SOME THOUGHTS ON PROBABILISTIC DATABASES

Hector Garcia-Molina

Daryl Porter

CS-TR-090-87

April 1987

Some Thoughts on Probabilistic Databases

Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey
08544

ABSTRACT

It is often desirable to represent entities in a database whose properties cannot be deterministically classified. We develop a new data model that includes probabilities or confidences associated with the values of the attributes. Thus we can think of the attributes as random variables with probability distributions dependent on the entity the tuple purportedly describes. We study two sets of issues, one dealing with the proper model for probabilistic data and the other dealing with the choice of operators and language necessary to manipulate such data.

April 28, 1987

Some Thoughts on Probabilistic Databases

Hector Garcia-Molina
Daryl Porter

Department of Computer Science
Princeton University
Princeton, New Jersey
08544

I. Introduction

Traditional database systems represent entities and relationships whose properties are deterministically defined. For example, in an EMPLOYEE relation we can record the fact that employee Smith has a given salary and address, and there is only one address and salary for this employee. This deterministic view has been extended in the so-called *incomplete information* database systems [5,6]. Here each attribute can take on a set of values, indicating that the real value is unknown but is one of these possible values. For instance, employee Smith may have the set {toy, sales} as a value for the DEPARTMENT attribute. This means that Smith either works in the toy or the sales department.

In this paper we extend the incomplete information model to include probabilities or confidences associated with the values of attributes. Thus, we can think of the attributes as random variables with probability distributions dependent on the entity the tuple purportedly describes. For instance, we may want to record in the database that the likelihood that Smith works in the toy department is 70% and that he works in the sales department is only 30%. These likelihood numbers can represent the belief of the person who entered the data, or the result of some statistical experiment conducted by one or many database users.

Our objective in this paper is to study two sets of issues. One set deals with the proper model for probabilistic data. The second set concerns the choice of operators for manipulating such data. The choice of model and operators is not simple, for as the following examples illustrate, there are many different ways to think about probabilistic data. These examples also illustrate the types of applications where the need for probabilistic databases arises.

EXAMPLE 1.1 Discrete Valued Data:

520 observers are asked to make a value judgment on an entity using a non-numeric scale (excellent, very good, good, bad, very bad, horrible). 120 record the outcome as very good, 200 as excellent, 100 as only good, and 100 as bad. Since the sample size is so large, the frequency will make a good approximation for the probability:

$P[\text{Property} = \text{Excellent} \mid \text{Entity} = e_1] \approx 0.39$. Also, a relation presented in

this fashion will clearly communicate more information to a user than a typical relation:

Relation RATING	
ENTITY	JUDGMENT
e_1	120/520 Very Good 200/520 Excellent 100/520 Good 100/520 Bad

EXAMPLE 1.2 Real Valued Data:

A group of N observers of varying skill scattered throughout the world are being asked to monitor the position of many ships. These observers enter sightings into a database. These measurements are used to determine a confidence interval as to the ship's *real* position. Further, based on an observer's track history of sightings, indirect information about, say, observer Smith's reliability can be recorded (e.g. Smith's sightings are more than two standard deviations away from the mean):

Relation SHIPS	
SHIP_NAME	POSITION
Henrietta	$(\mu - \sigma, \mu + \sigma)$

EXAMPLE 1.3 Fuzzy Data

A teacher is asked to "rate" his class on a number of *fuzzy* qualities such as study habits, attitude, and compatibility with fellow students. A relation CLASS is set up with key NAME, and dependent attributes SH (Study Habits), A (Attitude), and C (Compatibility). The domains for each are the same as in EXAMPLE 1.1. The teacher is allowed to decide in a fuzzy set fashion to which set each student belongs. The sets here are the underlying domains. The teacher comments that Susan Anthony usually exhibits excellent work habits except when she is fighting with her boyfriend, which she does about 20% of the time. Her attitude is always good, but she is not compatible with the other girls (who make up 40% of the class). Ms. Anthony's entry in the database looks like the following:

Relation CLASS			
NAME	SH	A	C
Susan Anthony	0.8 Excellent 0.2 Horrible	1.0 Good	0.6 Excellent 0.4 Bad

EXAMPLE 1.4 Relations with Probabilistic Keys

Military intelligence is interested in keeping track of the hardware at a

particular camp. In such a scenario, classes of weaponry such as tanks are prone to incomplete information. Observers speculate the presence of two new classes of tanks at the camp they call alpha and beta based on properties they think these tanks exhibit: The observers feel 70% certain that the camp contains 15 alpha class tanks. The conjecture is that alpha tanks have a new heat masking ability that render them invisible on infra-red scopes when idling. Also, these tanks have tremendously augmented armor plating. Class beta tanks exist with only 40% assurance, and have no heat masking ability. They do have greater armor like alpha, and there are 25 of them. The camp is forced to keep a total complement of tanks with the U.N. This claims that the camp contains 50 bravo type tanks which are known to be lightly armored. In a relation TANKS one data representation might be:

Relation TANKS			
NAME	ARMOR	HEAT_MASKING	COMPLEMENT
0.7 [Alpha	Heavy	Yes	15]
0.4 [Beta	Heavy	No	25]
1.0 [Bravo	Light	No	10]

The paper is organized into 6 additional sections. In section 2 we propose a new model for probabilistic databases. A number of variations and extensions are possible; some of these are briefly discussed in the same section. In Section 3 we explore methods for generating probabilistic relations, introduce a new operator STOCHASTIC that transforms a deterministic relation into a probabilistic one, and provide more motivating examples. In the following section we discuss various basic operators for the proposed probabilistic model, including PROJECT, SELECT, and JOIN. Some additional non-conventional operators are presented in Section 5. These include a COMBINE operator to union two probabilistic relations, and a DISCRETE operator (STOCHASTIC's counterpart) that converts a probabilistic relation into a deterministic one. In Section 6 we discuss the use of particular probability distributions in the relations. In particular, we present notation to describe a uniform distribution and show how our operators can manipulate such distributions. In the last section we conclude by exploring additional extensions, directions for future research, and some suggestions for implementing a probabilistic database system.

II. A New Model

We will depict the conventional relational schema for a relation as $R<K,A_1,...,A_n>$. Each attribute A_i, $i\epsilon\{1,...,n\}$, has underlying domain D_{A_i} and K has underlying domain D_K. The K attribute is an artifice: We are giving the primary key attribute a special name and assuming without loss of generality that it is a simple attribute. Let E be the set of objects being described by relation R. E might be ships at sea, football players, secretaries at Princeton, or students. There is a one to one function $f:E\rightarrow D_K$. For example, if E is football players then D_K might be first, middle, and last names.

Now let us restrict our attention to the instances where all underlying domains are countably infinite or finite. So, attributes can be viewed as discrete random variables. We will leave continuous random variables to another paper except for unjustified allusions and examples where continuous random variables are going to be useful and interesting. Despite EXAMPLE 1.4's indication of the possibility of having probabilistic keys, we will assume that the K field is not probabilistic. This view of uncertainty is simpler to handle but is not overly restrictive. In fact, we can still model probabilistic keys using this simpler view, albeit not as conveniently, as the following example demonstrates.

EXAMPLE 2.1

Recall from EXAMPLE 1.4 that each tuple was assigned a probability of existence (e.g., there was a 0.70 probability that the camp had "alpha" type hardware). This could be represented in our simpler deterministic key model by two relations:

Relation EXISTS	
NAME	Found?
alpha	0.7 alpha 0.3 null
beta	0.4 beta 0.6 null
bravo	1.0 bravo

Relation DATA			
NAME	Armor	Heat_Mask	Complement
alpha	heavy	yes	15
beta	heavy	no	25
bravo	light	no	10

The first relation states whether each hardware type exists at the camp. The second one describes the hardware and is a conventional relation. The two relations can be joined over Found? = Name to obtain a relation similar to

the one given in EXAMPLE 1.4:

Relation EXISTS >Found?=Name< DATA				
Name	Found?	Armor	Heat_Mask	Complement
alpha	0.7 [alpha	heavy	yes	15]
beta	0.4 [beta	heavy	no	25]
bravo	1.0 [bravo	light	no	10]

(Joins will be discussed in more detail in section IV.). In summary, this example illustrates how probabilistic relation keys can be emulated in our simpler model. Hence, it seems reasonable to focus initially on deterministic keys.

The deterministic key assumption means that the entities in E *do* exist and hence the tuples describing them exist as well. Further, the K attribute is the conditional attribute. In short, we are thinking about observers measuring *properties* of real entities (attribute K) and that these properties (attributes $A_1,...,A_n$) are prone to errors or may be qualitative judgments. The attributes ($A_1,...,A_n$) are all dependent on the value of K as to what value they take on.

