EMBEDDING EXPERT KNOWLEDGE AND HYPOTHETICAL DATA BASES INTO A DATA BASE SYSTEM

by

M. Stonebraker and K. Keller

Memorandum No. UCB/ERL M80/15

14 April 1980

EMBEDDING EXPERT KNOWLEDGE AND HYPOTHETICAL DATA BASES INTO A DATA BASE SYSTEM

by

M. Stonebraker and K. Keller

Memorandum No. UCB/ERL M80/15

14 April 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

This paper is concerned with adding knowledge to a data base management system and suggests two appropriate mechanisms, namely hypothetical data bases (HDB's) and experts. Herein we indicate the need for HDB's and define the extensions that are needed to a data base system to support HDB's.

In addition, we suggest that the notion of "experts" is an appropriate way to add semantic knowledge to a data base system. Unlike most other proposals which extend an underlying data model to capture more meaning, our proposal does not require extensions to the schema. Moreover, the DBMS does not even have to know how an expert functions. In this paper we define an expert and indicate how it would be added to one existing data base system.

1. Introduction

There has been considerable interest recently in adding semantics to a DBMS so that it becomes "smarter". The general approach of all investigators with whose work we are familiar is to extend some existing data model with more semantic constructs. In this way one enriches the class of possible schemas by providing mechanisms which are "global" i.e. can apply to any application domain. Proposed constructs include the notions of entities, properties and relationships [CHEN76], roles [HAMM78], aggregation and generalization [SMIT77a, SMIT77b], convoys [HAMM78], and temporal ordering [CODD79]. We view the recent work of Codd [CODD79] as an excellent example of the "schema extension" approach.

However, there appear to semantic constructs which are not handled well by the above sorts of mechanisms. We now discuss three of them.

1) containment

Often data base objects are inside other data base objects. For example, Berkeley is contained in California, people are often inside rooms, parts are inside warehouses or trucks, etc. It might be argued that both aggregation and convoys deal with this situation. For example, the cities in California as an aggregate have the property of containment within the state. Moreover, they form a convoy, called the California cities convoy.

We view containment as a different notion because it need not apply at all times. For example persons are sometimes in rooms, sometimes in airplanes, and sometimes at bus stops and not contained in any data base object. Hence, containment is a dynamic construct and many different properties can apply depending on what the containment vessel is. Also, the convoy notion seems to model groups. For example, there can be two museum tours, i.e. two convoys, that are distinguishable objects. However, they can both be contained in the same room.

2) distance

Many data base objects have a physical location and consequently it makes sense to have a notion of the distance between them.

3) time

Although [CODD79] suggests the notion of a temporal ordering that may be required between data base objects, there is much more which can be exploited about this concept. For example many data base objects (events) have a starting time, a finishing time, a time duration, the property that they must be carried out between 8 and 5, the property that they must be done tomorrow, etc.

These three examples are notions which are handled poorly (or not at all) by the semantic extensions indicated above. Rather than extend one data model with these constructs, we propose instead the construct of "experts" which allow such notions to be easily added to a data base system. One key feature of experts is that a data base system does not have to understand how an expert works or even what sort of semantic knowledge is embedded in an expert.

In Section 3 we define an expert and show how one is embedded into an existing data base system. Then, in Section 4 we suggest that minor modifications are required to properly capture the notion of a time expert.

One of the application areas where expert-augmented data base systems are clearly desirable are those suited to artificial intelligence oriented front-end programs. A good example of such an area is the "Navy ships" application around which the LADDER [HEND78] system is constructed. In such application areas we also see the need for what we term "hypothetical data bases" (HDB's).

A HDB is obtained from a real data base (RDB) by making some sort of alternate assumption about the current state of the data. The purpose is to explore alternative scenarios, test new application programs, run simulations, produce test data, etc. We give example situations to illustrate HDB's in the context of the Navy ships data base.

