES + 1];
 char uk[MAXSOURCES + 1];
 char ul[MAXSOURCES + 1];
 char um[MAXSOURCES + 1];
 char un[MAXSOURCES + 1];
 char uo[MAXSOURCES + 1];
 char up[MAXSOURCES + 1];
  /* begin Algorithm1 */

   int numvec;                                        /* number of source vectors in set          */
   int i, j, k, l, m, n, o, p;
   float K, K1, K2, K3, K4, K5, K6, K7, K8, rel;

   /* Initialization */
   i = 1; j = 2; k = 3; l = 4; m = 5;
   K = 0; K1 = 0; K2 = 0; K3 = 0; K4 = 0;
   K5 = 0; K6 = 0; K7 = 0; K8 = 0; rel  = 0;
   /* example source vector
   x[1][0] = '0'; x[1][1] = '1'; x[1]'2] = '0'; x[1][3] = '1'; x[1][4] = '1';
   x[1][5] = '0'; x[1][6] = '0'; x[1][7] = '1'; x[1][8] = '0'; x[1][9] = '1';
   x[1][10] = '\0';

   x[2][0] = '1'; x[2][1] = '1'; x[2][2] = '0'; x[2][3] = '1'; x[2][4] = '0';
   x[2][5] = '0'; x[2][6] = '1'; x[2][7] = '1'; x[2][8] = '0'; x[2][9] = '1';
   x[2][10] = '\0';

   x[3][0] = '0'; x[3][1] = '0'; x[3][2] = '0'; x[3][3] = '1'; x[3][4] = '0';
   x[3][5] = '0'; x[3][6] = '1'; x[3][7] = '0'; x[3][8] = '0'; x[3][9] = '0';
   x[3][10] = '\0';

   x[4][0] = '\0'; x[4][1] = '\0'; x[4][2] = '\0'; x[4][3] = '\0'; x[4][4] = '\0';
   x[4][5] = '\0'; x[4][6] = '\0'; x[4][7] = '\0'; x[4][8] = '\0'; x[4][9] = '\0';
   x[4][10]= '\0';
   */
   /*
   ** Browse the array and check if there are duplicates among the
   ** source vectors sent. The result array should not have any duplicates.
   */
   i = 0;
   while ( (x[i][0] != '\0') && (x[i+1][0] != '\0') )
   {
     if ( !(CmpSourceVectors(x[i], x[i+1])) )
     {
       /*
       ** they are equal then keep one of them and remove the other one.
       ** Search the end of source vectors and switch with the last. DO
       ** forget to overwrite the last source vector with '\0'. Since
       ** later we need to count the number different sources we have.
       */
          j = i + 2;
          while ( x[j++][0] != '\0' ){;}
          j = j - 2;
          /* swap the two source vectors */
          k = 0;
          while ( x[j][k] != '\0' )
          {
            x[i][k] = x[j][k]; /* swap      */
            x[j][k] = '\0';    /* overwrite */
            k++;
          }
```

```
    }
    i++;
}
/* count the number of source vectors sent */
i = 0;
while ( x[i][0] != '\0' ){i++;}
numvec = i;
for ( i = 0; i < numvec; i++)
{ /* begin i loop */
  /* calculate the reliability of x[i] */
  rel = SourceVectorReliability(x[i]);
  /* accumulate K1 */
  K1 = K1 + rel;
  for ( j = i+1; j < numvec; j++)
  { /* begin j loop */
    Conjunction(x[i], x[j], uj);
    rel = SourceVectorReliability(uj);
    /* accumulate K2 */
    K2 = K2 + rel;
    for ( k = j+1; k < numvec; k++)
    { /* begin k loop */
      /* uj = x[i] /\ x[j] */
      Conjunction(uj, x[k], uk);
      rel = SourceVectorReliability(uk);
      /* accumulate K3 */
      K3 = K3 + rel;
      for ( l = k+1; l < numvec; l++)
      { /* begin l loop */
        Conjunction(uk, x[l], ul);
        rel = SourceVectorReliabilitv(ul);
        /* accumulate K4 */
        K4 = K4 + rel;
        for ( m = l+1; m < numvec; m++)
        { /* begin m loop */
          Conjunction(ul, x[m], um);
          rel = SourceVectorReliability(um);
          /* accumulate K5 */
          K5 = K5 + rel;
          for ( n = m+1; n < numvec; n++)
          { /* begin n loop */
            Conjunction(um, x[n], un);
            rel = SourceVectorReliability(un);
            /* accumulate K6 */
            K6 = K6 + rel;
            for ( o = n+1; o < numvec; o++)
            { /* begin o loop */
              Conjunction(un, x[o], uo);
              rel = SourceVectorReliability(uo);
              /* accumulate K7 */
              K7 = K7 + rel;
              for (p = o+1; p < numvec; p++)
              { /* begin p loop */
                Conjunction(uo, x[p], up);
                rel = SourceVectorReliability(up);
                /* accumulate K7 */
                K8 = K8 + rel;
                /* accumulate K6 */
              } /* end p loop */
            } /* end o loop */
          } /* end n loop */
        } /* end m loop */
      } /* end l loop */
    } /* end k loop */
  } /* end j loop */
```