Assuming that all of the A_i are independent of each other is overly restrictive. For example, consider functional dependencies. If one attribute functionally determines the value of another attribute then it hardly seems sensible to assume that these are independently distributed. Consequently, the model we propose allows users to define groups of these attributes as dependent and thus jointly distributed. So, a general relational schema will be represented by $R<K;G_1;...;G_n>$ where each G_i is a non-empty set of jointly distributed attributes. All G_i must be disjoint. We will forego set notation for the G_i wherever context makes it clear to simplify notation further. The following examples demonstrate the notation.

EXAMPLE 2.2

Traditional Schema: R<K,A,B,C,D,E> with $D_K,D_A,...,D_E$.

Description: A,C,D are inter-dependent attributes while B,E are independent.

Probabilistic Schema: R<K; B; E; A C D> with same underlying domains.

Instance:

Relation R					
K	B	E	A	C	D
k_1	p_{11} b_1 p_{21} b_2	p_{12} e_1 p_{22} e_2 p_{32} e_3	p_{13} [a_1 p_{23} [a_1	c_1 c_2	d_1] d_3]

EXAMPLE 2.3 Ship's Position Schema and Instance

Traditional Schema: POSITION<NAME,TY,TS,CTY,CS,CG>.

Description: TY=Type, TS=TopSpeed, CTY=Country, CS=CurrentSpeed, CG=Cargo.

Functional Dependency: $TY \rightarrow TS, CG$.

Probabilistic Schema: POSITION<NAME; CTY; CS; TY TS CG>.

Instance:

Relation POSITION					
NAME	CTY	CS	TY	TS	CG
Titanic	0.4 USA 0.6 Britain	1.0 0	0.8 [LuxLiner 0.2 [OilTanker	25 35	Passengers] Oil]

It is useful to think of each G_i as a function. It takes as parameters a key and a tuple of values; it returns the probability that the tuple in the relation with the given key takes on the given values. In EXAMPLE 2.3 above, for instance, we can state that:

$$\text{CTY(Titanic,<USA>)} = \text{P[CTY = USA } \mid \text{ NAME = Titanic]} = 0.4$$

$$\text{TY\_TS\_CG(Titanic, <LuxLiner,25,Passengers>)} = \text{P[TY = LuxLiner,TS = 25, CG = Passengers } \mid \text{ NAME = Titanic]} = 0.8$$

This provides a convenient shorthand for the probabilities. If the tuple $<v_1, \ldots, v_j>$ does not appear in the G_i attribute of tuple k, then the function $G_i(k, <v_1, \ldots, v_j>)$ returns the value zero. For example, in relation POSITION,

$$\text{CTY(Titanic, <France>)} = 0$$

All examples so far present tuples that are clearly not in normal form. The form selected is for ease of display and to convey as much information as possible. If the tuple of EXAMPLE 2.1 were normalized, the result would be twelve simple tuples on the traditional attributes with one additional attribute to record the probability. EXAMPLE 2.3 would yield four simple tuples. In such a normal form, K could not be used as the primary key for R, but rather the entire tuple would be necessary.

This normalization issue brings up the following feature. Since we are determining at relation creation time which of the A_i are dependent and which are independent of each other, we can always determine the joint probability distribution by multiplying probabilities of the individual components. We can also determine the absolute probability of one or more of the attributes of a jointly distributed group. This is equivalent to finding the marginal probability distribution for the attributes in question. If a group is split apart in this manner, however, it is not possible or sensible to multiply the individual probabilities and hope to regain the same joint distribution. To illustrate

how joint probabilities can be obtained from independent attributes, consider the following example:

EXAMPLE 2.4 Finding Joint Distributions given the Independent Distributions:

Independent			Joint		
K	A	B	K	A	B
k	0.3 a_1 0.7 a_2	0.4 b_1 0.6 b_2	k	0.12 [a_1 b_1] 0.18 [a_1 b_2] 0.28 [a_2 b_1] 0.42 [a_2 b_2]	

The next example demonstrates the reverse operation:

EXAMPLE 2.5 Finding Marginal Distributions given Joint Distributions From

Relation R_1		
K	A	B
k	0.2 [a_1 b_1] 0.7 [a_2 b_1] 0.1 [a_2 b_2]	

We can conclude that

$$P[A = a_1 \mid K = k] = 0.2$$

$$P[A = a_2 \mid K = k] = 0.7 + 0.1 = 0.8$$

$$P[B = b_1 \mid K = k] = 0.2 + 0.7 = 0.9$$

$$P[B = b_2 \mid K = k] = 0.1$$

This information could be represented as follows, but it would be misleading:

Relation R_2		
K	A	B
k	0.2 a_1 0.8 a_2	0.9 b_1 0.1 b_1

Note that we did not recover the original distributions (given by R_1 at the beginning of this example). (Jumping ahead a bit, what we have done is project R_1 over A, project R_1 over B, and then joined the two partial results to obtain R_2. We lose information in the decomposition, so this is a "lossy join decomposition," analogous to lossy decompositions in relational database theory.)

In all of our examples so far, the probabilities for each attribute group in each tuple add up to one. Clearly, all probability distributions must have this property. However, in some cases it may be convenient to represent only a part of a distribution, possibly because the complete distribution is unknown or is not of interest. For example, consider the following relation, where the domain for attribute A is $\{a_1, a_2, a_3, a_4\}$.

EXAMPLE 2.6

Relation R	
K	A
k_1	0.5 a_1 0.3 a_2
k_2	0.9 a_1 0.1 a_3
k_3	0.5 a_1 0.0 a_2

For tuple k_1, there is a probability of 0.5 that its A attribute is a_1, and a probability of 0.3 that it is a_2. With probability 1 - 0.5 - 0.3 = 0.2 k_1 will take a different value, either a_3 or a_4. We will assume that this *missing probability* is distributed in an unknown way. It could be that the probability that k_1 takes on the value a_3 is 0.2 and a_4 is zero. Or it could be a_3 with probability 0.1 and a_4 with probability 0.1, or any other combination. We will call this the "no assumption assumption" for missing probabilities. (In section VI we explore a uniform distribution that could be used for missing probabilities.)

Notice that if there are missing probabilities in a relation, then the underlying domain must allow it. In our example, if the domain for A were $\{a_1, a_2\}$, then tuple k_1 would be invalid because the missing probabilities could not be assigned to any other values. Also notice that it now makes sense to explicitly store a zero probability value in a relation. For instance, in tuple k_3, it is impossible that its A attribute take on a value of a_2. If this entry were deleted, it would mean that A could take on a_2. As a matter of fact, the probability that k_3 takes on the value a_2 could be as high as 0.5. (If there were no missing probabilities, then it would be equivalent to have an explicit zero probability value or not to represent the value at all.)

The functions G_i we defined earlier (where G_i is an attribute group) retrieve explicitly stored probabilities. For example, for relation R, $A(k_3,<a_1>) = 0.5$, $A(k_3,<a_2>) = 0.0$, $A(k_3,<a_3>) = 0.0$. In some cases we may wish to obtain the maximum probability taking into account missing probabilities. For this we define a new set of functions G_i^m. In our example, $A^m(k_3,<a_1>) = 0.5$, $A^m(k_3,<a_2>) = 0.0$, $A^m(k_3,<a_3>) = 0.5$, $A^m(k_1,<a_4>) = 0.2$.

As we argued above, missing probabilities are useful because they allow a user to manipulate partially known probability distributions. In addition, they make possible some types of relational operations (e.g., some joins cannot be performed without missing probabilities) and allow the definition of some new operators. These issues will be discussed in Sections IV and V.

A final observation concerning the new model's flexibility should be made. When $G_i(k, val) = 1.0$ we have a traditional deterministic attribute entry, and when $G_i^m(k, val) = 1.0$ for all domain values we have a null value. Also, when the explicit probabilities add up to one and are equally divided amongst the entries we have an entry very similar to that of Lipski's model in [5,6]. In other words, the new model encompasses many simpler data models as well as providing a different interpretation to what these models mean.

EXAMPLES 1.2, 1.3, 1.4 depict some of the many extensions to the simple model we have introduced here. Other extensions include lifting the no-assumption assumption about implicit distributions in favor of other enticing interpretations such as the uniform distribution for missing probabilities. In subsequent treatments of this topic we will consider probabilistic keys describing tuples which may or may not exist. Also, when continuous normally distributed random variables are considered, then confidence intervals can be formed to create another type of probabilistic relation. Whenever an underlying domain is numeric then the expected value of the attribute can define yet another probabilistic relation. The relationship to fuzzy sets bears further scrutiny although at present more traditional stochastic models will be considered.