Planning Applications

The goal is to hypothetically move the fifth fleet to the Sea of Japan so that an analyst can explore logistic problems associated with resupplying the fleet. Here, one requires a HDB to be constructed from the RDB differing only in the position of the fifth fleet. The HDB is to be maintained until the analyst has completed his work. This scenario also applies to the creation of "what if" test data for simulation programs.

Debugging

A programmer has a new application program which he wants to test on a "live" data base. Rather than risk "trashing" the real data base, he can use a hypothetical data base for his purposes. In this case the HDB may be identical to the RDB, or the programmer may want to explore alternative test cases. This example is suggested in [SEVR76].

The above examples indicate contexts in which a user would want to construct and maintain a HDB. In Section 2 of this paper we suggest the extensions which are needed in a data base system to support HDB's.

The data base system which we choose to extend with the notions of HDB's and experts is the INGRES [STON76] system. However, the results are easily applicable to any data base system.

2. Data Base Support for HDB's

The current INGRES data base system supports the notion of real data bases which may be created and destroyed, each of which contains an arbitrary collection of real relations. Although it is possible to support the notion of hypothetical relations in a real data base, we feel it is more appropriate to support complete hypothetical data bases. This will free the user from iteratively having to specify the hypothetical relations of interest.

Hence, the INGRES command language must be extended to allow the following command.

CREATEHDB HDB_name FROM real_data_base_name

This command will create a HDB which initially will be identical to the real data base. A user can then modify relations in his HDB to any desired state using QUEL [HELD75] commands. We now specify the effect which QUEL commands have on such a HDB.

2.1 Processing Commands Against HDB's

On the first update to any relation in an HDB a differential file will be created for that relation. This differential file (DF) looks very similar to those of [SEVR76] and contains tuples with the same format as tuples in the original relation except for the addition of a new field which is a tuple identifier (TID) for a tuple in the real relation. Basically, DF indicates how the hypothetical relation differs from the real relation.

An APPEND command in QUEL will ultimately add a collection of zero or more tuples to a relation. An APPEND to a relation in a hypothetical data base will have the effect of adding tuples to the DF which have the property that their TID field is null. Table 1 shows the effect of adding Baker to the hypothetical EMPLOYEE relation.

A REPLACE command alters field values in zero or more tuples in a relation. A REPLACE to a relation in a hypothetical data base causes an insert to the DF for that relation of a new tuple for each updated tuple with the property that a combination of old and new field values are present and the TID field has the tuple identifier for the updated tuple. Table 1 also shows Brown receiving a hypothetical raise.

A DELETE command deletes zero or more tuples in a relation. A DELETE to a relation in a hypothetical data base causes an insert to the DF for each tuple deleted. The inserted tuple has the TID of the deleted tuple and null values for all data fields. Table 1 shows the effect of deleting Jones from the hypothetical EMPLOYEE relation.

EMPLOYEE

NAME	SALARY	DEPT
Brown	20	shoe
Smith	15	toy
Jones	25	shoe

DF

NAME	SALARY	DEPT	TID
Baker	30	shoe	-
Brown	25	shoe	TID(Brown)
-	-	-	TID(Jones)

A Hypothetical Relation
Table 1

All updates are, in fact, implemented by doing a RETRIEVE first to isolate the changes to be made followed by lower level modifications. The above paragraphs have indicated the modifications that are appropriate to relations in HDB's. In all cases the real relation is not modified. We now turn to the effect which a RETRIEVE must have in a HDB.

In [SEVR76] an algorithm is presented that supports RETRIEVES to a read-only main file augmented by a read-write DF. Basically, the suggestion requires that the request be for exactly one record which is specified by a unique key. Hence, one looks first in the DF for the record. Only if the request fails does one have to pay a second access to the main file. Moreover, a hashed bit map (called a Bloom filter) in main memory is proposed that can be used to guarantee that the requested record is not in the DF. In this case the first access can be avoided.

There are two problems with this approach:

a) QUEL allows a collection of records to be retrieved via one RETRIEVE command

b) There is no way to tell the INGRES system that a field must be unique. In other words, a request for Stonebraker's record may result in two records being returned, and there is no way to alert the system that this event is impossible.