```
   } /* end i loop */
  K = K1 - K2 + K3 - K4 + K5 - K6 + K7 - K8;
  /* printf(" K = %f", K ); */
  return K;
  }/* end Algorithm1 */


/*-------------------------------------------------------------------------------------
** Procedure: SourceVectorReliability()
**    Purpose: Given a source vector, this procedure calculates the
**             reliability of the given source vector and returns
**             the reliability. Reliability of source vectors is to be
**             read from a file.
**     return: the reliability of the source vector of type float.
**             float rel;
**
**-----------------------------------------------------------------------------------*/
float SourceVectorReliability(u1)
char u1[MAXSOURCES + 1];
{

  int i;

  /*
  ** the reliabilities of information sources is be read from a file.
  ** the file name is "options.rel". We assume that we keep track of
  ** one file and save in it the reliabilities of informaion sources.
  */
  float rel;

  /*
  ** Reliability Calculation for a single
  ** source vector only.
  */
  i = 0;
  rel = 1;
  while ( u1[i] != '\0' )
  {
   if (u1[i] == '1')
   {
     rel = rel * re[i+1];
   }
   else
   {
     if (u1[i] == '-')
     {
       rel = rel * ( 1 - re[i+1]);
     }
   }
  i++;
  }
 return(rel);
}


/*-------------------------------------------------------------------------------------
** Procedure: CmpSourceVectors().
**
**   Purpose: Compares two character arrays and returns zero if they
**            are equal else returns a non zero if they are not equal.
**    return: integer number indicating whether the two characters arrays
**            are equal or not.
```

```
**-----------------------------------------------------------------------------------------------*/
int CmpSourceVectors(s, t)
char s[MAXSOURCES + 1];
char t[MAXSOURCES + 1];
{
  int i;

  for( i = 0; (s[i] == t[i]) && (s[i] != '\0') && (t[i] != '\0'); i++);
      if  ( (s[i] == '\0') && (t[i] == '\0') )
         return 0;
  return s[i] - t[i];
}



/*-----------------------------------------------------------------------------------------------
** Procedure: EqualStrings().
**
**   Purpose: Compares two character arrays and returns zero if they
**            are equal else returns a non zero if they are not equal.
**    return: integer number indicating whether the two characters arrays
**            are equal or not.
**-----------------------------------------------------------------------------------------------*/
int EqualStrings(a1, a2)
char a1[MAXSOURCES + 1];
char a2[MAXSOURCES + 1];
{
  int i;

  for( i = 0; (a1[i] == a2[i]) && (a1[i] != '\0') && (a2[i] != '\0'); i++);
      if  ( (a1[i] == '\0') && (a2[i] == '\0') )
         return 0;
  return a1[i] - a2[i];
}



/*-----------------------------------------------------------------------------------------------
** Purpose: Opens a file for input, read tuples, perform the
**          source vector conjunction operation for all sources
**          associated with the tuples, and finds the reliability
**          of the pure tuples. The tuples are displayed to the
**          screen of the user together with the reliabilities
**          associated with the pure tuples.
**
**  return: Non.
**-----------------------------------------------------------------------------------------------*/
void RelCalculation()
{
  typedef struct tuples {
  char pure[MAXLINE];                        /* pure tuple                          */
  char vec[MAXSOURCES+1];                    /* source vector                       */
 } tup;

