Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

AN INFERENCE TECHNIQUE FOR DISTRIBUTED QUERY PROCESSING IN A PARTITIONED NETWORK

Wesley W. Chu
Andy Y. Hwang
Qiming Chen
Rei-Chi Lee

February 1990
CSD-900005

Table of Contents

Page

1. INTRODUCTION 2

2. KNOWLEDGE ACQUISITION BY RULE INDUCTION 2

3. THE ARCHITECTURE OF A DDBMS WITH DATA INFERENCE 5

4. DATA INFERENCE 6

4.1 Characteristics of Data Inference 6

4.2 Open Inference and Reasoning 7

4.2.1 Variable Null,Open Range-Object and Valuation 8

4.2.2 Algebra For Open Inference 10

4.2.3 Satisfaction and Toleration 13

5. IMPLEMENTATION 16

5.1 Infernece Engine 16

5.2 A Data Inference Example 18

6. DISCUSSION 19

7.CONCLUSION 20

REFERENCE 22

Appendix A 24

Appendix B 26

Appendix C 28

An Inference Technique for Distributed Query Processing in a Partitioned Network*

Wesley W. Chu, Andy Y. Hwang, Qiming Chen and Rei-Chi Lee

Computer Science Department

University of California, Los Angeles

Abstract

A new knowledge-based approach to query processing during network partitions is proposed. The approach uses available domain and summary knowledge to infer inaccessible data to answer the given query. A rule induction technique is used to extract correlated knowledge between attributes from the database contents. This knowledge is represented as rules for data inference. Correlated knowledge between attributes may be incomplete; as a result, the inferred answer may be exact (complete or incomplete) or approximate. To evaluate inference results under incomplete knowledge, a weaker correctness criterion, called *toleration*, is introduced. New algebraic tools are also developed to support such inference.

The architecture of distributed database system with data inference is presented. A prototype distributed database system that uses the proposed inference technique with correlated attributed knowledge from a ship database has been implemented at UCLA. Our experience reveals that the proposed rule induction technique is capable of obtaining useful correlated knowledge for data inference. As a result, data inference can be viewed as virtual replication and can significantly improve the availability of distributed database during network partitions.

This research is supported by DARPA contract F29601-87-C-0072 and ONR contract N00014-88-K0434

1. INTRODUCTION

To improve reliability and response time in distributed database systems, databases are often partitioned into fragments which are replicated and stored at several sites. Such fragment replication requires additional communication and processing overhead for maintaining consistency among the replicated copies. Further, due to channel and node failures, a network may be partitioned into two or more isolated parts. Since fragments may not be fully replicated at all sites, certain fragments may be inaccessible during network partitions. Most prior work use syntactic information to handle operations during network partitioning which leads to blocking or a partial operable system [GARC87]. However, in many real time applications, the availability of data is of primary importance. It is often not acceptable for a site to suspend processing when it cannot communicate with other sites. Because database attributes are often correlated and contain redundant information (e.g., salary and rank, ship type and cargo), we propose to use data inference technique to *infer inaccessible data from accessible data*. Such knowledge-based approach can greatly increase the availability of distributed database systems.

Knowledge acquisition is a key element for providing successful data inference. In this paper, we shall first introduce a rule induction technique for knowledge acquisition. Next, we present the architecture of a DDBMS with data inference and a technique for inference with incomplete information. Finally, we present the prototype inference system implemented at UCLA based on the proposed knowledge acquisition and inference technique for a naval ship database.

2. KNOWLEDGE ACQUISITION BY RULE INDUCTION

Because database attributes are often correlated and contain redundant information, data inference can be used to infer inaccessible data objects from other *accessible and correlated* data objects. In particular, two different levels of correlated knowledge are used for inference. At the

schema level, correlated knowledge between objects is represented as *inference paths*. Inference paths suggest proper objects and directions that the system should select for data inference. This depends on such criteria as : target objects in the query, object availability status, database schema and degree of correlations between objects. At the instance level, *correlated rules* are used to represent their detailed correlations. In our approach, machine learning techniques are used to induce correlated rules from the database contents. The total space for rule base should be much smaller than its original data since rules are represented as summarized information. The induced rules represent the current state of the database instance which may contain both static and dynamic parts of the database characteristics. Static rules such as integrity constraints do not change while dynamic rules may change as data value updated. However, the induced rules are less volatile than the original data since rules are summarized as range values such as "If '0209' ≤ Shipclass ≤ '0215' then Sonar = 'BQS-4'".

Although different forms of rules may exist in a database, we shall only acquire the pairwise relationship among attributes. Relational operations are to generate correlated rules for pairwise attributes. To induce rules between attributes X and Y, the method retrieves the instances of the (X,Y) pair from the database, and then selects those pairs that X has unique corresponding Y value. The acquired rules can then be summarized in the range form :

Rule: if $x_1 \leq X \leq x_2$ then $Y = y$

or in the set form :

Rule : if X in $\{ x_1, x_2,..,x_n \}$ then $Y = y$

To reduce the size of rule base, we can discard the rules which cover too few pairs of instances. For example, in the ship database, ship name can uniquely determine its ship type. However, the volume of correlated rules between ship name and ship type are too large since each rule covers only one pair instance of ship name and ship type. Storing such correlated rules

will require at least the same size as data replication. Further, the overhead of maintaining the rules will be reduced since such rules are more volatile than others which cover more instance pairs. We shall use a naval ship database to illustrate the knowledge acquisition approach. The ship database was created by the System Development Corporation (SDC, now UNISYS) to provide a fairly realistic naval database based on [JANE81]. For illustration purposes, we use a portion of the ship database which only contains the following relations (the database instances are given in Appendix A):

SHIP = (Id, ShipName, Class)

TYPE = (Type, ss, TypeName)

CLASS = (Class, Type, MaxSpeed)

INSTALL = (Weapon, ShipId)

The result of applying the rule induction algorithm to an instance of this database is given in Appendix A. Depending on the database schema, *intra-object* and *inter-object* knowledge can be identified. Intra-object knowledge defines correlated rules between attributes within one object while inter-object knowledge defines rules between different objects. For instance, correlated rules between ship class and ship type can be categorized as intra-object knowledge since they represent semantic relationship within the same ship object. On the other hand, rules about installed weapons on specific ship types can be categorized as inter-object knowledge since they represent correlation between ship objects and weapon objects.