It is evident the tactics proposed in [SEVR76] only work for unique key retrievals. Consequently, in other environments a RETRIEVE must always be run against both the real relation, R, and the DF. Let TID(DF) be the TID's of the qualifying tuples in DF, TID(R) be the TID's of qualifying tuples in the real relation, and TID(total) be the collection of all TID's in DF.

The TID's of actually qualifying tuples for a RETRIEVE, Q, are:

TID(Q) = TID(DF) union [TID(R)-TID(total)]

These tuples must be retrieved from both the real relation and from DF. Appropriate action can be then taken for this collection.

In summary, one can process a RETRIEVE by:

a) run the RETRIEVE against the DF to find TID(DF)

b) run the RETRIEVE against the real relation to find TID(R)

c) for each tuple returned from b) use a Bloom filter as in [SEVR76] to guarantee that it is not in TID(total). Any tuple with this property can be added to the result of a)

d) for those tuples which are not guaranteed to be absent from TID(total) in step c), perform an auxiliary RETRIEVE to find the collection actually absent from TID(total) and add those to the result of a).

Several comments are appropriate about the performance of this algorithm.

1) In general it will be at least twice as slow as a RETRIEVE against a real relation. This is because the query must be done against two relations. Even though one (DF) may be small this fact will not always speed processing.

2) It will pay to have the DF keyed on the same field(s) as the main file. Obviously, access patterns will be identical for both relations. Moreover, it will clearly pay to have a secondary index on the TID field in DF, since this will speed the lookup in step d) above.

3) In multivariable queries INGRES currently can choose the relation for which to tuple substitute [WONG76]. Also, in the current INGRES query processing tactics, any one variable clauses in a query will result in a temporary relation that has no associated DF. Hence, the above processing need not be done when accessing such a temporary. Consequently, when processing a two variable query against relations, one of which has no DF, INGRES should choose, if possible, to do tuple substitution on the other relation. In this case INGRES can iterate over all tuples in DF and then scan all tuples in the real relation. All it need do is inspect the secondary index for DF for each tuple in the real relation that it uses, discarding the ones in the secondary index. This amounts to a merge of the secondary index and the real relation and is very fast.

2.2 Updating Rules for HDB's

We now turn to updating rules for HDB's and present examples designed to indicate that sometimes one wants the hypothetical environment to be updated when updates occur in the real environment. This should be contrasted with the notion of views [STON75, CHAM75] where one is interested in reflecting updates from unreal objects, namely views, into updates to real relations.

Suppose a user has constructed a HDB with the Enterprise in the Sea of Japan. However, in the real data base the Enterprise is scuttled in San Diego harbor. Should the Enterprise be deleted from the HDB? Alternately, the real Enterprise is in San Diego harbor and 100 new seamen report for duty. Should these insertions be reflected in the HDB? Lastly, suppose a HDB is constructed in which the Enterprise is twice as fuel efficient as currently. Here, the HDB does not alter the current state of the data base, only the way in which updates to the fuel supply are handled. Clearly, the fuel reserve in the HDB and the real data base quickly diverge for the Enterprise. Consequently, how should one reflect the real Enterprise being refueled?

These examples all indicate that real updates should optionally be reflected into the hypothetical environment. On a relation by relation basis, we plan to allow updates to be reflected or not reflected. Hence, the update rules for updating real data bases must be extended as follows:

For a real update to be NOT REFLECTED and the operation is a:

DELETE

In this case one must perform the delete to the real relation and do an insert into the DF for each tuple deleted. If the appropriate tuple already exists in DF, no DF update is needed.

REPLACE

In this case one must perform the update to the real relation and do an insert into the DF for each tuple modified. This insert must put the old values into DF. Also, if the tuple already exists in DF, it must be updated with old values

APPEND

In this case one must perform the append to the real relation and do an insert of a null valued tuple with the appropriate TID into DF for each tuple appended.

For a real update to be REFLECTED and the operation is a:

DELETE

Perform the deletion operation to the real relation and then delete any tuple in DF that corresponds to a deleted tuple.