 FILE *ifileptr, *ofileptr;
 int  inchar, j, i, k, i1, i2;
 int  numattr,  regattr, srcattr, numtuples, lastentered;
 char u[MAXSOURCES + 1];                     /* neutral source vector               */
 char v[MAXSOURCES + 1];                     /* the read source vector              */
 char c[WIDTH];                              /* maximum width of a column           */
 tup  arr[MAXTUPLES];                        /* array to hold read tuples           */
 char set[MAXTUPLES][MAXSOURCES+1];          /* set of source vectors associated with t   */

 float rel;
```

```
/*
** Initialization
*/
i = 0; j = 0; inchar = 0; numattr = 0;
regattr = 0; srcattr = 0; k =0; i2 = 0;
/*
** Initialize the arrays of source vectors
*/

for (i = 0; i <= MAXSOURCES + 1; i++) {u[i] ='\0'; v[i] = '\0';}
for (i = 0; i <= WIDTH; i++) { c[i] = '\0';}

for (i = 0; i <= MAXTUPLES; i++)
{
    arr[i].pure[0] = '\0';
    arr[i].vec[0] = '\0';
    set[i][0] = '\0';
}
i = 0; j = 0;
/*
** Call the SortFile() procedure to sort the tuples that are
** the answers to the users query.
*/
SortFile();

ifileptr = fopen("query.rel", "r");

fscanf(ifileptr, "%d", &numattr);              /* read number of attributes                 */
regattr = (numattr - 1)/2;
inchar = getc(ifileptr);                       /* to skip the <cr> characters                */
inchar = getc(ifileptr);

/*
** print the regular attributes names only
*/
j = 1;
i = 0;
while ( j <= regattr )
{
  if (inchar != '~')
    { /* print the characters
      putchar(inchar);*/
      c[i++] = inchar;
    }
    else
    {
      c[i] = '\0';
      printf("[%d] %s ", j, c);
      j++;                          /* increment j towards the number of regular attributes */
      i = 0;
    }
  inchar = getc(ifileptr);
}
printf("\n");

numtuples = 0;
lastentered = numtuples;

inchar = getc(ifileptr);
while ( inchar != EOF )
{ /* not EOF */
 k = 0; /* counts the number of tuples read */
 while (inchar != '\n')
```

```
{ /* '\n' */
 j = 1;
 i = 0;
 while ( j <= regattr )
 { /*
   ** print regular attribute values
   */
   if (inchar != '~')
   {
     c[i++] = inchar;
   }
   else
   {
     c[i++] = inchar;
     j++;                             /* increment j towards the number of regular attributes */
   }
   inchar = getc(ifileptr);
 } /* print regular attribute values */

 /*
 ** Read the source vectors for every tuple
 ** Do the source vector conjunction for all
 ** source vectors found to get one source
 ** vector that represents the contributing
 ** sources to that tuple.
 */
 j = 1;
 i = 0;
 srcattr = numattr - regattr;
 while ( j <= srcattr )
 { /* start reading source vectors */
    if (inchar != '~')
    { /* print the characters
      putc(inchar, ofileptr); */
      /*
      ** store the source vector bit by bit, then increment i
      */
      v[i++] = inchar;
    }
    else
    {
      j++;                            /* increment j towards the number of regular attributes */
      /*
      ** Call the conjunction procedure passing u[], and v[]
      ** the answer should be returned in u[]
      */
      Conjunction(u, v, u);
      for (i = 0; i <= MAXSOURCES + 1; i++) {v[i] = '\0';}
      i = 0; /* prepare for the second source vector */
    }
    inchar = getc(ifileptr);
 } /* end reading source vectors */

 /*
 ** before reading another line of input from the sorted file
 ** scan the array ar[][][] ans test if the newly read tuple
 ** is previously stored in it.
 */
 if (  ( (EqualStrings(c, arr[lastentered].pure)) ||
        (EqualStrings(u, arr[lastentered].vec)) )  )
 {
     /*
     ** save the pure tuple and the source vector associated with
     ** it to the array arr[]
```

```
        */

        i1 = 0;
        while ( c[i1] != '\0' )
        {
          arr[numtuples].pure[i1] = c[i1];         /* save pure tuple                        */
          i1++;
        }
        arr[numtuples].pure[i1++] = '\0';

        i1 = 0;
        while ( u[i1] != '\0' )
        {
          arr[numtuples].vec[i1] = u[i1];          /* its source vector                      */
          i1++;
        }
        arr[numtuples].vec[i1++] = '\0';
        lastentered = numtuples;
        numtuples++;
   }
  } /* '\n' */
  k++;
  inchar = getc(ifileptr);
  /*
  ** Initialize the source vector again
  */
  i1 = 0;
  while ( u[i1] != '\0' ) { u[i1++] = '\0';}
 }/* not EOF */
 fclose(ifileptr);