For intra-object knowledge, the algorithm finds rules about the relationships between *Ship Id* and *Ship Class*; *Ship Class* and *Ship Type*; *Ship Class* and its *maximum speed*; *Ship type* and *surface, or subsurface*. These rules are fairly stable since the classification of ships into different classes and types does not change often. Further, shiptype can also be used to determine whether the ship is a surface or subsurface ship. For inter-object knowledge, we have found rules indicating that the characteristic of surface or subsurface can also be determined by the

ship's class. Further, the acquired correlated rules also indicate that certain weapons are only installed on specific shiptypes or shipclasses.

Due to the limitation of the correlated knowledge between attributes, the rules for inferring the required object may not be complete. For instance, the correlated rules we acquired does not cover all the pair instances between shiptype and the weapons installed on the ships. To infer inaccessible objects using such knowledge, several paths may be selected. Each path may provide partial information and these intermediate results will be extracted or merged to derive the answer. To accommodate such incompleteness, the conventional semantic notions and their algebra in the relational model cannot be adopted directly. Therefore, new algebraic tools are introduced to extract and merge intermediate results to derive inaccessible objects. They will be discussed in the Section 4.

3. THE ARCHITECTURE OF A DDBMS WITH DATA INFERENCE

A distributed database system with inference capability consists of a query parser and analyzer, an information module, and an inference system as shown in Fig. 1. Information module provides allocation and availability information of all the attributes in the system. Inference system consists of a knowledge base and an inference engine. The correlated knowledge between attributes is represented as rules and stored in the knowledge base. During normal operations, queries can be processed by the query processor since all the database fragments are accessible. When network partition occurs, information module and inference system will be invoked if any of the required attributes is inaccessible. Based on the query operations and correlated knowledge between attributes, inference engine modifies the original query to a new one so that all the required data for the query is accessible from the requested site. Depending on physical allocation of the database fragments and domain semantics, the modified query may provide the exact, approximate or summarized answer of the original query. In the following section, we shall present data inference technique used in our inference system.

DDBMS with Data Inference

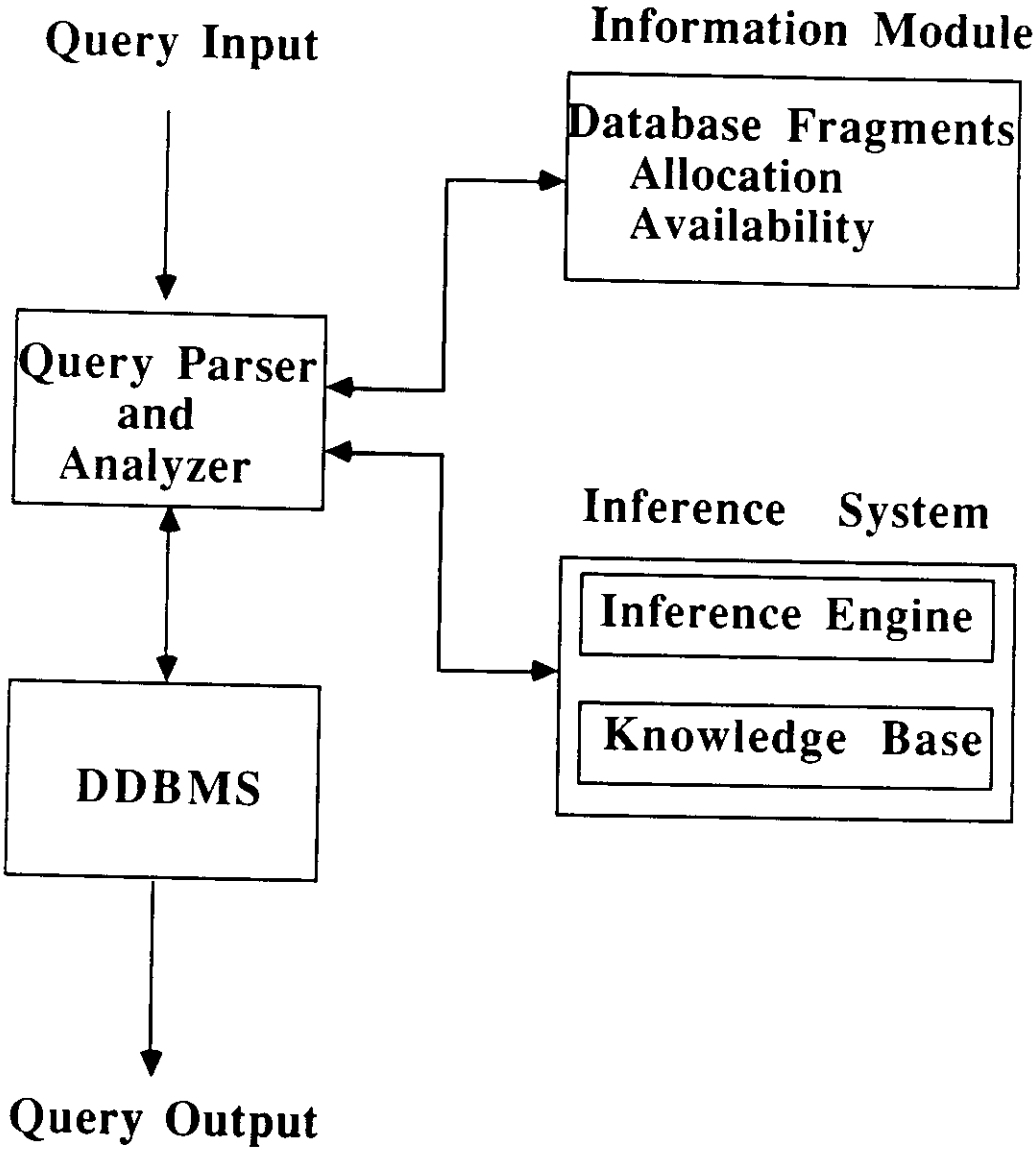

Figure 1

4. DATA INFERENCE

In this section, we discuss the characteristics of our inference approach. Further, new algebraic tools are presented to underlie the data modeling and inference semantics involving incomplete objects.