III. Generating Probabilistic Relations

We now turn our attention to the practical concern of where these probabilistic relations might come from. If no examples existed then this paper would be merely a theoretical exercise and thus rather uninteresting. In this section we explore an idea hinted at in earlier examples and summarize a recently published paper which offers a more sophisticated framework out of which more interesting examples might later come to light.

The first and most obvious method is to allow a user to enter the probabilistic information himself. The user is then calling on his own belief heuristic to measure the likelihood of the properties. A more interesting method involves the division of the data and the relations describing it into two classes: (1) Raw Data and (2) Summary Data. In the Raw Data relations many different users (observers) are permitted to record the properties of the same entity. In this way information about an entity is defined as a sample. In the Raw Data tables we see that the primary key will be the K attribute we have been using plus the OBSERVER attribute. Computing the probabilistic information for the Summary relation will only amount to computing the relative frequency of each value in an entity's sample and dividing by the size of the sample.

EXAMPLE 3.1 Belief Heuristics:

In football recruiting there are a host of metrics for hundreds of potential recruits. Many of these properties are clearly deterministic such as 40 speed, bench press, SAT scores or class attendance. In addition, however, there are many properties of interest that might be measured qualitatively by a player's coach or doctor, such as attitude, home life, or potential for weight gain or likelihood to get injured often. For example, growth potential can be estimated fairly accurately by taking X-rays of hips, ankles and wrists to determine how open the growth plates are. Often, football players have been injured enough to have such X-rays available on some subset of these three regions. Another source of uncertainty might be when these X-rays were taken and how far apart. In the final analysis, however, a Doctor usually looks at what information he has at hand and gives his own estimate based on the very imperfect database he carries in his head. A DBMS of the type we have been discussing would allow such qualitative measurements along with an estimate of his belief.

EXAMPLE 3.2 Raw Data Relation:

The scenario of EXAMPLE 3.1 could be modified slightly to produce a Raw Data relation. Each coach from all the high schools of a particular league could be asked to rate local players on a set of qualitative attributes and these could be entered into the database as the Raw Data.

EXAMPLE 3.3 Summary Relation

Domains: $D_A \in \{a_1, a_2, a_3\}$ $D_B \in \{b_1, b_2\}$ $D_C \in \{c_1, c_2\}$

Traditional Relational Schema: Raw Data<K,OBSERVER,A,B,C>

Probabilistic Relational Schema: Sum Data<K; A;B C>

Raw Data					Sum Data			
K	OBSERVER	A	B	C	K	A	B	C
k_1	1	a_1	b_1	c_1	k_1	0.6 a_1	0.2 [b_1	c_1]
k_1	2	a_1	b_1	c_2		0.2 a_2	0.4 [b_1	c_2]
k_1	3	a_1	b_2	c_2		0.2 a_3	0.2 [b_2	c_1]
k_1	4	a_2	b_1	c_2			0.2 [b_2	c_2]
k_1	5	a_3	b_2	c_1				

The Summary relation based on an underlying Raw Data relation suggests a new relational operator called STOCHASTIC. STOCHASTIC could be implemented using the traditional relational model. The STOCHASTIC operator could take as parameters:

[1] A Raw Data Relation and its Schema,
[2] A Composite KEY for the Schema in two parts:
  (a) The K attribute of the Probabilistic Output Relation,
  (b) The rest of the attributes in the Raw Data's KEY,
[3] A Probabilistic Relational Schema.

This syntax would make it easy to generate many different probabilistic relations based on the same data as the following examples illustrate:

EXAMPLE 3.4

Let Raw Data be the relation described in EXAMPLE 3.3. The Sum Data relation would be generated by the following call:

STOCHASTIC(Raw Data<K,OBSERVER,A,B,C>,
<K,OBSERVER>,
<K;A;B C>)

EXAMPLE 3.5

STOCHASTIC(RawData<K,OBSERVER,A,B,C>,
<K,OBSERVER>,
<K;A B; C>) =

Relation $Sum\ Data_2$			
K	A	B	C
k_1	0.4 $[a_1$	$b_1]$	0.6 c_1
	0.2 $[a_1$	$b_2]$	0.4 c_2
	0.2 $[a_2$	$b_1]$	
	0.2 $[a_3$	$b_2]$	

EXAMPLE 3.6

Pseudo-code algorithm for STOCHASTIC using the relational algebra

0:Initialize Sum Data

1: FOR EACH $k \in \pi_K(Raw\ Data)$ DO
BEGIN
2: Sample $:= \sigma_{K=k}(RawData)$

3: FOR EACH G_i , $i \in Sum\ Data\ Schema$, DO
4: FOR EACH $tuple \in \pi_{G_i}(Raw\ Data)$ DO

$$5{:}G_i(k, tuple) := \frac{|\sigma_{G_i=tuple}(Raw\ Data)|}{|Sample|}$$

6: Add k to Sum Data
END

The above statistical approach to generating a probabilistic relation relies heavily on the law of large numbers which says that as the sample size grows larger and larger the relative frequency approaches the probability of the event occurring.

The idea of relational layering has recently been approached in a different sense by Gosh in [2]. In his framework relations can have both column and row attributes which correspond to logical attributes and equivalence classes. In this way Gosh builds up a two dimensional hash table for getting to data very fast by following pointers. The column and row attributes can be broken up over and over into more and more specific classes and ranges of values. The raw data is stored by determining exactly which cell describes the data statistically and following a pointer within the cell to the data page and storing the record at the next convenient location. This form of relation layering has important implications in both statistics and database management. It allows the elegance of relational algebra methods of data retrieval coupled with fast data recovery and it adheres to well established statistical practices of analyzing subsets of data. Extensions of relational query languages that Gosh recommends include sampling with and without replacement. This framework appears quite flexible and seems an ideal method for organizing raw data. Such a scheme would clearly facilitate the methods sketched earlier in this paper for defining probabilistic tables from raw data. Gosh's relational layering also seems pertinent to an area of further research in defining a

probability density function language so probabilities of attributes could be analytically generated. Such functions could be tested thoroughly using the sophisticated statistical analysis offered in Gosh's proposed framework.

STOCHASTIC could be given even more flexibility by allowing a more general set of parameters to specify the probabilistic relation it is to produce. Note that the same conventional Raw Data relation can yield many different probabilistic relations depending on the probabilistic conceptual schema. If this schema is given as a variable parameter to the operator, different combinations could be tested automatically to determine which offers the best summary of the data at hand.

IV. The Conventional Relational Algebra Operators Revisited

We will look at the conventional relational algebra operators of PROJECT, SELECT, and JOIN. We will define these operators in a precise way for probabilistic relations. Recall, however, that probabilistic relations abandon the familiar first normal form of most relational database management systems (DBMS). In fact, each tuple can be thought of as a relation within the relation (r-w-r). This suggests the opportunity to apply the relational algebra operators at two levels. Operators can act at the relation level or at the tuple (r-w-r) level. Performed at different levels these same operators will yield different results. In this section we define these operators at the relation level and in the next section we define them at the r-w-r level.

We begin with conventional PROJECTS, and recalling the probabilistic relational schema $R<K;G_1;...;G_n>$, this definition will be a relatively straightforward task. There are two broad classes of PROJECTS, PROJECTS that include the key field, and PROJECTS that do not. We will only consider PROJECTS that include the key field at this point. Under these restrictions $\pi_X(R)$ *where* $X = \{K, G_{i_1}, ..., G_{i_z}\}$ is simply $R<K;G_{i_1};\cdots;G_{i_z}>$.

Now, let us turn to relation level SELECTS. This is a slightly more interesting case, and will correspond to Lipski's UPPER and LOWER bound SELECTS from [5,6]. SELECTS here will also feature probability range SELECTS, which have no counterpart in any other system. First, as usual, SELECTS will be denoted by $\sigma_P(R)$ where P is a boolean predicate. At this point we need to define the predicate language of P taking into account the probabilistic nature of the tuples. For example, recalling EXAMPLE 1.3, a user should be able to "select all students having a probability greater than 0.5 of having excellent study habits." The answer should be a relation with exactly the same schema, only fewer tuples. Each selected tuple should have an identical counterpart in the original relation. In order to define the Predicate Language we must first introduce a few definitions and assumptions.