 /*
 ** Read the array arr[] selecting one tuple at a time. A pure tuple
 ** could have a set of source vectors associated with it. We would
 ** like to calculate the reliability of the pure tuple t using
 ** Algorithm1().
 */
 i = 0;                                    /* runs over the pure tuples in arr[]          */
 while ( (arr[i].pure[0] != '\0')  &&  (arr[i].vec[0] != '\0') )
 {
  /*
  ** compare the ith and the (i+1)th pure tuples. In case they are equal
  ** add the source vector associated with the (i+1)th tuple to the array
  ** set[]. set[] represents the set of source vectors associated with the
  ** pure tuple being read. Before any comparison happens, save the first
  ** source vector associated with the pure tuple t into set[].
  */
  j = 0;                                   /* counts the number of source vectors in array set[] */
  k = 0;
  while (arr[i].vec[k] != '\0')
  {
    set[j][k] = arr[i].vec[k];
    k++;
  }
  set[j][k] = '\0';
  j++;

  i1 = i;
  while ( !EqualStrings(arr[i1].pure, arr[i1+1].pure) )
  {
    drows = drows + 1;

    /*
    ** Print spaces instead of duplicating the pure tuple.
    k = 0;
    while (arr[i1+1].pure[k] != '\0')
    {
      printf(" ");
      k++;
    }
    */
    /*
    ** print the source vector

    printf(" %s", arr[i1+1].vec);*/

    /*
    ** add the source vectors associated with arr[i+1].pure to set[j]
    */
    k = 0;
    while (arr[i1+1].vec[k] != '\0')
    {
      set[j][k] = arr[i1+1].vec[k];
      k++;
    }
    set[j][k] = '\0';
    j++;
    i1++;
  }

  /*
  ** at this point we have the pure tuple in arr[] stored at the ith
  ** position and the set of source vectors set[] contains K+1 sources
  ** associated with the pure tuple. Call the reliability calculation
  ** algorithm Algorithm1() to find the reliability associated with the
  ** pure tuple.
  */
  rel = 0;
  rel = Algorithm1(set);
  /*
  ** before printing the attribute values filter out the field delimiters\
  */
  i2 = 0;
  while (arr[i].pure[i2] != '\0')
  {
    if (arr[i].pure[i2] == '~')
      arr[i].pure[i2] = ' ';
    i2++;
  }
  printf("%s %s  %8.4f \n",arr[i].pure, arr[i].vec, rel);

  i2 = 1;
  while (set[i2][0] != '\0')
  {
    /*
    ** print spaces instead of the pure tuple
    */
    k = 0;
    while (arr[i].pure[k] != '\0')
    {
      printf(" ");
      k++;
    }
    /*
    ** print the source vector associated with the
    ** pure tuple.
    */
```

```
      printf(" %s\n", set[i2]);

      i2++;
    }

    i = i1;

  /*
  ** Initialize the array that contains the set of source vectors
  ** set[] associated with a single pure tuple.
  */
  for (k = 0; k <= MAXTUPLES; k++)
  {
    set[k][0] = '\0';
  }
  k = 0;
  i++;
 }
}



/*--------------------------------------------------------------------------------------------
** Procedure: str2flt()
**   Purpose: Convert a string into a float. All the elements of the
**            passed string must be digits between 0 and 9.
**    return: a number that represents the float of the passed
**            string.
**
**      Note: It is the responsibility of the calling function to
**            to pass a string whose elements are not characters.
**-------------------------------------------------------------------------------------------*/
double str2flt (s)
char s[WIDTH + 1];
{

 double val, power;
 int i, sign;

 for (i = 0; isspace(s[i]); i++) /* skip spaces */
   ;
 sign = (s[i] == '-') ? -1 : 1;
 if (s[i] == '+' || s[i] == '-')
      i++;
 for (val = 0.0; isdigit(s[i]); i++)
      val = 10.0 * val + (s[i] - '0');
 if (s[i] == '.')
      i++;
 for (power = 1.0; isdigit(s[i]); i++) {
      val = 10.0 * val + (s[i] - '0');
      power *= 10.0;
 }
 return sign * val / power;