4.1 Characteristics of Data Inference

We refer to a program, which contains facts (source data) and rules, as a *Data Inference Program (DIP)* and refer to the execution of a DIP as a derivation. Our inference approach is designed to derive inaccessible data objects from other correlated and accessible data objects in the network. However, due to volatile nature of the network and incompleteness of the correlated knowledge, our data inference approach is characterized by the following :

(a) Incomplete Knowledge

Our inference approach takes advantage of the correlated knowledge between inaccessible objects and other accessible objects. To infer the missing data, we start from certain correlated database objects by deriving and merging certain intermediate objects. However, due to the limitation of the correlated knowledge, the rules for a derivation may be incomplete. Since the rules may not be fully provided, both intermediate and final results may also be incomplete. To accommodate incomplete knowledge for data inference, the conventional semantic notions and their algebra in the relational model cannot be adopted directly. Therefore, the algebra of incomplete object reasoning is required to underlie the data modeling and inference semantics involving incomplete objects. We shall refer to this technique as *open data inference.*

(b) Dynamic inference plan

The merging of intermediate results from different derivations may depend on goals, net-

work partition status and response time requirement. Since we cannot predict the availability status for each site, it is not reasonable to predefine a complete set of inference paths for each object. Therefore, execution of Data Inference Program may have to be planned dynamically. The resulting inference plan consists of a set of paths, each representing a derivation from certain available data objects to an intermediate or final data inference result. The data inference is then carried out via those sequential and/or parallel derivations that are scheduled in the above plan.

(c) Toleration

In logic programming, the correctness of a program execution is determined by testing if the resultant interpretation satisfies all the rules and facts specified in that program. However, due to the incompleteness of correlated knowledge, both intermediate and final results may be incomplete. Therefore, the execution of an open data inference may not yield a model containing complete and exact information but rather a *tolerant interpretation* that contains partial information. To accommodate this, we introduce a weaker correctness notion, called *toleration*, for evaluating the data inference results.

4.2 Open Inference and Reasoning

An open data inference consists of one or more statically or dynamically planned derivations. Each derivation is the execution of a Data Inference Program. In the following, the algebra of incomplete object reasoning and two levels of correctness criteria are discussed. Formal descriptions are given in Appendix B. We shall use --> and <-- to represent logical implications, and $\rightarrow$ to represent mappings.

4.2.1 Variable Null, Open Range-Object and Valuation

In order to handle incomplete and dynamically reconstructed database objects, we need to formally distinguish the two types of objects : *closed objects* which do not contain unknown components, and *open objects* which contain unknown components.

Variable Null

The treatment of "incomplete" information in the relational model has been addressed based on the Closed World Assumption (CWA) and Open World Assumption (OWA) [REIT78]. Under CWA, only the facts expressed by the database are true. Thus, a null may be interpreted either as an existing but "unknown" fact [CODD79][BISK81], or as a "non-existing" one [VASS79][ZANI84][CODD86]. Under OWA, besides the facts specified in the database, no further information is available, thus things are left open [RKS 85][GOTT88][OLA 89]. Since our goal is to develop a model theory for open inference, we concentrate on the impact of incomplete information on the inference process and classify nulls as :

- *variable-null* uniquely denoted as ' _ ' may be substituted for by different actual values, or

- an actual "non-exist" value, called *undefined* and denoted as ' | '.

Open-Range Object

To model database objects formally, the existence of some finite sets of values referred to as domains is assumed, and the special value ' | ' (called undefined) is introduced. The product over domains $D_1, D_2,..., D_n$, denoted as $D_1 \times D_2 ... \times D_n$, is the set of all tuples $[x_1, x_2,..., x_n]$ such that $\forall i \in \{1,...,n\}\ x_i \in D_i$. A relation schema, called a *range*, consists of a list of attributes $A_1, A_2,...,A_n$, where each A_i is a subset of a domain. Unique Name Assumption (UNA) on attributes is assumed.

Tuples and relations are generally called range-objects where an attribute value is allowed to be ' | ' or ' _ '. The definition of range-objects is given in Appendix B. In general, a *closed range-object* is free of *variable-nulls*. An *open range-object* contains *variable-nulls*. Thus, a tuple is open if at least one of its attribute values is *variable-null*. A relation is open if it contains at least one open tuple. An open range-object cannot be compared with other range-objects. For example, assuming A,B,C are attribute names, we cannot determine whether [A:1, B:2, C:_] and [A:1, B:2, C:_] are equal since both range-objects are left open, and the *variable-nulls* in each tuple may stand for different actual values. To define relationships and operations on open range-objects, it is necessary to extend the notion of equality to represent syntactically identical objects. Therefore, we adopt the notion of *symbolic equality* , denoted as == , as described in [CODD 86][GZC 87][GOTT 88]. Under this notion, all the *variable-nulls* represented by the same notation ' _ ' are symbolically equal. Further, two tuples are symbolically equal if the values of each attribute are symbolically equal. Two relations are symbolically equal if their tuples are pairwise symbolically equal.

The notion of closed and open objects is related to the notion of closed formula in logic programming. A non-closed formula may contain one or more specific variables such as X, Y, etc. However, in the framework described here, all the unknown components in the open objects are syntactically specified by the same *variable-null* notation.

Valuation

In order to define the notion of satisfaction for open data inference, the concept of *valuation* is introduced. Valuation plays the role of instantiation of variables and *variable-nulls* by actual values under attribute type constraints. The formal definition of valuation is given in Appendix B. For instance, as shown below, the relation "SHIP" is a valuation of another relation "ship" according to the given rule set r1 and r2 :

rule r1 : IF 'B01' ≤ battle_group ≤ 'B02' THEN radar = 'SPS'.

rule r2 : IF 'S120'≤ ship_id ≤ 'S150' THEN battle_group = 'B03'.

ship

ship_id	battle_group	radar
S100	B01	_
S122	_	_

SHIP

ship_id	battle_group	radar
S100	B01	SPS
S122	B03	⊥

4.2.2 Algebra For Open Inference

Data inference consists of mappings between range-objects. Open data inference involves open range-objects. In order to study its interpretation semantics, appropriate algebraic tools are required. This includes special set membership and set containment between range objects, and the extension of these notions for dealing with open range-objects. We first define the *sub-tuple* relationship between tuples. We say tuple t is the *sub-tuple* of another tuple t', denoted as t ≤* t', if any attribute value of t either equals to the same attribute value of t' or is a "non-exist" (i.e. ⊥). For example, we have the following sub-tuple relationships :

[a,b] ≤* [a,b,c]

[a,b] ≤* [a,b,_]

[a, ⊥] ≤*[a,b,_].