REPLACE

One must perform the update to the real relation and then inspect DF. Any tuple in DF that corresponds to an updated tuple in the real relation will have appropriate fields set to the modified values. If one becomes equal to the tuple in the main file, it will be deleted.

APPEND

One must perform the indicated append to the real relation.

In summary, to support HDB's we require utilities to create and destroy HDB's and a syntax such as

UPDATES TO hypothetical_relation_name ARE {visible, invisible}

to indicate whether to reflect updates. In addition, we need to alter the INGRES search engine to perform the algorithms indicated in the previous two subsections.

3. The Notion of Experts

We introduce the notion of experts by indicating some of the functions which a geography expert should be able to do. The user in a previous example wished the Enterprise in the Sea of Japan. It is entirely possible that he does not care exactly where in the Sea of Japan the ship is located. For example, he might only be concerned with refueling it in this remote location. As such, he might then ask how long it would take for a tanker in San Diego to reach the Enterprise. Clearly, the answer only very minorly depends on the exact location of the Enterprise. Hence, the user is interested in a context where an exact location is irrelevant.

It should be noted that the position of the Enterprise is not null-valued because the Sea of Japan is at least a coarse specification. Moreover, it is not fuzzy in the sense of Zadeh because a user could, in fact, specify an exact position; he

simply chooses not to. This is different than a fuzzy concept whose value can never be known with certainty. Rather the data is imprecise because it represents a level of detail inappropriate to the application at hand.

Moreover, when presented with a query inquiring if the Enterprise is in Tokyo Harbor, a DBMS augmented by a geographic expert can only answer "I don't know". It is possible that the answer is "yes" because Tokyo harbor is indeed in the Sea of Japan; however, the Enterprise may also be elsewhere. In general, the answer to any query directed to an expert augmented system is an answer qualified by "yes" and a second answer qualified by "maybe".

In addition, it must be possible to move the Enterprise a certain distance from its current position in the Sea of Japan. Consequently, a geographic expert must be able to handle arithmetic.

We now treat each of the following topics in turn:

1) creating data bases involving experts

2) the functions provided by an expert and their integration into INGRES

3) communicating knowledge to an expert

3.1 Creating Expert Oriented Data Bases

Each field of any relation in INGRES will be allowed to be supported by an expert. The syntax of the CREATE command will be extended to allow the following:

CREATE rel_name_1 {(field_name = {format, expert_name})}

This syntax is identical to the one currently supported except for the possibility that the format clause is replaced by an expert_name. The effect of this command is to create the indicated relation and indicate in the system catalogs that the appropriate field name is associated with the indicated expert. We will use the following relation to illustrate the use of experts:

CREATE SHIP_POSITION(name=C20, position = geography_expert)

Consequently, INGRES will support any number of experts, each associated with certain fields in various relations. We turn now to the definition of an expert.

3.2 The Definition of Experts

An expert is a procedure (in the language "C" [RITC75]) which has been duly registered with the data base system and can process the following four calls.

1) Whenever the parser recognizes a term of the form

expert_field operator value

it calls the expert associated with that field to provide an internal representation for that value. For example, a user could place the Enterprise in the Sea of Japan with the following replace statement:

```
RANGE OF S IS SHIP_POSITION
REPLACE S(position = "Sea of Japan")
  WHERE S.name = "Enterprise"
```

The first call allows the expert to provide a code for the geographic entity "Sea of Japan" which is stored by the data base system in the position field. Of course, the expert must return an internal value which is the appropriate length defined during the registration process.

2) The expert must accept a qualification of the form:

value_1 comparison_operator value_2

and return a value from the set

(true, maybe, false)

For example, to find the ships in Tokyo Harbor one would query the data base as follows:

RETRIEVE (S.name) WHERE S.position = "Tokyo_Harbor"

A type 1 call would convert "Tokyo_Harbor" to internal form (i.e. to value_2). Then, INGRES would retrieve the record for the Enterprise (among other records). The geographic expert would resolve whether the code for the "Sea of Japan" matched the code for "Tokyo Harbor".