}



/*--------------------------------------------------------------------------------------------
** Procedure: SortFile()
**   purpose: Sort a file of records. The file contains the answers
**            to users queries after a query is presented to the system.
**            We need to sort such a file because we want to have all
**            the source vectors associated with a pure tuple to find
**          the reliability of the pure tuple.
**-------------------------------------------------------------------------------------------------*/
void SortFile()
{
  char ln[MAXTUPLES][MAXLINE];
  char temp[MAXLINE];
  int  i, j, i1, k, satisfaction;
  FILE *ifileptr;
  int inchar;


 ifileptr = fopen("query.rel", "r");

 for (i = 0; i < MAXTUPLES; i++){ln[i][0] = '\0';}
 for (i = 0; i < MAXLINE; i++){temp[i] = '\0';}

 i = 0; j = 0; i1 = 0; k = 0; satisfaction = 0;
 inchar = getc(ifileptr);
 while (inchar != EOF)
 { /* read until end of file */
  j = 0;
  while (inchar != '\n')
  { /* read until end of line */
    ln[i][j] = inchar;
    /* putchar(ln[i][j]);*/
    j++;
    inchar = getc(ifileptr);
  }
  ln[i][j] = '\0';
  /*
  printf("%s", ln[i]);
  printf("\n"); */
  inchar = getc(ifileptr);
  j = 0;
  i++;
 }
 fclose(ifileptr);

 /*
 ** Sort the array read from the file.
 */
 i = 2;
 j = 0;
 while (ln[i][0] != '\0')
 {
   k = 0;
   i1 = 0;
   while ( ln[i][i1] != '\0')
   { /*
     ** save ln[i] in a temporary array tem[] (as a pivot)
     */
     temp[i1] = ln[i][i1];
     i1++;
   }
   temp[i1] = '\0';
   j = i + 1;
   while (ln[j][0] != '\0')
   { /*
     ** read the rest of the array and try to find who is
     ** the smallest among the tuples
     */
      for ( i1 = 0; (temp[i1] == ln[j][i1]) &&
                    (temp[i1] != '\0') &&
                    (ln[j][i1] != '\0'); i1++);
    if ( (temp[i1] == '\0')  )
    { /* they are less or equal */
      satisfaction = 1;
    }
    else
    { /*
      ** test if opr1[i] is less than opr2[i] by subtracting one from the other
      */
       if ( (temp[i1] - ln[j][i1] <= 0 )  )
       { /*
         ** then opr1[] is less than opr2[]
         */
          satisfaction = 1; /* ln[i][i1] is less than ln[j][i1] */
       }
       else
       {  /*
          ** we need to keep a pointer to ln[j][i1]
          */
          satisfaction = 0;
          /*
          ** save the value of j in k
          */
          k = j;
          /*
          ** copy ln[j][i1] to temp[]
          */
         i1 = 0;
         while (ln[j][i1] != '\0')
         {
           temp[i1] = ln[j][i1];
           i1++;
         }
         temp[i1] = '\0';
       }
    }
    j++;
} /* end while j */
/*
** We need to swap the ith and the kth tuples
** in the array ln[][] provided that k != i.
*/
if ( !(k == 0) )
{
  /*
  ** Swap the ith and the kth tuples in the array ln[][]
  */
  i1 = 0;
  while (ln[k][i1] != '\0')
  {
    temp[i1] = ln[k][i1];
    i1++;
  }
  temp[i1] = '\0';

  i1 = 0;
  while (ln[i][i1] != '\0')
  {
    ln[k][i1] = ln[i][i1];
    i1++;
  }
  ln[k][i1] = '\0';

  i1 = 0;
  while (temp[i1] != '\0')
```

```
      {
        ln[i][ii] = temp[ii];
        ii++;
      }
      ln[i][ii] = '\0';
    }
    i++;
  } /* end while i */

  /*
  ** at this point the tuples are sorted and are ready
  ** to be saved in the file from where they were obtained.
  ** Open the file for write and then write the sorted
  ** array back to it.
  */
  ifileptr = fopen("query.rel", "w");

  i = 0; j = 0;
  while (ln[i][0] != '\0')
  { /* read until end of array */
    ii = 0;
    for (j = 0; j < MAXLINE; j++) { temp[j] = '\0'; }
    while ( ln[i][ii] != '\0')
    {
      temp[ii] = ln[i][ii];
      ii++;
    }
    fputs(temp, ifileptr);
    fputs("\n", ifileptr);
    i++;
  }
  fclose(ifileptr);