The third sub-tuple relationship holds since the first tuple contains the special non-exist value (i.e. ⊥). We then define a special set membership and set containment [CHEN 89a][CHEN 89b], called *s-Membership* and *s-Containment*, denoted as ∈* and ⊆* respectively. The formal definitions for s-Membership and s-Containment are given in Appendix B. Let t be a tuple and R be a relation, we say t ∈* R if t is a sub-tuple of any tuple in R. Further, let R and S be two relations, we say R ⊆* S if every tuple in R is a s-Member of S. In this case, R is also called the *sub-relation* of S. For example, given the following relations "ship1" and "ship2", we have

ship1 $\subseteq^*$ ship2

	ship1		ship2	
ship_id	battle_group	ship_id	battle_group	radar
S100	B01	S100	B01	SPS
S122	B03	S122	B03	I
		S130	B01	_

We call a relation as *s-reduced* if no other tuple in this relation is the sub-tuple of another tuple. For the set of s-reduced relations, it can be proved that the $\subseteq^*$ relationship is reflexive, transitive, and antisymmetric. We can further show that the set of s-reduced relations form a partial order lattice under the $\subseteq^*$ relationship. This property is used to underlie s-union and s-intersection operations which we used in our inference process. The least upper bound (lub) and the greatest lower bound (glb) of two relations under the $\subseteq^*$ relationship are referred to as their s-union ($\cup^*$) and s-intersection ($\cap^*$) respectively as illustrated below.

A	B
a	1
b	2

R_1

A	C
a	3
c	4

R_2

A	B	C
a	1	3
b	2	
c	_	$\bar{4}$

$R_1 \cup^* R_2$

A
a

$R_1 \cap^* R_2$

The above sub-relationships are not restricted to closed range-objects. They also exist between closed range-objects and open range-objects. In general, a closed object R is a sub-range-object of another open object S when R is the sub-range-object of all the valuations of S. Such sub-relationships are definite and thus closed. Relationships associated with open range-objects may be indefinite or open.

Now let us extend the above notions to open range-objects. By using symbolic equality ==, we can develop the notions of *open sub-tuple* $\leq^{\wedge}$, *open s-Membership* $\in^{\wedge}$ and *open s-Containment* $\subseteq^{\wedge}$ (see Appendix B). In short, sub-tuple relationship $t \leq^{*} t'$ is extended to $t \leq^{\wedge} t'$ by allowing t to contain *variable-nulls* (i.e. _). For example :

$$[a,\_] \leq^{\wedge} [a,b]. \quad [a,\_] \leq^{\wedge} [a,\_].$$

Consequently, we can also introduce open s-Membership $\in^{\wedge}$ and open s-Containment $\subseteq^{\wedge}$ similar to the s-Membership and s-Containment shown above. Further, the least upper bound (lub) and the greatest lower bound (glb) of two relations under the $\subseteq^{\wedge}$ relationship are referred to as their open s-union ($\cup^{\wedge}$) and open s-intersection ($\cap^{\wedge}$) respectively. As illustrated in the following example, while R is the open s-union result of relations R_1 and R_2, it also shows $R_1 \subseteq^{\wedge} R$ and $R_2 \subseteq^{\wedge} R$:

A	B
a	1
b	2

R_1

A	C
a	3
_	4

R_2

A	B	C
a	1	3
b	2	
_	_	$\overline{4}$

$R = R_1 \cup^{\wedge} R_2$

It is easy to see that $\leq^{*}$, $\in^{*}$, $\subseteq^{*}$ relationships are special cases of the corresponding open relationships, and the operations $\cup^{*}$ and $\cap^{*}$ are special cases of the corresponding open operations. In fact, when we say that relation R is openly contained in relation S under $\leq^{\wedge}$ or $\subseteq^{\wedge}$ relationship, we mean that

- S contains R, or

- S contains a valuation of R, or

- S contains a partial valuation of R.

The reason $\subseteq^{\wedge}$ is weaker than $\subseteq^{*}$ can be explained as follows : let R and S be two relations; $R \subseteq^{\wedge} S$ implies that there exists valuations from R to R' and from S to S' such that $R' \subseteq^{*} S'$. Conversely, if $R \subseteq^{\wedge} S$ does not hold, no such valuations exist.

The notions introduced in this section provide us with the necessary mathematical tools for handling the interpretation semantics of data inference involving open objects, as shown in the next section.

4.2.3 Satisfaction and Toleration

An open data inference consists of one or more statically or dynamically planned derivations. Each derivation is the execution of a Data Inference Program which contains source data (facts) and rules. The basic actions of a derivation are :

- performing rule based deduction of data,

- updating data to match summary information (knowledge), and

- merging intermediate results.

In logic programming, the model of a program is the interpretation which satisfies all the rules and facts specified in that program. In open data inference, since rules may not be sufficient for inducing all the necessary data, both intermediate and final results may be left open. Since intermediate results can be used as base objects for further inferencing, the base data objects in a DIP may be open. This requires us to study the interpretation semantics in which both source and target data may be incomplete.

In our system, a DIP consists of relations and rules (tuple or relation oriented). The interpretation of a DIP is a set of relations $R_1,..,R_n$ with the following form :

$$I = \{R_1, ..., R_n\}$$

An interpretation is closed if it contains only closed relations; an interpretation is open if it contains at least one open relation. There exist valuation mappings from open interpretations to closed interpretations. In the following discussions, interpretations are handled at the relation level rather than at the tuple level.

The inference results can be evaluated in terms of two levels of correctness criteria : satisfaction and toleration. The notion of satisfaction is usually applied to handle closed objects. We shall now extend the meaning of satisfaction for open data inference involving open range-objects. In general, the satisfaction of a possibly open range-object in a DIP by an interpretation I means I contains an appropriate valuation of that range-object, and all the rules specified in that DIP are satisfied by that valuation. We denote the notion of satisfaction as |=. (For a formal definition, see Appendix B.) For example, as shown below, range-object "SHIP" is correctly derived from another range-object "ship" based on the rule r. For the given DIP containing range-object "ship" and rule "r", we say this DIP is satisfied by the interpretation I = {SHIP}.

rule r : IF 'S120' ≤ ship_id ≤ 'S150' THEN battle_group = 'B03'.

ship		SHIP	
ship_id	battle_group	ship_id	battle_group
S100	B01	S100	B01
S122	_	S122	B03

For a Data Inference Program P and an interpretation I of the above program, we say I is the model of P iff I |= P. Therefore, {SHIP} is a model of the above DIP.