In our predicate language we will allow comparisons with the values stored in the tuple, as in conventional relational query languages. To deal with attribute groups, we extend the binary operators so that they operate in a pairwise fashion. Thus, $A\_B >$ <5, "cat"> will select tuples with an A value greater than 5 and a B value greater than "cat." For added flexibility, we add a "*" wildcard that matches any value. For example, $A\_B >$ <5, *> can be used to select tuples that have an A value greater than 5 and any B value.

We also allow comparisons based on probability values. For this we use the G functions, either G_i or G_i^m, defined in Section II. For instance, the predicate A_B (*, <5, "cat">) > 0.5, used in a select, will retrieve tuples that have an A_B value of <5, "cat"> with probability greater than 0.5. Similarly, the predicate A_B (*, <5, *>) ϵ (0.4, 0.7] can be used to retrieve tuples that have an A value of 5, regardless of their B value, with a probability between 0.4 (exclusive) and 0.7 (inclusive).

We now define our predicate language more formally.

Definition. Let **B** be the set of binary operators, **B** = { $<$, $>$, $\leq$, $\geq$, =, }. Let "*" be the wildcard value such that $*\ B\ q$, $q\ B\ *$, and $*\ B\ *$ evaluate to true, where B ϵ **B** and q is any value.

Definition. Let attribute group G_i be $\{A_1, ..., A_k\}$. We define $<G_i>$ to be $\{<g_1, ..., g_k> \mid g_j \epsilon D_{A_j} \bigcup \{ \text{"*"} \}, 1 \leq j \leq k \}$.

Definition. Let $R = <K, G_1, ..., G_n>$ be a probabilistic relation. A term in a predicate for this relation can be of the following form:

(1) $N\ I$, where I is an interval in [0,1] and N is an arithmetic expression involving real numbers and the functions G_i and G_i^m $(1 \leq i \leq n)$. Each function call is of the form $G_i\ (k;\ g)$ (or $G^{m_i}\ (k,\ g)$), where $k \epsilon <K>$, and $g \epsilon <G_i>$.

(2) $N_1\ B\ N_2$, where $B \epsilon$ **B** and N_1, N_2 are arithmetic expressions as in (1) above.

(3) $G_i\ B\ g$, where $B \epsilon$ **B**, $g \epsilon <G_i>$, or

(4) $K\ B\ k$, where $B \epsilon$ **B**, $k \epsilon <K>$.

Definition. A predicate for a select on $R = <K;\ G_1;\ ...;\ G_n>$ is any logical expression containing terms connected with logical operators AND, OR, NOT. We finish our discussion of conventional selects by demonstrating the power and flexibility of the above predicate language in a series of examples below. We will be constructing queries of the form $\sigma_P(R)$ where R has schema R<K; A B C; D E; F>.

EXAMPLE 4.1 Lipski's Lower Bound

QUERY: Select all tuples such that it is CERTAIN that A = a, B = b, and C = c.

$$P = A\_B\_C(*, <a, b, c>) = 1$$

EXAMPLE 4.2.

QUERY: Select all tuples where it is certain that $A\_B\_C = <a_1, b_1, c_1>$ or $A\_B\_C = <a_2, b_2, c_2>$.

$$P = (A\_B\_C(*, <a_1, b_1, c_1>) + A\_B\_C(*, <a_2, b_2, c_2) = 1.$$

Those readers familiar with Lipski's model will recall that this type of lower bound is harder to evaluate in his model. With our model, the query is simple to write and to evaluate. Of course, when one performs operations with probabilities one must be careful that they make sense. For example, if we want all tuples where it is certain that
($A\_B\_C = <a, b, c>$ or $D\_E = <d, e>$, we need the predicate

$$P = (A\_B\_C(*, <a, b, c>) = 1) \text{ OR } (D\_E(*, <d, e>) = 1)$$

and *not* the predicate

$P = A\_B\_C(*, <a, b, c>) + D\_E(*, <d, e>) = 1).$

EXAMPLE 4.3 Probability range query

QUERY: Select all tuples such that D = d,E = e with probability greater than 0.73.

$P = D\_E(*, <d, e>) \epsilon (0.73, 1]$

EXAMPLE 4.4 Lipski's Upper Bound

QUERY: Select all tuples such that we cannot rule out that $B = b$.

$P = A\_B\_C^m(*, <*, b, *>) \epsilon (0, 1]$

Observe that this query will retrieve almost all tuples in the relation with missing probabilities for the A_B_C attribute. (The only way one of these tuples can be excluded is if all possible b values are excluded with explicit zero probabilities.

EXAMPLE 4.5

QUERY: Does a tuple exist with K = k for which we can't rule out the possibility that B = b?

$P = A\_B\_C(k, <*, b, *>) \epsilon (0, 1]$

EXAMPLE 4.6

QUERY: Retrieve tuples that have a $B = b$ value with probability greater than 0.5.

$P = A\_B\_C(*, <*, b, *>) > 0.5$

Note that this query will only get tuples with A_B_C value $<a, b, c>$ occurring with probability greater than 0.5, for some a and c. If the tuple contains value $<a_1, b, c_1>$ with probability 0.4 and $<a_2, b, c_2>$ with probability 0.3, it will *not* be retrieved. To retrieve such a tuple we need to first apply a relation-within-relation project that will be discussed in Section V. (See Example 5.4.)

EXAMPLE 4.7 Value Based Query

QUERY: Select all tuples such that A $>$ 5 and C $>$ 8 and F $<$ 4.

$P = (A\_B\_C > <5, *, 8>)\ AND\ (F < 4)$

EXAMPLE 4.8

QUERY: Select the set of completely specified tuples.

$P = (A\_B\_C(*,*) \in [1,1])\ AND\ (D\_E(*,*) \in [1,1])\ AND\ (F(*,*) \in [1,1])$

EXAMPLE 4.9

QUERY: Select the set of incompletely specified tuples.

$P = (A\_B\_C(*,*) \in [0,1))\ OR\ (D\_E(*,*) \in [0,1))\ OR\ (F(*,*) \in [0,1))$

EXAMPLE 4.10

QUERY: Select all tuples that have NULL values in the traditional sense in the *D_E* attribute.

$P = (\text{ NOT } (\ D\_E(*,*) > 0\ )\ )\ \text{AND } (\text{ NOT } (D\_E^m(*,*) = 0\ )\ )$

Recall that for a null value, there should be no explicitly stored probabilities. The first half of the predicate checks that there are no explicit probabilities greater than zero. Since this part of the predicate cannot distinguish between explicit and implicit zero probabilities, we include the second half. Since the predicate is rather awkward, we may want to define some shorthand notation for this case, for instance $G_i = \text{NULL}$. In this case, the predicate for the query is

$P = (D\_E = \text{NULL})$

Next we turn our attention to JOINs. The next two examples illustrate that at least some types of joins make sense in our probabilistic model.

EXAMPLE 4.11

Let R be ships at sea, and let an observer have recorded the following information: "The ship's name is definitely 'Maria' and I'm 65% sure that she's a frigate, but she may just be a tugboat." If a relation R_2 exists and contains properties about ships such as maximum speed, and it is a conventional deterministic relation it makes perfectly good sense that all properties in the tuple with key 'frigate' would be inherited probabilistically in a JOIN with R. In other words if I'm 65% sure that a ship is a 'frigate' and I know that a frigate's top speed is 32 knots, then it appears sound to say that I'm 65% sure the ship's top speed is 32 knots.

EXAMPLE 4.12 Join R_1,R_2 over $R_1.A = R_2.A$

R_1		R_2		$R_1 >< R_2$		
K	A	A	B	K	A	B
k	0.9 a_1	a_1	0.6 b_1	k	0.54 [a_1	b_1]
	0.1 a_2		0.4 b_2		0.36 [a_1	b_2]
		a_2	1.0 b_3		0.10 [a_2	b_3]

This still looks optimistic since multiplying yields a probability density function without having to normalize. When we look at the probabilistic field of R_1 and R_2 we have information of the following form:
P[A = a_1 ¦ K = k] and P[B = b_1 ¦ A = a_1]. In general we can say our information is of the form P[A¦K] and P[B¦A]. The join table interpreted in this fashion is P[A,B¦K] = P[A¦K] P[B¦A]. Is this justifiable? Yes, if we can assume that B and K are independent then we can use Bayes' rule to rewrite P[A,B¦K] = P[B¦A,K] P[A¦K] = P[B¦A] P[A¦K]. This is implicit in R_2 where the likelihood of all tuples on A and B is given independently of K.