}
```

8.4 Makefiles

```
##########################################################################
#
#    MAKEFILE FOR STAND-ALONE UX CODE APPLICATION.
#
#
#      EXECUTABLE       is the name of the executable to be created
#      MAIN             is the .c file containing your main() function
#      INTERFACES       is a list of the interfaces C code files
#      APP_OBJS         is a (possibly empty) list of the object code
#                       files that form the non-interface portion of
#                       your application
#
##########################################################################

EXECUTABLE = QE
MAIN = QE.c
INTERFACES =  UncertainDatabase.c \
QueryEditor.c \
ScrolledText.c

LANGUAGE        = KR-C
APPL_OBJS       =

UX_DIR = /pkg/uimx
```

```
UX_LIBPATH = $(UX_DIR)/lib
X_LIBS = -lXm -lXt -lX11

X_LIBPATH      = -L/pkg/X11/lib
MOTIF_LIBPATH = -L/pkg/X11/lib
X_CFLAGS       = -I/pkg/X11/include
MOTIF_CFLAGS  = -I/pkg/X11/include

# For sun4 K&R cc
KR_CC = cc
KR_CFLAGS = -D_NO_PROTO
KR_LDFLAGS = -Bstatic

# For Centerline K&R cc
# KR_CC          = clcc -traditional
# KR_CFLAGS      =
# KR_LDFLAGS = -Bstatic

# For GNU K&R cc
# KR_CC          = gcc
# KR_CFLAGS      = -traditional -D_NO_PROTO
# KR_LDFLAGS = -Xlinker -Bstatic

# For Centerline ansi cc
# ANSI_CC = clcc
# ANSI_CFLAGS =
# ANSI_LDFLAGS = -Bstatic

# For GNU ansi cc
ANSI_CC = gcc
ANSI_CFLAGS      = -ansi
ANSI_LDFLAGS = -Xlinker -Bstatic

# For Centerline c++
# CPLUS_CC = CC
# CPLUS_CFLAGS =
# CPLUS_LDFLAGS = -Bstatic

# For GNU c++
CPLUS_CC         = gcc
CPLUS_CFLAGS = -xc++
CPLUS_LDFLAGS = -Xlinker -Bstatic

CFLAGS =   -I$(UX_DIR)/include $(X_CFLAGS) \
$(MOTIF_CFLAGS) -DXOPEN_CATALOG -I/usr/xpg2include

XOPEN_LIBS = -lxpg

LIBPATH = $(X_LIBPATH) $(MOTIF_LIBPATH) -L$(UX_LIBPATH) \
 -L/usr/xpg2lib
LIBS = -luimx $(X_LIBS) -lm $(LD_FLAGS) -lc $(XOPEN_LIBS)


OBJS = $(MAIN:.c=.o) $(INTERFACES:.c=.o) $(APPL_OBJS) $(EXTRA_OBJS)

$(EXECUTABLE): $(OBJS)
@echo Linking    $(EXECUTABLE)
$(LD) $(OBJS) $(LIBPATH) $(LIBS) -o $(EXECUTABLE)
@echo "Done"


.SUFFIXES:
.SUFFIXES: .o .c .c
```

```
.c.o:
@echo Compiling $< [$(LANGUAGE)] [UX-CODE]
$(CC) -c $(CFLAGS) $< -o $@
.c.o:
@echo Compiling $< [$(LANGUAGE)] [UX-CODE]
$(CC) -c $(CFLAGS) $< -o $@



CC = \
@`if [ "$(LANGUAGE)" = "C++" ]; then echo $(CPLUS_CC) $(CPLUS_CFLAGS);fi` \
`if [ "$(LANGUAGE)" = "ANSI C" ]; then echo $(ANSI_CC) $(ANSI_CFLAGS); fi`\
`if [ "$(LANGUAGE)" = "KR-C" ]; then echo $(KR_CC) $(KR_CFLAGS); fi`

LD = \
@`if [ "$(LANGUAGE)" = "C++" ]; then echo $(CPLUS_CC);fi` \
`if [ "$(LANGUAGE)" = "ANSI C" ]; then echo $(ANSI_CC); fi`\
`if [ "$(LANGUAGE)" = "KR-C" ]; then echo $(KR_CC); fi`

LD_FLAGS = \
`if [ "$(LANGUAGE)" = "C++" ]; then echo $(CPLUS_LDFLAGS);fi`\
`if [ "$(LANGUAGE)" = "ANSI C" ]; then echo $(ANSI_LDFLAGS); fi`\
`if [ "$(LANGUAGE)" = "KR-C" ]; then echo $(KR_LDFLAGS); fi`
```