The execution of an open data inference may not yield a model containing complete and exact information but rather a tolerant interpretation containing partial information. To accommodate this, we introduce a weaker correctness notion, called toleration, denoted as |-. Its formal

definition is given in Appendix B. In general, a derivation is tolerated by an interpretation if the known facts and rules are not violated and there exist valuations of the open objects involved in the interpretation that makes it satisfy the derivation. Therefore, let P be a data inference program and I be an interpretation of P. If I satisfies P, then I tolerates P, that is

$$I \models P \rightarrow I \models P$$

Now let us observe the following example, where range-object "$SHIP_1$" is partially valuated from another range object "$ship_1$" based on the rule r :

rule r : IF 'S120' ≤ ship_id ≤ 'S150' THEN battle_group = 'B03'.

$ship_1$

ship_id	battle_group
S100	B01
S122	_
S300	_

$SHIP_1$

ship_id	battle_group
S100	B01
S122	B03
S300	_

Let $I_1 = \{SHIP_1\}$ be an interpretation. For the given DIP containing rule r and range object $ship_1$, we cannot say that I_1 satisfies the DIP since I_1 is still open. However, $SHIP_1$ is indeed a reasonable derivation of $ship_1$ although it is still open. In general, when a possibly range-object in a DIP is tolerated by an interpretation I, then I contains an appropriate *partial* valuation of that range-object and the rules in the DIP are not violated. We cannot say interpretation I satisfies the given DIP since I is still open and may or may not be valuated to satisfy the given DIP. For instance, in the above example, the execution of the given DIP containing rule r and range object $ship_1$ generates a tolerant interpretation $I_1 = \{SHIP_1\}$. We can say range object $ship_1$ is tolerant by the interpretation I_1 since I_1 contains an appropriate *partial* valuation of $ship_1$ and I_1 does not violate rule r.

5. IMPLEMENTATION

In this section, we discuss the implementation of a data inference engine based on the architecture proposed in Section 3. An example based on a ship database is also included to illustrate the inference process.

5.1 Inference Engine

An experimental data inference system has been implemented for a distributed database running on a set of Sun 3/60 workstations interconnected by an Ethernet at UCLA. The system is based on the relational model where all the source and target data objects are relations. The inference actions are extensions of the relational operations which allow us to build the inference engine on top of Sybase, a relational database system. Currently, two types of rules are available :

1) Deductive rules specified in terms of relational operations.

2) Correlated rules which are specified as summarized knowledge. This consists of condition and action parts such as "if 'S120' ≤ ship_id ≤ 'S150' then battle_group = 'B03'".

Consider the following two derivations proceeded by * :

*DERIVATION select ship_id, type from SHIP, CLASS

*DERIVATION CLASS(type) --> INSTALL(weapon)
RULE : IF type in "CG,CV,DD" THEN weapon = 'AAM01'
RULE : IF type in "SSBN,SSG" THEN weapon = 'ASW07'

The first represents a deductive rule, but is expressed by a view definition, where a natural join is implicit due to the Unique Name Assumption of attributes. The second derivation represents the inference path from the type attribute in the CLASS relation to the weapon attri-

bute in the INSTALL relation. It indicates the correlated knowledge between ship types and the weapon installed on each type.

The inference system operates in the following way :

a) When a network partition renders required data objects inaccessible, the inference system develops an inference plan based on the given query, object availability status, database schema and correlated knowledge stored in the rule base.

b) Data inference is then carried out via the inference plan which consists of a set of derivations and the execution sequence of those derivations. Each derivation process represents a derivation from certain available data objects to an intermediate or final data inference result. Three general types of derivations are implemented in the system :

   1) Derive new relation based on certain source relations. It is specified as relational views and implemented through the view generation mechanism.

   2) Valuations of incomplete relations based on summary information and correlated knowledge. The valuation process is implemented through the relation alteration mechanism.

   3) Combining intermediate results in terms of appropriate system operations (viewed as meta-rules). The combination of two relations can be implemented similar to relational outer-join. The usual relation join is not a proper operation for merging intermediate results since certain open tuples may be dropped during the join operation. Those tuples may be valuated through other derivations or combined with the data obtained from other derivations.

c) Select the required data objects from the final result. In the current implementation, since the target objects to be inferred are relations, the inference process is designed to infer as much

of the missing relations as possible. The required attribute information is then selected from the final result.

For instance, consider the following derivation :

*DERIVATION class(type) --> install(weapon)

We first create a temporary relation with all the target values filled by *variable-nulls*. Those *variable-nulls* are then replaced by actual values according to the correlated rules. The results from different derivations can then be extracted or merged to get the final results. Such a merging process either replaces open tuples with closed ones in the same relation according to the correlated rules or it creates a temporary open relation for each derivation, valuates them, and finally combines them via an s-union operation.

5.2 A Data Inference Example

Consider a distributed database that consists of three database fragments : SHIP(sid,sname,class), INSTALL(sid,weapon), CLASS(class,type,tname) which are stored at sites LA, SF and NYC respectively. When the site SF is partitioned, the following query cannot be answered since relation INSTALL is not accessible :

Q1 : " Find the ship names that carry weapon 'AAM01' "

Since the target objects are relations, our inference engine needs to make an inference plan, selecting relevant inference paths for inferring the missing relation INSTALL. We have not yet implemented a dynamic inference plan. To infer the missing relation, the inference engine currently exhaustively searches all the derivations in the knowledge base and selects the relevant derivations. In this example, the following two derivations are used to infer the missing INSTALL relation :

DERIVATION 1 : select ship_id, type from SHIP, CLASS

DERIVATION 2 : CLASS(type) --> INSTALL(weapon)

Derivation 1 represents the first type of derivation where deductive rule is expressed by a view definition. This derivation creates a temporary relation which contains ship_id and type information. Derivation 2 illustrates the second type of derivations, where derivation is performed from [type] to [type,weapon]. This derivation also creates a temporary relation with shiptype and weapon information. While information of shiptype is filled by accessing CLASS object, weapon information is filled based on the provided correlated rules between shiptype and weapon. The above two intermediate results are then combined by the third type of derivation, via open s-union operation, as shown in the following :

R1			∪^	R2		----->	INSTALL_INF		
ship_id	type			type	weapon		ship_id	type	weapon

The result relation, INSTALL_INF, is used to replace the inaccessible INSTALL relation. Query Q1 can be answered by joining SHIP relation with the INSTALL_INF relation. A detailed script of the inference process is given in Appendix C.