In the two examples we have considered so far, one of the join attributes is a primary key, while the other is a foreign key. Furthermore, the referential integrity constraint holds. (For each foreign key value there exists a primary key value.) The next example illustrates that this last restriction is not necessary, at least if one is willing to have missing probabilities.

EXAMPLE 4.13

R_1		R_2	
K	A	A	B
k	0.7 a_1	a_1	0.6 b_1
	0.2 a_2		0.4 b_2
	0.1 a_3	a_3	1.0 b_3
		a_4	0.5 b_4
			0.5 b_5

The join of R_1 and R_2 over the A attribute is

$R_1 >< R_2$		
K	A	B
k	0.42 [a_1	b_1]
	0.28 [a_1	b_2]
	0.10 [a_3	b_3]

In the result there is 0.2 missing probability corresponding to the a_2 value in R_1. Since this value was not matched in the join, it is not known how the a_2 values are distributed in the resulting relation. If missing probabilities were not

allowed in the model, then this type of join would be invalid.

Joins where neither attribute is a primary key do not seem to have intuitive motivation, so we will not consider them here.

Rather than formally define the JOIN operation we resort to a final illustrative example to avoid the rather tedious notational task of a general definition of a JOIN:

EXAMPLE 4.14 Extended JOIN

Relation R				Relation S			
K	A	B	C	B	D	E	F
k	0.4 $[a_1$	b	c]	b	0.7 $[d_1$	$e_1]$	0.8 f
	0.5 $[a_2$	b	c]		0.3 $[d_2$	$e_2]$	

Relation R $><$ S						
K	A	B	D	E	F	C
k	0.224 $[a_1$	b	d_1	e_1	f	c]
	0.096 $[a_1$	b	d_2	e_2	f	c]
	0.280 $[a_2$	b	d_1	e_1	f	c]
	0.120 $[a_2$	b	d_2	e_2	f	c]

In summary, we must first find relation S's joint distribution and then we can join the two relations by determining all legal combinations, multiplying their probabilities, and placing the combinations into the resulting relation.

V. New Operators

We have just finished exploring the implications of our new data model on the traditional relational algebra operators. We now turn our attention to the possibility of new relational operators. The relational algebra was designed for the relational model of data, and we consider our model to be an expansion of the relational model. It seems likely that new operators will be necessary to exploit its more flexible definition of data. Indeed, we introduced one new operator, STOCHASTIC, in section III of this paper. In addition to STOCHASTIC, probabilistic relations support a host of other new operations such as relation within relation (r-w-r) PROJECTS, PROJECTS without the key, r-w-r SELECTS, a probabilistic union called COMBINE and an expected value operation called DISCRETE. In this section we describe these new operators and offer examples of them in action.

We now consider a r-w-r PROJECTION, that is, A PROJECTION onto a *proper* subset of a G_i set. Such an operation under the no-assumption assumption is fairly easy to discuss and demonstrate. First, however, we need to expand our query language to differentiate between a r-w-r PROJECT and the conventional PROJECT, $\pi_{K,G_i}(R)$, defined in the last section. Let X be a proper subset of a G_i set. We may denote a r-w-r PROJECTION of G_i onto X as $\pi_X(R.G_i)$, $1 \leq i \leq n$. A r-w-r PROJECT will only operate on the G_i in question. All other attribute groups will be included in the tuple unchanged. If X includes all attributes of G_i then the PROJECT will return the relation R unchanged. Now, all that remains is a formal definition of such a PROJECT. Such a PROJECT amounts to replacing the G_i set by the unique group of values for the X attributes of the G_i set along with their marginal probabilities.

First, let G_i be defined on $\{A_1,...,A_k,...,A_n\}$.

Now, without loss of generality let X be defined on $\{A_1,...,A_k\}$.

Formally, $\pi_X(R.G_i) = R<K;\ G_1;...;G_{i-1};\ X\ ;G_{i+1};...;G_n>$

where $(For\ All\ k \in \pi_K(R), x \in D_X)(X(k,x) = \sum_Q G_i(k, <x,g>))$

where $Q = \{<x,g> \mid <x,g> \in D_{G_i}\}$.

The r-w-r PROJECT can be used in conjunction with the relation level PROJECT to produce a conventional looking PROJECT on a proper subset of a G_i set. We conclude this discussion with a few examples. The first two examples will refer to the database instance for Football Players listed below:

Schema: FootballPlayers<NAME, ATT, WT, INT, HLI, WGP, IPR, SRA>
Description:

ATT Attitude ϵ {1,...,10}
WT Weight ϵ {150,...,400}
INT Intelligence ϵ {1,...,10}
HLI Home Life ϵ {1,...,10}
WGP Weight Gain Pot ϵ {1,...,10}
IPR Injury Prone ϵ {1,...,10}
SRA Strength Rating ϵ {1,...,10}

Functional Dependency: HLI $\rightarrow$ ATT; WGP $\rightarrow$ SRA,IPR
Probabilistic Schema: FootballPlayers<NAME; INT; WT; HLI ATT; WGP SRA IPR>
Instance:

Relation Football Players							
NAME	INT	WT	HLI	ATT	WGP	SRA	IPR
Blados, Brian	0.4[6] 0.6[7]	1.0[300 lbs]	0.3[6 0.5[6 0.2[7	7] 6] 7]	0.2[10 0.3[9 0.1[9 0.4[8	10 9 9 9	2] 3] 4] 2]

EXAMPLE 5.1 r-w-r PROJECT onto Weight Gain Potential, Strength Rating (omitting Injury Prone).

$\pi_{WGP\_SRA}(R.WGP\_SRA\_IPR)$						
NAME	INT	WT	HLI	ATT	WGP	SRA
Blados, Brian	0.4[6] 0.6[7]	1.0[300 lbs]	0.3[6 0.5[6 0.2[7	7] 6] 7]	0.2[10 0.4[9 0.4[8	10] 9] 9]

Computing this query amounted to adding the probabilities of the second and third tuples of the WGP_SRA_IPR sub-relation and omitting the extraneous column and suppressing the redundant row.

EXAMPLE 5.2

The following demonstrates how the two kinds of PROJECTS defined can be used together:

Conventional Project (include key NAME) onto Home Life and Attitude, and onto Weight Gain Potential and Strength Rating.

$\pi_{NAME,HLI\_ATT,WGP\_SRA}(\pi_{WGP\_SRA}(R.WGP\_SRA\_IPR))$				
NAME	HLI	ATT	WGP	SRA
Blados, Brian	0.3[6	7]	0.2[10	10]
	0.5[6	6]	0.4[9	9]
	0.2[7	7]	0.4[8	9]

The nested query produced the result of EXAMPLE 5.1, a relation with schema <NAME; INT; WT; HLI ATT; WGP SRA>. The outer query could then operate on this schema in the conventional way since it involved proper G_i sets to produce the final result.

EXAMPLE 5.3

Given $R<K;\ G_1\ ;\ G_2>$ A r-w-r PROJECT onto X Y where X is a subset of G_1 and Y is a subset of G_2. This can be done recursively as follows

$\pi_X(\ (\ \pi_Y(R.G_2)\ ).G_1\ )$

EXAMPLE 5.4 Combining SELECTS with PROJECTS with JOINS

Given R<K; A B C; D E> Select all tuples such that C = c with probability greater than 0.75. Note that the join below is the natural join over the key attribute K.

$(\pi_K(\ \sigma_P(\pi_C(R.A\_B\_C))\ ))\ ><\ R$

where $P = C(*, <c>)\ \epsilon\ (0.75,1]$

The innermost r-w-r PROJECT suppresses the A_B attributes and at the same time determines the marginal probabilities for the C attribute values.

The SELECT then accepts all tuples with a C = c value in the specified range.

The conventional project then determines exactly which tuples (by Key) should be in the result.

Finally, the JOIN with the original relation R gives back the tuples we

want in their original format.

At this juncture, we relax the constraint that the Key Field be included in the PROJECTION. What does such a PROJECT mean? Certainly, as in all regular PROJECTS of this nature, redundancy will occur. The redundant values will be accompanied in a probabilistic relation by the probability that they exist conditional on a now invisible KEY attribute. The following example illustrates the situation.