Notice that in general ALL position codes must be evaluated by the expert to answer this query because INGRES has no idea what positions actually match the code for "Tokyo Harbor". Later in this section we discuss mechanisms to overcome this source of overhead.

3) The expert must be able to do computations of the form:

expert_field arithmetic_operator constant

For example, the Enterprise might be in the Sea of Japan and its position might be updated to be 10 miles north of wherever it is now. This would require an update of the form:

```
REPLACE S(position = S.position + 10N)
  WHERE S.name = "Enterprise"
```

The 10N would be converted to internal form by a type 1 call. Then the expert would be required to return a code for the arithmetic sum of the code for 10N and the one for the Sea of Japan. This code would be stored as the position of the Enterprise.

4) Before any expert-oriented field is returned to the user or application program, it must be passed to the expert for a possible conversion to external format. For example, if the user wishes to know the position of the Enterprise, he would query as follows:

RETRIEVE (S.position) WHERE S.name = "Enterprise"

Obviously, the code for the "Sea of Japan" should never be returned to someone outside the data base system; rather the external representation is returned by calling the expert.

Moreover, note that the expert can return more than one value if he wishes. For example, the Enterprise is likely to be in the Sea of Japan as a result of the 10N update above. However, it is possible that it is in open ocean to the north. Hence, the expert can return both possibilities in response to a type 4 call.

Lastly, the expert can return "I don't know" as a possible conversion. This could result from the following sequence of operations. The user wishes to know the distance of the enterprise from Tokyo Harbor and inquires as follows:

```
RETRIEVE (desired_distance = S.position - "Tokyo_Harbor")
    WHERE S.name = "Enterprise"
```

First "Tokyo Harbor" would be converted to internal form and then a type 3 call would be required to compute a code for:

code_of_enterprise_position - code_of_Tokyo_Harbor

Clearly, the answer is somewhere between 0 (if the Enterprise is in the harbor) and the maximum distance between Tokyo Harbor and any point in the Sea of Japan. Given this uncertainty, the expert can only compute a code representing "I don't know". Finally, a type 4 call converts this "I don't know" code to an external representation which is returned to the user.

The last issue associated with the above notion of experts is what to do if a qualification evaluates to "maybe" as a result of a type 2 call. For true and false there are obvious courses of action; for maybe the course of action must be the following.

Any tuple for which "maybe" was returned by the expert must be kept for further processing in the normal course of INGRES algorithms as if the value were "true". However, it must be flagged as a "maybe". Ultimately a relation is returned to the user or calling program; some tuples in which may have the "maybe" flag set.

For example, to find the ships in the Sea of Japan one would ask

RETRIEVE (S.name) WHERE S.position = "Sea of Japan"

The answer to this query is a collection of ships with certainty and a collection of ships with maybe.

We now turn to avoiding exhaustive searches when type 2 calls are required. Obviously, an expert must be registered with the data base system, since the DBMS must call it at run time (and link in the expert's code) and know how wide the code values are.

The registration process includes a specification for the answers to the following

questions.

1) Does code_1 < code_2 imply that value_1 < value_2

i.e. does the coding process preserve order.

2) Does code_1 != code_2 imply that value_1 != value_2

These two pieces of information will often allow INGRES to avoid an exhaustive search when a qualification involves an expert field. In addition, the following also appears useful.

3) Does code_1 .AND. code_2 = FALSE imply that value_1 != value_2

Here, one could code values in such a way that Sea of Japan was 1000 and Tokyo Harbor was 1XXX. This would allow the data base system to search for matches efficiently when property 2) above is not true.

Clearly, it must be possible for a user to communicate information to the expert. We now turn to how this might be accomplished.

3.3 Communication With an Expert

Knowledge will be communicated to an expert as a byproduct of certain updates. For example, the coding expert suggested for MacAims [GOLD70] was a mechanism obeying our expert paradigm. Their expert assigned an internal representation for any external string. This internal representation was supported by a binary tree data structure and had the property that if string_1 was less than string_2 then code_1 was less than code_2. This is exactly property 1 which would be communicated in the registration process noted above.