6. DISCUSSION

To reduce the overhead in maintaining replicated data, it is desirable to minimize the number of replicas of database fragments yet satisfy the response time and reliability. Since inference can be view as virtual replication, with data inference, the number of replicas may be reduced. Further, since the induced rules can be summarized in range values, the total storage space will be less than full replication. As a result, data inference provides an alternative to data replication for increasing availability. Therefore, with selective database fragment replication

and the use of data inference, the availability of distributed database system can be significantly improved particularly during network partitioning.

It is well known that the allocation strategy of the database fragments has impact on response time and reliability. The optimal allocation for normal operations may be different from that required to provide high availability during network partition. When two database fragments are often referenced together, allocating them at the same site reduces communication cost and thus response time. However, from the data inference point of view, allocating two un-correlated database fragments on the same site and strongly correlated database fragments at different sites provide higher inferential capability and thus increase virtual replication of that database fragment. Since locality and correlation may be dependent, we need to consider both factors in allocating database fragments to different sites in distributed database design.

7. CONCLUSION

A new knowledge-based approach is proposed to improve data base availability for query processing during network partitions. The approach uses available domain and summary knowledge to infer inaccessible data to answer the query. A rule induction algorithm is used to acquire correlated knowledge for data inference application. Since the correlated knowledge between objects may be incomplete, a weaker correctness criterion is introduced to evaluate the inference results. New algebraic tools are developed to support such open inference. A prototype inference system has been implemented on a network of Sun 3/60 workstations at UCLA. Our implementation experience reveals that the proposed rule induction technique is capable of obtaining useful correlated knowledge for data inference. As a result, data inference can significantly improve the availability of the distributed database during network partitions.

Acknowledgements

The authors would like to thank Brian Boesch of DARPA ISTO for his encouragement and support for implementing the prototype system. We also thank G. Popek and T. Page of UCLA for their stimulating discussions.

REFERENCE

[BISK83] Biskup, J, "Foundations of Codd's Relational Maybe operations", ACM Transactions on Database Systems, Vol. 8(4) 1983 pp. 608 - 636.

[CHEN89a] Chen, Qiming "A High Order Logic Programming Framework for Complex Objects Reasoning", International Computer Software & Applications Conference (COMPSAC 89), 1989, USA.

[CHEN89b] Chen, Qiming & Chu, Wesley "A High Order Logic Programming Language (HILOG) for NON-1NF Deductive Databases", Proc. of 1st International Conference on Deductive and Object-Oriented Databases, 1989, Japan.

[CODD79] Codd, E. F. "Extending the Database Relational Model to Capture More Meaning", ACM Transactions on Database Systems, Vol. 4(4) 1979 pp. 397 - 434.

[CODD86] Codd, E. F. "Missing Information (Applicable and Inapplicable) in Relational Databases, SIGMOD RECORD, Vol. 15, no. 4 December 1986.

[GARC87] Garcia-Molina, H. and Abbott, R. K. ''Reliable Distributed Database Management'', *Proc. of the IEEE*, May 1987, pp. 601-620.

[GOTT88] Gottlob, Georg and Zicari, Roberto "Closed World Databases Opened Through Null Values", Proc. of 14th VLDB Conference 1988, pp. 50 - 61.

[GZC87] Gueting, R. H. Zicari, R. Choy, D. M. "An Algebra for Structured Office Documents", IBM Almaden Research Report RJ 5559 (56648), San Jose, CA 1987.

[JANE81] *Jane's Fighting Ships*, Jane's Publishing Co., 1981.

[KING81] King, J. J., "QUIST: A system for semantic query optimization in relational databases," In *proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9-11). IEEE, New York, pp. 510-517.

[OLA 89] Ola, Adegbemiga and Ozsoyoglu, Gultekin, "A Family of Incomplete Relational Database Models", Proc. of 15th VLDB pp. 23 - 31.

[REIT84] Reiter, R. "Towards a Logical Reconstruction of Relational Database Theory", in On Conceptual Modeling, pp 191-234, Springer- Verlag Ed ., 1984.

[RKS85] Roth, M. A. Korth, H. F. Silberschatz, "Null Values in Non 1NF Relational Databases", Report TR-85-32, University of Texas at Austin, July 1985.

[VASS79] Vassiliou, Y. "Null Values in Database Management : A Denotational Semantics Approach", ACM-SIGMOD 1979, pp. 162 - 169.

[ZANI84] Zaniolo, C. "Database Relations with Null Values", Journal of Computer and System Science, Vol 28, No. 1, 1984 pp. 142 - 166.

Appendix A

The Ship Database and Its Induced Rules:

Relation SHIP

id	name	class
S101	Wisconsin	C02
S102	Dale	C03
S103	America	C11
S104	Barry	C11
S105	Texas	C12
S106	John_Hancock	D02
S107	Peterson	D02
S108	Nicholson	D04
S109	John_Rodgers	D12
S110	Paul	D14
S111	Donald_Berry	D15
S112	Clark	D15
S113	Thomas_Hart	D12
S114	Iowa	D12
S115	John_Hall	S02
S116	Seahorse	S03
S117	Rams	SN02
S118	Dallas	SN03
S119	Delta_I	SN04
S120	Delta_II	SN11
S121	Batfish	SN12
S122	Bluefish	S15
S123	Atlanta	S15
S124	LaJolla	S16
S125	Skate	S18

Relation CLASS

class	type	maxspeed
C02	CG	32
C03	CG	32
C11	CV	30
C12	CV	30
D02	DD	34
D04	DD	34
D12	DDG	34
D14	DDG	28
D15	DDG	28
S02	SS	20
S03	SS	20
SN02	SSN	24
SN03	SSN	30
SN04	SSN	30
SN11	SSBN	24
SN12	SSBN	24
S15	SSG	26
S16	SSG	22
S18	SSG	22

Relation INSTALL

weapon	sid
SAM01	S106
SAM01	S107
SAM02	S107
SAM03	S107
SAM03	S108
SAM03	S109
SAM03	S110
SAM03	S111
SAM03	S112
SAM03	S113
SAM03	S114
AAM01	S101
AAM01	S102
AAM01	S103
AAM01	S104
AAM01	S105
ASW02	S115
ASW02	S116
ASW03	S116
ASW03	S117
ASW03	S118
ASW03	S119
ASW07	S120
ASW07	S121
ASW07	S122
ASW07	S123
ASW07	S124
ASW07	S125