EXAMPLE 5.5 PROJECTING out the relation KEY

Relation Q		
K	A	B
k_1	0.6 a_1 0.4 a_2	0.8 b_1 0.1 b_2
k_2	0.7 a_1 0.3 a_3	0.5 b_1 0.4 b_2
k_3	0.9 a_2 0.1 a_3	0.8 b_2 0.2 b_3

$\pi_{A,B}(Q)$		
K	A	B
new	1.3/3 a_1 1.3/3 a_2 0.4/3 a_3	1.3/3 b_1 1.3/3 b_2 0.2/3 b_3

This project gives average distributions of values in the relation. Note that the A column gives a *true* average distribution (since for all keys, the total distribution is explicitly recorded), while the B column only gives an approximation. Thus, it could be the case that tuple k_1 had with 0.1 probability a value b_3 for B. In this case, the average reported would be low. In other words, the values in the PROJECT are lower bounds.

As a final remark, nothing new need be added to support non-key PROJECTS. The user need only omit the KEY field from the conventional relation level PROJECT.

SELECTS can be performed at the r-w-r level as well. For instance, within an attribute group a user may want to select those values that have a probability greater than 0.2 of occurring. This operation would not reduce the number of tuples appearing in the relation. Instead, within each tuple, there could possibly be less lines for the selected attribute. This operation reduces the known information on the attribute. In spite of this, the user may want to perform this operation because it displays the data he is interested in more clearly, with less clutter.

To support such SELECTS we must again consider how to augment our query language. Fortunately, we may use the same artifice as in PROJECTS. We will merely specify a r-w-r SELECT as one whose input is not an entire relation, but rather a specified group of that relation. We can then use the exact same predicate language as before but now the action of the operator will be slightly different. (Incidentally, note that r-w-r SELECTS are only valid if missing probabilities are allowed.)

EXAMPLE 5.6 r-w-r SELECT

Recall the Relation Football Players defined above.

QUERY: Make visible WGP_SRA_IPR triples with probability greater than 0.2.

$\sigma_P(Football\_Players.WGP\_SRA\_IPR)$
where $P = WGP\_SRA\_IPR(*, <*, *, *>) \epsilon (0.2,1]$
RESULT:

Relation Football Players							
NAME	INT	WT	HLI	ATT	WGP	SRA	IPR
Blados, Brian	0.4[6] 0.6[7]	1.0[300 lbs]	0.3[6 0.5[6 0.2[7	7] 6] 7]	0.3[9 0.4[8	9 9	3] 2]

The SELECT amounts to suppressing the two possibilities for WGP_SRA_IPR with probabilities of 0.1 and 0.2.

EXAMPLE 5.7 r-w-r SELECTS: Recursive calls and empty relation responses

QUERY: The QUERY from EXAMPLE 5.6 followed by the restriction:
Make visible doubles on HLI_ATT in the range (0.6,0.9).
$P_1 = HLI\_ATT(*, <*, *>) \epsilon (0.6,0.9)$
The query looks like:
$\sigma_{P_1}((\sigma_P(Football\_Players.WGP\_SRA\_IPR)).HLI\_ATT)$
RESULT:

Relation Football Players							
NAME	INT	WT	HLI	ATT	WGP	SRA	IPR
Blados, Brian	0.4[6] 0.6[7]	1.0[300 lbs]			0.3[9 0.4[8	9 9	3] 2]

The probabilistic model also offers the opportunity to create a new relational operator called COMBINE. Such an operator grows out of considering the operations of UNION and INTERSECTION on a probabilistic model. The algorithms to perform UNION and INTERSECTION are as straightforward as in the deterministic case. The resulting relations in both cases seem weaker than necessary in this model. Consider the UNION first. After performing the operation on two relations there may be two tuples describing the same object each with different probability densities. Performing the INTERSECTION, on the other hand, will only include tuples that are described by exactly the same probability densities in two places. It appears that a great deal of information is being lost or unused in both these processes. This suggests an alternative in which an attempt is made to merge all the information at hand.

Tentatively, a COMBINE would work like both a UNION and an INTERSECTION: Certainly, all entities from both relations would be present and if both relations have the same probability distributions then that exact tuple should be in the result. In combining probability densities on the same entity three methods come to mind:

(1) Add the probabilities and normalize the result,

(2) Multiply the probabilities and normalize the result,

(3) A weighted add and normalize schema.

Method (3), which is just a general form of the first method, seems best suited for minimizing information degradation especially after repeated applications. In order to compare the three proposed methods for COMBINE, we will first make an apparent digression. In EXAMPLE 5.8 we will offer some methods for creating Summary Data relations in which it is assumed that we always have the underlying Raw Data available. In EXAMPLE 5.9 we assume the Raw Data was available only long enough to create the Summary Data relations of EXAMPLE 5.8.1. We can then apply our various COMBINE suggestions to the Sum Data at hand and compare our results with the true COMBINATIONS calculated in EXAMPLE 5.8.2.

EXAMPLE 5.8 Comparing methods of summarizing data

Suppose we have three Raw Data relations on the following schema:

Schema: <SKBT,User,QR>
Domain:

SKBT	Skateboard Type	{Jet,Speedster,Coaster}
QR	Quality Rating	{VF,F,S,VS}
User	Users	{Names}

The following Raw Data was collected from 3 stores about Jet Skateboard:

$Data_1$	$Data_2$	$Data_3$
30 users: QR=VF	128 users: QR=VF	76users: QR=VF
40 users: QR=F	72 users: QR=F	100 users: QR=F

30 users: QR=S 200 users: QR=S 24users: QR=S

EXAMPLE 5.8.1 Applying STOCHASTIC to each of the Raw Data Relations

From these Raw Data relations three summary data relations can be induced using

STOCHASTIC($Data_i$<SKBT,User,QR>,<SKBT,User>,<SKBT;QR>)

$1 \leq i \leq 3$:

Sum_{100}		Sum_{400}		Sum_{200}	
SKBT	QR	SKBT	QR	SKBT	QR
Jet	0.3 VF 0.4 F 0.3 S	Jet	0.32 VF 0.18 F 0.50 S	Jet	0.38 VF 0.50 F 0.12 S

EXAMPLE 5.8.2

STOCHASTIC($Data_1 \bigcup Data_2$<SKBT,User,QR>,<SKBT,User>,<SKBT;QR>) and Applying STOCHASTIC to $Data_1 \bigcup Data_2 \bigcup Data_3$ yields the following results...

Sum_{500}		Sum_{700}	
SKBT	QR	SKBT	QR
Jet	0.316 VF 0.224 F 0.460 S	Jet	0.334 VF 0.303 F 0.363 S

EXAMPLE 5.9 Comparing various COMBINE methods

At this point we will assume that we start with the Sum Data relations of EXAMPLE 5.8.1. In EXAMPLES 5.9.1 through 5.9.3 we will apply each of the three proposed COMBINE methods to see how they compare with the real results of EXAMPLE 5.8.2.

EXAMPLE 5.9.1 COMBINE method 1: Add the probabilities and normalize

Sum_{500}		Sum_{700}	
SKBT	QR	SKBT	QR
Jet	0.31 VF 0.29 F 0.4 S	Jet	0.345 VF 0.395 F 0.260 S

This method will suffer greatly when the underlying data is of vastly different sample sizes.

EXAMPLE 5.9.2 COMBINE method 2: Multiply the probabilities and normalize

Sum_{500}		Sum_{700}	
SKBT	QR	SKBT	QR
Jet	0.302 VF 0.226 F 0.472 S	Jet	0.403 VF 0.397 F 0.200 S

EXAMPLE 5.9.3 COMBINE method 3: Weighted add and normalize

For example,

P[QR=VF ¦ SKBT = Jet] = [0.3(100)+0.32(400)]/[100+400] = 0.316.

Sum_{500}		Sum_{700}	
SKBT	QR	SKBT	QR
Jet	0.316 VF 0.224 F 0.460 S	Jet	0.334 VF 0.303 F 0.363 S

The weighted add and combine will always be identical to the STOCHASTIC operator applied to the corresponding underlying samples, but at the cost of carrying extra information in the Probabilistic relation, namely the weights of the samples. The other two methods can suffer different amounts of degradation. If the weights are not managed carefully, however, another kind of information decay can take place. In a weighted add and combine it is necessary to add the tuple weights in the final relation. What happens, for example, if we COMBINE a Probabilistic relation with itself? In the relational model $R \bigcup R = R$ and so $STOCHASTIC(R) = STOCHASTIC(R \bigcup R)$ but COMBINE(R,R) doubles the weight of its resulting tuples. All subsequent applications will distort the information content of the data.

Multiplying the probabilities and normalizing offers some interesting results in extreme cases as the following example demonstrates:

EXAMPLE 5.10 Multiply and Normalize Probabilities

Relation P		Relation Q		COMBINE(P,Q)	
E	A	E	A	E	A
e	0.3 a_1 0.5 a_2 0.2 a_3	e	1.0 a_1	e	0.3/0.3 a_1

This form of COMBINE forces the probabilistic relation to defer to the deterministic relation.