Such an expert, when presented with a new external value will simply assign a new code and insert the correspondence into whatever data structure it is maintaining. However, for some experts this mechanism is not sufficient.

For example, the geographic expert is totally ignorant of new concepts. Presented with the query "Find the names of the ships in the Bering Sea" e.g.:

```
RETRIEVE (S.name) WHERE S.position = "Bering Sea"
```

the expert can clearly assign a code to the "Bering Sea"; however, he has no way of knowing what OTHER codes match the code for "Bering Sea". Hence, he must be provided with this information.

We propose that experts receive information through the data base system from end users or programs. In this way, the information must be provided in a very stylized way that is under the control of the data base system. Consequently, humans are discouraged from "hand crafting" knowledge directly into the internal form accepted by an expert. As such it may be possible to write a "meta expert" which can be adapted to multiple application areas by inserting different knowledge.

The data base system is prepared to accept the following commands:

1) COMMUNICATE WITH expert_name "name_1" operator "name_2"

2) COMMUNICATE WITH expert_name "name_1" operator "string"

3) COMMUNICATE WITH expert_name "name_1" operator "name_2" operator "string"

4) COMMUNICATE WITH expert_name "string"

The legal operators for syntax 1) are expected to be:

a) comparison operators (=, !=, <, <=, >, >=)

For example COMMUNICATE WITH geography_expert "Taiwan" = "Formosa"

b) part_of

For example, COMMUNICATE WITH geography_expert "midwest" part_of "United States"

c) IN

For example, COMMUNICATE WITH fleet_expert "Enterprise" IN "fifth_fleet"

Syntax 2) is intended to allow definition of terms to an expert. For example,

COMMUNICATE WITH geography_expert "Mississippi River" = "definition_of_Mississippi_River_in_expert_terms"

Moreover, syntax 3) is intended to allow definition of relative terms, e.g.

COMMUNICATE WITH time_expert "yesterday" = "today" - "24 hours"

Assuming "today" and "24 hours" have already been defined, this allows the definition of yesterday. We expect to implement syntax 3) allowing any arithmetic operator.

The last syntax allows passing an arbitrary string to an expert. In the next section we indicate some uses for this general construct.

4. The Time Expert

It is clear that a time expert can obey the paradigm of the preceding section. One need only specify that some field in a relation be the time field controlled by the time expert. Presumably, this field stores the time from the system clock or some more complex representation. In this way a row in such a relation is essentially timestamped with a value. We now give an example to show why such an expert is not sufficient.

Suppose we have a relation of the form:

CREATE SHIP_POSITION (name = C20, position = geography_expert, time =

time_expert)

SHIP_POSITION contains information on the position of ships and the time at which that sighting took place.. Suppose a program is periodically inspecting a sensor and issuing the update:

```
RANGE OF S IS SHIP_POSITION
REPLACE S(position = "some_value", time = "current_time")
  WHERE S.name = "Enterprise"
```

In this case the SHIP_POSITION relation will contain only the most recent sighting for any given ship. Suppose a user now issues the query:

```
RETRIEVE (S.position) WHERE S.name = "Enterprise" and
                    S.time = "yesterday"
```

Obviously, the data base will respond that no tuples match the qualification and that the answer to the query is "I don't know" (or more accurately I forgot!)

In order to avoid failing to answer such queries, we propose an extension to our paradigm appropriate for the time expert.

Whenever, a REPLACE operation is indicated for a relation containing a field managed by the time expert, it is automatically turned by INGRES into an APPEND. For example, the command

```
REPLACE S(position = "some_value", time = "current_time")
   WHERE S.name = "Enterprise"
```

would be altered to

```
APPEND TO SHIP_POSITION (position = "some_value",
time = "current_time", other_fields = S.other_fields)
    WHERE S.name = "Enterprise"
```

An APPEND, of course remains and APPEND. However, a DELETE causes a problem. For example, to sink the Enterprise one would:

```
DELETE S WHERE S.name = "Enterprise"
```

If this command is processed as stated, then the whole sightings history disappears and one would not be able to find out where the Enterprise was yesterday. The only rational course of action appears to be to disallow DELETES. Consequently, to sink the Enterprise one would have to do a REPLACE on a status field which had allowed values {operational, sunk}.