Relation TYPE

type	ss	name
CG	surface	guided missile carrier
CV	surface	aircraft carrier
DD	surface	destroyer
DDG	surface	guided missile destroyer
SS	subsurface	patrol submarine
SSN	subsurface	nuclear submarine
SSBN	subsurface	ballistic nuclear missile submarine
SSG	subsurface	guided missile submarine

Intra-Object Rules						
Entity	**IF**					**THEN**
SHIP (Id --> Class)	S103	≤	Id	≤	S104	Class = C11
SHIP (Id --> Class)	S106	≤	Id	≤	S107	Class = D02
SHIP (Id --> Class)	S111	≤	Id	≤	S112	Class = D15
SHIP (Id --> Class)	S113	≤	Id	≤	S114	Class = D12
SHIP (Id --> Class)	S122	≤	Id	≤	S123	Class = S15
TYPE (Type --> ss)	CG	≤	Type	≤	DDG	ss = surface
TYPE (Type --> ss)	SS	≤	Type	≤	SSG	ss = subsurface
CLASS (Class --> Type)	C02	≤	Class	≤	C03	Type = CG
CLASS (Class --> Type)	C11	≤	Class	≤	C12	Type = CV
CLASS (Class --> Type)	D02	≤	Class	≤	D04	Type = DD
CLASS (Class --> Type)	D12	≤	Class	≤	D15	Type = DDG
CLASS (Class --> Type)	S02	≤	Class	≤	S03	Type = SS
CLASS (Class --> Type)	SN02	≤	Class	≤	SN04	Type = SSN
CLASS (Class --> Type)	SN11	≤	Class	≤	SN12	Type = SSBN
CLASS (Class --> Type)	S15	≤	Class	≤	S18	Type = SSG
CLASS (Class --> Maxspeed)	C02	≤	Class	≤	C03	Maxspeed = 32
CLASS (Class --> Maxspeed)	C11	≤	Class	≤	C12	Maxspeed = 30
CLASS (Class --> Maxspeed)	D02	≤	Class	≤	D04	Maxspeed = 34
CLASS (Class --> Maxspeed)	D14	≤	Class	≤	D15	Maxspeed = 28
CLASS (Class --> Maxspeed)	S02	≤	Class	≤	S03	Maxspeed = 20
CLASS (Class --> Maxspeed)	SN03	≤	Class	≤	SN04	Maxspeed = 30
CLASS (Class --> Maxspeed)	SN11	≤	Class	≤	SN12	Maxspeed = 24
CLASS (Class --> Maxspeed)	S16	≤	Class	≤	S18	Maxspeed = 22

Inter-Object Rules						
Relationship	**IF**					**THEN**
SHIP(class) --> TYPE(ss)	C02	≤	class	≤	D15	ss = surface
SHIP(class) --> TYPE(ss)	S02	≤	class	≤	SN12	ss = subsurface
SHIP(class) --> CLASS(type)	C02	≤	class	≤	C03	type = CG
SHIP(class) --> CLASS(type)	C11	≤	class	≤	C12	type = CV
SHIP(class) --> CLASS(type)	D02	≤	class	≤	D04	type = DD
SHIP(class) --> CLASS(type)	D12	≤	class	≤	D15	type = DDG
SHIP(class) --> CLASS(type)	S02	≤	class	≤	S03	type = SS
SHIP(class) --> CLASS(type)	SN02	≤	class	≤	SN04	type = SSN
SHIP(class) --> CLASS(type)	SN11	≤	class	≤	SN12	type = SSBN
SHIP(class) --> CLASS(type)	S15	≤	class	≤	S18	type = SSG
CLASS(class) --> TYPE(ss)	C02	≤	class	≤	D15	ss = surface
CLASS(class) --> TYPE(ss)	S02	≤	class	≤	SN12	ss = subsurface
CLASS(type) --> INSTALL(weapon)	CG	≤	type	≤	CV	weapon = AAM01
CLASS(type) --> INSTALL(weapon)	DDG	≤	type	≤	DDG	weapon = SAM03
CLASS(type) --> INSTALL(weapon)	SSN	≤	type	≤	SSN	weapon = ASW03
CLASS(type) --> INSTALL(weapon)	SSBN	≤	type	≤	SSG	weapon = ASW07
CLASS(class) --> INSTALL(weapon)	C02	≤	class	≤	C12	weapon = AAM01
CLASS(class) --> INSTALL(weapon)	D04	≤	class	≤	D15	weapon = SAM03
CLASS(class) --> INSTALL(weapon)	S02	≤	class	≤	S02	weapon = ASW02
CLASS(class) --> INSTALL(weapon)	S15	≤	class	≤	S18	weapon = ASW07
CLASS(class) --> INSTALL(weapon)	SN02	≤	class	≤	SN04	weapon = ASW03
CLASS(class) --> INSTALL(weapon)	SN11	≤	class	≤	SN12	weapon = ASW07

Appendix B

Formal Definitions and Algebraic Operations for Open Inference

Range-Objects

For a range $R = A_1 \times A_2 \times ... \times A_n$,

a) $\forall i \in \{1,...,n\}\ t_i \in A_i \cup \{\ |\ \} \cup \{\_\} \rightarrow t = [A_1{:}t_1, ..., A_n{:}t_n] \in R$ is called a tuple of R and can be abbreviated as $[t_1, ..., t_n]$. The attribute value of t on A_i can be denoted as $t.A_i$.

b) $\forall i \in \{1,...,n\}\ t_i \in R \rightarrow \{t_1, ..., t_n\} \subseteq R$ is called a relation.

A closed range-object is free of variable-nulls. An open range-object contains variable-nulls.

Valuation

The set Ω of valuation mappings from the set of objects to the set of closed objects are defined as follows :

(a) For a constant value on attribute A, $a \in A, a \rightarrow a \in \Omega$.

(b) For a variable x on attribute A, $(\forall a \in A)\ x \rightarrow a \in \Omega$.

(c) For a null-variable _ on attribute A, $(\forall a \in A \cup \{\ |\ \})\ \_ \rightarrow a \in \Omega$.

(d) For a tuple $t = [A_1{:}t_1, ..., A_n{:}t_n]$, $t' = [A_1{:}t_1', ..., A_n{:}t_n']$,

$\forall i \in \{1,...,n\}\ t_i \rightarrow t_i' \in \Omega$ --> $t \rightarrow t' \in \Omega$.