If we had added and normalized, then the result would be different:

COMBINE(P,Q)	
E	A
e	0.65 a_1 0.25 a_2 0.10 a_3

Notice how the likelihood of a_1 is now larger than in P, but not as large as in Q.

EXAMPLE 5.11. Combines with missing probabilities.

When we normalize probabilities, we should take into account missing probabilities, as the following examples illustrate. The first two are with addition of probabilities, the last two with multiplication.

With addition:

P		Q		COMBINE(P,Q)	
E	A	E	A	E	A
e	0.3 a_1	e	1.0 a_1	e	0.65 a_1

P		Q		COMBINE(P,Q)	
E	A	E	A	E	A
e	0.3 a_1	e	0.5 a_1	e	0.40 a_1

With multiplication:

P		Q		COMBINE(P,Q)	
E	A	E	A	E	A
e	0.3 a_1	e	1.0 a_1	e	1.0 a_1

P		Q		COMBINE(P,Q)	
E	A	E	A	E	A
e	0.3 a_1	e	0.5 a_1	e	0.3 a_1

In the last case, we selected a "worst case" scenario. That is, all the missing probabilities could be assigned to the same unknown value. The product probabilities would be 0.3 times 0.5, or 0.15, for a_1, and 0.7 times 0.5, or 0.35, for the unknown value. Normalizing this we obtain the result shown. We could have also selected a "best case" scenario, in which case none of the missing probabilities match up. Then the resulting tuple e would have the value a_1 with probability 1.0. It is not clear

at this point which of these scenarios would be most appropriate.

If we give COMBINE a pair of deterministic relations a probabilistic relation results as output. Also, the STOCHASTIC operator, as its principal function takes a deterministic relation and returns a probabilistic one. We propose an operator called DISCRETE to go the other way. So, DISCRETE would take a probabilistic relation and output a deterministic one. Two ways in which DISCRETE might work are presented here.

Gelenbe discusses in [1] a model for computing the expected value of a query based on a statistical sampling of the data, and such an approach seems appropriate here as well. When attributes have integer values, calculating the expected value in the model we have discussed in this paper is simple. When attributes do not take on integer values, or a group of attributes are jointly distributed the problem becomes less rigidly defined. A tentative DISCRETE procedure could take as parameters a probabilistic relation and its schema, and mappings from the underlying domains of the G_i sets to the natural numbers. Equipped with these mappings, computing the expected value for each G_i is again a well defined task. Note that users could produce many different expected value relations based on the same probabilistic relation by calling DISCRETE with different mappings. Alternatively, DISCRETE could be used to produce a kind of cross product within a tuple. In other words DISCRETE could create a relation consisting of all possible combinations of values defined by a probabilistic relation.

VI. The Uniform Assumption

Up until now when the probabilities for a G_i set do not add up to exactly one we have made no assumption about the missing information. Sometimes, however, particular probability distributions are appropriate to describe the missing information. A user should be able to request that missing information be considered distributed in a particular way if he so desires. In this section we will study the uniform distribution since it is the easiest and the most likely to occur. For example, in traditional systems NULL values are taken to mean "value applicable but unknown." This means that any value in the underlying domain could be the correct value and it is intuitive to assume that they are all equally likely candidates.

For now, we will assume that when probabilities add up to a number strictly less than one, the leftover margin is divided up equally among the other legal values of the domain. Suppose that a user enters the following tuple in relation R<K; A> where $D_A = \{a_1, ..., a_6\}$.

Relation Y	
K	A
k	0.3 a_1 0.2 a_2

Under the uniform assumption for missing probabilities the above tuple is equivalent to the following tuple.

Relation Y	
K	A
k	0.3 a_1 0.2 a_2 0.05 a_3 0.05 a_4 0.05 a_5 0.05 a_6

Obviously, this style of display obscures the information rather than enhances it. A shorthand notation is needed to maintain clarity while still presenting all the information. A simple possibility is to have a special symbol "n, $\Phi<$ *Range* $>\Phi$" that indicates that the values in the Range are uniformly distributed with total probability n.

So, an equivalent representation for the two tuples above might be

Relation Y	
K	A
k	0.3 a_1 0.2 a_2 0.5 $\Phi<a_3..a_6>\Phi$

If the *Range* of a $\Phi<$ *Range* $>\Phi$ symbol includes all values that are not shown, then we can define a special symbol, say "::" for this. Here, "::" will mean all possible values that do not yield a duplicate. We define "::" instead of using the "*" notation defined before to emphasize that redundancy is not allowed here. Thus, the two relations below are identical assuming the same schema and domain as before.

Relation X		Relation Y	
K	A	K	A
k	0.6 a_3 0.2 $\Phi<a_1..a_2>\Phi$ 0.2 $\Phi<a_4..a_6>\Phi$	k	0.6 a_3 0.4 $\Phi<::>\Phi$

This shorthand extends naturally to groups of attributes. For instance, $\Phi<a_1..a_4;\ b_3..b_7>\Phi$ or $\Phi<a_1..a_3;\ ::\ >\Phi$. Operations using this notation are clearly defined. The result of a SELECT, PROJECT, or JOIN where these symbols appear should be identical to the result produced by the expanded relation. For the remainder of this section we will present examples demonstrating query processing under this assumption.

EXAMPLE 6.1

Schema: R<K;A B C D; E> Domains:

$$D_A = \{a_1,...,a_4\} \quad D_B = \{b_1,...,b_4\} \quad D_C = \{c_1,...,c_3\}$$
$$D_D = \{d_1,...,d_5\} \quad D_E = \{e_1,...,e_7\}$$

Instance:

Relation R					
K	A	B	C	D	E
k	0.1[a_1	b_1	c_1	d_1]	0.2 e_1
	0.5[a_1	b_1	c_1	d_2]	0.3 e_2
	0.05[a_2	b_2	c_1	d_1]	0.1 e_3
	0.15[a_2	b_2	c_2	d_1]	0.4 e_4
	0.1[a_2	b_2	c_3	d_5]	
	0.1$\Phi<$::	::	::	::$>\Phi$	

QUERY: $\pi_{K,A\_B}(\ (\pi_{K,A\_B\_C\_D}(R)).A\_B\_C\_D\ )$

(1) Total Explicit Probability = 0.9.
(2) Explicitly stored values of A,B = $\{<a_1, b_1>, <a_2, b_2>\}$.
(3) Marginal Probability of $<a_1, b_1>$ = 0.1 + 0.5 + 0.1(3*5-2)/[4*4*3*5-5].
(4) Marginal Probability of $<a_2, b_2>$ = 0.05 + 0.15 + 0.1(3*5-3)/[4*4*3*5-5].

In (3) and (4) we must determine the total probability associated with the explicitly stored attributes in this tuple. The bulk of this probability can be read from the explicit information. The final term corresponds to the implicit information. To determine this for (3) we not that there are a total of 3*5 = 15 possible mappings with A = a_1 and B = b_1. Of these 2 are explicitly stored with combined probability of 0.6. Each of the remaining 13 have an equal likelihood of existing, namely 0.1/[4*4*3*5-5]. The 0.1 is the total probability to be split equally among the remaining mappings. The denominator depicts the total number of mappings minus the explicitly stored mappings. The solution to the above query looks like:

K	A	B
k	0.6 + 1.3/235 [a_1	b_1]
	0.3 + 1.2/235 [a_2	b_2]
	0.0 + 21/235 Φ<::	::>Φ

The uniform distribution total in the last line can be explained as follows. There are 16 unique A_B pairs. Two of these are explicitly represented, which leaves 14 unique A_B pairs. Each of these 14 pairs can be associated with 15 combinations of C_D pairs, so the total number of combinations is 14*15= 210. Each of these has a probability of 0.1/235 of existing where 235 is again, the total number of combinations not explicitly denoted in the tuple. So, the resulting total probability is 0.1(210)/235 = 21/235.

EXAMPLE 6.2 r-w-r SELECT

Recalling the schema and instance from EXAMPLE 6.1, determine the response to the following query: QUERY: Make visible the tuple information when D = d_3.