Note that relations managed by the time expert have the property that they increase in size with each update. Obviously, this can't last long. Hence, we propose that users be able to communicate how forgetful the expert should be using the COMMUNICATE command as in the following examples.

```
COMMUNICATE WITH time_expert "size of relation_name < N-tuples"
```

COMMUNICATE WITH time_expert "remember last N updates"

COMMUNICATE WITH time_expert "remember last N time units"

These are all cases where more complex information must be communicated with an expert than allowed in syntax 1)-3) above.

We now briefly indicate the relationship between a time expert and an audit trail. Most audit trails contain old values and new values for each update to the data base [GRAY78] and are typically spooled onto an alternate volume and then to tape.

It is clear that the time expert maintains a complete audit trail, albeit in a slightly different form. If the time expert spools tuples to tape instead of throwing them away when they become too old, then it should be able to provide an audit trail as a side effect. This idea has been suggested previously, and we plan to explore the performance consequences of unifying the two concepts.

5. Conclusions

We have proposed two mechanisms to allow a DBMS to become "smarter", namely HDB's and experts. If possible we plan to implement both notions. Moreover, we expect to write a geography expert according to our proposed paradigm to test its robustness.

Acknowledgement

This research was supported by the Naval Electronics Systems Command under Contract N00039-76-c-0022 and by the Army Research Office under Grant DAAG29-76-6-0245. Ken Keller is a Fannie and John Hertz Foundation Fellow.

References

[CHAM75]
Chamberlin, D. et. al., "Views, Authorization and Locking in a Relational Data Base management System," Proc. 1975 National Computer Conference, Anaheim, Ca., June 1975.

[CHEN76]
Chen, P., "The Entity-Relationship Model: Toward a Unified View of Data," ACM TODS 1, 1, March 1976.

[CODD79]
Codd, E., "Extending the Relational Model to Capture More Meaning," to appear in ACM TODS.

[GOLD70]
Goldstein, R. and Strnad, A., "The MacAIMS Data management System," Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access, Houston, Texas, November 1970.

[GRAY78]
Gray, J., "Notes on Operating Systems," IBM Research, San Jose, Ca., Report RJ 3120, October 1978.

[HAMM78]
Hammer, M. and McLeod, D., "The Semantic Data Model: A Modelling

Mechanism for database Application," Proc. 1978 ACM-SIGMOD Conference on Management of Data, Toronto, Ontario, August 1978.

[HELD75]
Held, G. et. al., "INGRES: A Relational Data Base System," Proc. 1975 National Computer Conference, Anaheim Ca., June 1975.

[HEND78]
Hendrix, G. G. et. al., "Developing a Natural Language Interface to Complex Data," TODS, June 1978.

[RITC75]
Ritchie, D. and Thompson, K., "The UNIX Time-sharing System," CACM, June 1975.

[SEVR76]
Severance, D. and Lohman, G., "Differential Files: Their Application to the Maintainance of Large Databases," TODS, June 1976.

[SMIT77a]
Smith, J. and Smith, D., "Database Abstractions: Aggregation," CACM 20, 6, June 1977.

[SMIT77b]
Smith, J. and Smith, D., "Database Abstractions: Aggregation and Generalization," ACM TODS 2, 2, September 1977.

[STON75]
Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," Proc. 1975 ACM-SIGMOD Conference on Management of Data, San Jose, Ca., June 1975.

[STON76]
Stonebraker, M. et. al., "The Design and Implementation of INGRES," TODS 2, 3, September 1976.

[WONG76]
Wong, E. and Youseffi, K., "Decomposition - A Strategy for Query processing," ACM TODS 2, 3, September 1976.