(e) For a relation $R = \{t_1, ..., t_n\}$, $\forall i \in \{1,...,n\}\ \exists t \in R'\ (t_i \rightarrow t \in W)$ --> $R \rightarrow R' \in \Omega$.

Sub-tuple relationship $\leq^*$

Let t, t' be tuples with attribute list S and S', t is the sub-tuple of t', denoted as $t \leq^* t'$, is defined as :

$t \leq^* t'$ iff $S \subseteq S' \wedge \forall X \in S\ (t.X = | \vee t.X = t'.X)$

s-Membership $\in^*$ and s-Containment $\subseteq^*$

Let t be a tuple and R be a relation $t \in^* R$ iff $(\exists t' \in R)\ t \leq^* t'$.

Let R and S be two relations $R \subseteq^* S$ iff $(\forall t \in R)\ t \in^* S$. R is called the sub-relation of S.

Open sub-tuple

Let t, t' be tuples with attribute sets S and S' respectively, t is the sub-tuple of t', denoted as $t \leq\hat{} t'$, is defined as :

$$t \leq\hat{} t' \text{ iff } S \subseteq S' \wedge \forall X \in S\ (t.X == \_ \wedge t'.X \neq \bot \vee t.X == t'.X \vee t.X = \bot)$$

Open s-Membership $\in\hat{}$ and s-Containment $\subseteq\hat{}$

Let t be a tuple and R be a relation, $t \in\hat{} R$ iff $(\exists t' \in R)\ t \leq\hat{} t'$.

Let R and S be two relations, $R \subseteq\hat{} S$ iff $(\forall t \in R)\ t \in\hat{} S$.

Satisfaction

Let I be an interpretation. The notion of satisfaction, denoted as $\models$, is defined as

a) For a tuple t, $I \models t$ iff $(\exists R \in I)\ t \rightarrow t' \in \Omega \wedge t' \in^* R$.

For a relation r, $I \models r$ iff $(\exists R \in I)\ r \rightarrow r' \in \Omega \wedge r' \subseteq^* R$.

b) For a rule $h \leftarrow b_1,...,b_n$, $I \models (h \leftarrow b_1,...,b_n)$
iff for a substitution θ_I, $I \models b_1\theta_I, ..., I \models b_n\theta_I$ implies $I \models h\theta_I$.

c) For a data inference program P, $I \models P$ iff $\forall p \in P\ (I \models p)$.

Toleration

Let I be an interpretation. The notion of toleration, denoted as $\models\!\!\!\sim$, is defined as

a) For a tuple t, $I \models\!\!\!\sim t$ iff $(\exists R \in I)\ t \in\hat{} R$.

For a relation r, $I \models\!\!\!\sim r$ iff $(\exists R \in I)\ r \subseteq\hat{} R$.

b) For a rule $h \leftarrow b_1,...,b_n$,

$I \models\!\!\!\sim (h \leftarrow b_1,...,b_n)$ iff $I \models (h \leftarrow b_1,...,b_n)$.

c) For a data inference program P, $I \models\!\!\!\sim P$ iff $\forall p \in P\ (I \models\!\!\!\sim p)$.

Appendix C

UCLA FAULT TOLERANT DISTRIBUTED DATABASE PROJECT

Experimental Inference Engine

Version 0.0

March 1990

```
 *** This Database is Distributed at 3 Sites in the Network :
     site 1 : LA
     site 2 : NYC
     site 3 : SF

>> action 1 :  select sname from ship,install where install.weapon = "AAM01"

---------------
sname
---------------
Wisconsin
Dale
America
Barry
Texas
---------------

>> action 2 : turnoff SF
 ** Site SF is DOWN !
>> action 3 :  select sname from ship,install where install.weapon = "AAM01"
 ** Relation install not available since network partition!
 ** Site SF is not accessable.
 ** Try (1) Automatic, or (2) Interactive Inference (1/2/n) : 1
 ** Intermediate Result from Database Access : select sid,type from ship,class :

----------------
sid     type
----------------
S101    CG
S102    CG
S103    CV
S104    CV
S105    CV
S106    DD
S107    DD
```

```
S108    DD
S109    DDG
S110    DDG
S111    DDG
S112    DDG
S113    DDG
S114    DDG
S115    SS
S116    SS
S117    SSN
S118    SSN
S119    SSN
S120    SSBN
S121    SSBN
S122    SSG
S123    SSG
S124    SSG
S125    SSG
---------------
```

** Intermediate Result from Inference Path : class(type) --> install(weapon) :

```
-------------------------
sid     type    weapon
-------------------------
S101    CG      AAM01
S102    CG      AAM01
S103    CV      AAM01
S104    CV      AAM01
S105    CV      AAM01
S106    DD        -
S107    DD        -
S108    DD        -
S109    DDG     SAM03
S110    DDG     SAM03
S111    DDG     SAM03
S112    DDG     SAM03
S113    DDG     SAM03
S114    DDG     SAM03
S115    SS        -
S116    SS        -
S117    SSN     ASW03
S118    SSN     ASW03
S119    SSN     ASW03
S120    SSBN    ASW07
S121    SSBN    ASW07
S122    SSG     ASW07
S123    SSG     ASW07
S124    SSG     ASW07
S125    SSG     ASW07
-------------------------
```

** Inferred Relation install_inf :

```
-------------------------
sid     type    weapon
-------------------------
S101    CG      AAM01
S102    CG      AAM01
S103    CV      AAM01
S104    CV      AAM01
S105    CV      AAM01
S106    DD        -
S107    DD        -
```

```
S108    DD        -
S109    DDG       SAM03
S110    DDG       SAM03
S111    DDG       SAM03
S112    DDG       SAM03
S113    DDG       SAM03
S114    DDG       SAM03
S115    SS        -
S116    SS        -
S117    SSN       ASW03
S118    SSN       ASW03
S119    SSN       ASW03
S120    SSBN      ASW07
S121    SSBN      ASW07
S122    SSG       ASW07
S123    SSG       ASW07
S124    SSG       ASW07
S125    SSG       ASW07
---------------------------
```

** Modified Query : select ship.sname from ship, install_inf where
install_inf.weapon = "AAM01"

```
---------------
sname
---------------
Wisconsin
Dale
America
Barry
Texas
---------------
```