The predicate looks like $P = A\_B\_C\_D = <*, *, *, d_3>$

and the query is $\sigma_P(R.A\_B\_C\_D)$ with result as follows

Relation R					
K	A	B	C	D	E
k	0.1*48/235 Φ<::	::	::	d_3>Φ	0.2 e_1 0.3 e_2 0.1 e_3 0.4 e_4

Computing this query amounts to realizing that d_3 appears in none of the explicit probabilities, so all possible combinations of the other three attributes (48) will not yield duplicates. Finally, the 0.1 leftover probability is distributed among all other combinations (235).

EXAMPLE 6.3 JOINs under the uniform assumption

Given the following two relations, M and N, compute the natural JOIN. $D_A = \{a_1, ..., a_3\}$ and $D_B = \{b_1, ..., b_4\}$.

M		N	
K	A	A	B
k	0.8 a_1 0.2 Φ<::>Φ	a_1 a_2	0.6 b_1 0.4 b_2 1.0 Φ<::>Φ

M >< N		
K	A	B
k	0.48 [a_1 0.32 [a_1 0.1 Φ<a_2	b_1] b_2] ::>Φ

The above example violates the second referential integrity rule of the relational model as defined by Date in [10]. The violation is obscured due to the notation, but recall that Φ<::>Φ means that all possible values are applicable. The referential integrity rule states that the foreign key is not a legal value to be used until an entry exists for that value in the relation for which the value is the key. As we discussed in Section IV, when the referential integrity rule is violated, the resulting join has missing probabilities. In

the example, the missing probability is 0.1, corresponding to the probability that tuple k in M takes on a value of a_3.

VII. Conclusion

A new model for information incompleteness of some power with a host of areas for further research has been introduced. An entirely new class of data can be considered. A strong point of the model is its flexibility: Its two extreme cases include the deterministic case and the null value case while a small subset of values with equal probability is exactly the model considered by Lipski. Also, computation involving null values is well defined by the model and so no special actions need be taken. The model supports all of the traditional relational algebra operators as well as a large group of new operators. A notion of relational layering and the potential for summary relations was explored in the probabilistic context and in a more general statistical one. Finally, examples that can take advantage of this general framework do exist and a powerful statistical tool that could work in conjunction with this model is under development elsewhere.

Areas for future research open the door to more potential applications and examples: (1) Continuous attributes usher in all positional data. (2) Defining an analytical probability density function language would be useful in many stock market related issues such as predicting company growth or profit. (3) Functional dependencies (FDs) need to be considered further in such a model. (4) Perhaps the idea of a probabilistic FD would have interesting repercussions. (5) The marriage of Gosh's framework, this probabilistic model, and the statistical power of a system such as "S" needs to be considered. (6) JOINs merit further attention and the relational operator DIVIDEBY deserves formal treatment.

Continuous attributes are the next logical addition to this data model. The defining criteria of the probabilistic model, measuring properties of real objects, indicates that samples of observations will be erroneous measurements. Such samples should be normally distributed, the classic distribution of errors case. Normally distributed random variables offer the opportunity to use confidence intervals defined on these samples to define an interval in which the true value has a certain percentage chance of residing. Such intervals generate interesting possibilities for yet another kind of probabilistic relation based on Raw Data relations. The Confidence Interval relation will convey much more information than a conventional relation on such data. There are many indirect applications and opportunities to apply heuristics based on the users confidence in the data as well. For example, the Raw Data relations could be given a field to display the confidence interval that this observation lies within. Based on this augmented Raw Data relation another probabilistic relation about the observers could be created. This relation could be the output of a call to STOCHASTIC or DISCRETE. In either case the relation could be used to determine the reliability of the observers.

Throughout this paper, the methodology for creating probabilistic relations has been loosely based on sampling theory and statistics. Another method would be to develop a probability density function (pdf) language that the user or database administrator could invoke at relation creation time to analytically describe the probabilistic function that governs the presentation of the data, and the probabilities of attributes or tuples. It seems most likely that users would again be entering data in the

conventional way. The DBMS would then generate probabilities and offer another kind of relation based on the data and the function defined for this relation. For example, the DBA might have decided that a particular attribute was binomially distributed with the underlying domain as the sample space. Then as a user entered a tuple with a particular value in that attribute, the DBMS can determine the likelihood of that value and tag the field appropriately. So, a simple pdf language might have a group of the common distributions as candidates that the DBA can select to govern any attribute. A more general and more interesting method would be to allow the DBA to create pdfs independently. Now the DBMS must make sure that these pdfs are legal. That is, they sum over the sample space (underlying domain) to one.

Another interesting research issue involves FDs. In this paper we assumed that the FDs defined the groups of inter-dependent attributes. So, the schema could be decided based on the FDs. In a more general framework, the users might group attributes without regard to FDs. Does it make sense to allow this generalization? If it is sensible, then we must decide what FDs that cross group boundaries mean. Also, in such a case we may want to introduce a probabilistic FD in the same spirit as the conditional probabilities discussed in this paper. Such information could be kept in another probabilistic relation for each FD. the primary key of the relation would be the input attribute. The dependent attributes would be the attributes it functionally determines. These dependent attributes could be independent of each other or grouped together exactly as discussed in this paper.

The best way, however, to see if such a system does have potential for users is to implement it and give it to people. A simple form of the system equipped with the functionality described above could serve as an ideal starting place. These features include (1) discrete valued attributes, (2) Raw Data Relations, (3) the STOCHASTIC operator, (4) the DISCRETE operator, (5) the COMBINE operator, (6) the ability to define probabilistic relations, (7) The traditional relational algebra and (8) r-w-r relational algebra. The key would be to implement a test version in a relatively short period of time that would run on the greatest number of machines. A first version written in C and running in the UNIX environment could be implemented relatively quickly. This assures a broad academic audience and guarantees maximum portability. If feedback is constructive during the test bed phase then the system could be expanded to meet user needs and to incorporate the features described in this conclusion as areas for future research.

References

Principal References:

[1] Gelenbe, Erol and Hebrail, Georges, *A Probability Model of Uncertainty In Data Bases*, Proceedings International Conference on Data Engineering, Feb. 5-7, 1986, L.A. Calif., IEEE Computer Society Order Number 655, pp. 328-333.

[2] Ghosh, Sakti P. *Statistical Relational Tables for Statistical Database Management*, IEEE Transactions on Software Engineering, Dec. 1986.

[3] Imielisnki, T. *Query Processing in Deductive Databases with Incomplete Information*, Rutgers Technical Report, DCS-TR-177, March 1986.

[4] Imielinski, T. *Automated Deduction in Databases with Incomplete Information*, Rutgers Technical Report, DCS-TR-181, March 1986.

[5] Imielinski, T. and Lipski, W. *Incomplete Information in Relational Databases*, Journal of the ACM, Vol. 31, No. 4, October 1984, pp. 761-791.

[6] Lipski, W. *On Semanctic Issues Connected with Incomplete Information Databases*, ACM Transactions on Databases, Vol. 4, No. 3, Sept 1979, pp. 262-296.

[7] Mendelson, H. and Saharia, A. *Incomplete Information Costs and Database Design*, ACM Transactions on Database Systems, Vol. 11, June 1986, pp. 159-185.

[8] Vassiliou, Y. *Functional Dependencies and Incomplete Information*, Proceedings of the 6th Inernational Conference on Very Large Databases, Montreal, Oct. 1980, ACM, pp. 260-269.

[9] Wong, E. *A Statistical Approach to Incomplete Information in Database Systems*,ACM Transactions on Database Systems, Vol. 7, No. 3, Sept. 1982, pp. 470-488.

Secondary References:

[10]Date, C. *An Introduction to Database Systems Vol. 1*, Addison-Wesley, Fourth Edition, 1986.

[11]Horvitz, E. and Heckerman, D. and Langlotz, C. *A Framework for Comparing Alternative Formalisms for Plausible Reasoning*, Proceedings AAAI-86, Fifth National Conference on Artificial Intelligence, Aug. 11-15 1986, Phila. Pennsylvania, Vol. 1, pp. 210-214.

[12]Meyer, P. *Introductory Probability and Statistical Applications*, Addison-Wesley, Second Edition, 1970.

[13]Lipski, W. *On the Logic of Incomplete Information*, Proceedings of the 6th International Symposium on Mathematical Foundations of Computer Science, Tatranska Lomnica, Czechoslovakia, Sept. 5-9, 1977, T. Gruska Editor, Springer-Verlag, Berlin, pp. 374-381.

[14]Montgomery, C. and Ruspini, E. *The Active Information System: A Data-Driven System for the Analysis of Imprecise Data*, VLDB Proceedings 1981, pp. 376-384.

[15]Reiter, R. *A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values*, Journal of the ACM, Vol. 33, No. 2, April 1986, pp. 349-370.