AN IMPLEMENTATION OF HYPOTHETICAL RELATIONS

by

John Woodfill and Michael Stonebraker

Memorandum No. UCB/ERL M83/2

14 January 1983

ELECTRONICS RESEARCH LABORATORY

An Implementation of Hypothetical Relations

by

John Woodfill and Michael Stonebraker

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

UNIVERSITY OF CALIFORNIA

BERKELEY, CA.

ABSTRACT

In this paper we develop a different approach to implementing hypothetical relations than those previously proposed. Our design, which borrows ideas from tactics based on views and differential files, offers several advantages over other schemes. An actual implementation is described and performance statistics are presented.

1. INTRODUCTION

The motivation for, and applications of hypothetical relations (HR's) were introduced in [STON80]. They can be used to support "what if" changes to a data base and offer a mechanism for debugging applications programs on live data without fear of corrupting the data base. The suggested implementation in [STON80] involved a differential file [SEVR76]. In [STON81], supporting HR's as views [STON75] of the form W = (R UNION S) - T was suggested. In this case an implementation only requires extending a relational DBMS and its associated view mechanism with the UNION and - operators. Moreover, R can be a read-only relation while S and T are append only. As a result, hypothetical relations may

offer cheap support for crash recovery and logging. Unfortunately, there are problems with treating HR's as views. We first examine these problems and show general solutions in Section 2. Next we combine these solutions in Section 3 into a new mechanism for supporting HR's. Our proposal has several similarities but a different orientation from one in [KATZ82]. We then describe our implementation in Section 4. Finally we analyze the performance of this implementation in Section 5.

2. PROBLEMS AND SOLUTIONS

Proposals for hypothetical relations as views contain various flaws which must be removed before a realistic implementation can be attempted.

2.1. A Known Problem

[STON81] points out that the implementation of hypothetical relations as W = (R UNION S) - T is flawed in the case where one wants to re-append a tuple which has been deleted, as shown by the example in figure 1. Initially there is a tuple in relation R corresponding to Eric. Following the algorithm in [STON81], the tuple can be deleted by inserting it into relation T. Lastly a user re-appends Eric and an appropriate tuple is inserted into S. Unfortunately, the resulting relation, W does not contain the re-appended tuple, since (R UNION S) is the same as R, and R - T is empty.

2.2. A Solution

As noted in [Agra82], this problem can be solved by adding a timestamp field to the relations S and T, and modifying the semantics of the DIFFERENCE operator, "-". There will be no timestamps for the relation

R

name	salary
eric	10000

S

eric	10000

T

eric	10000

Figure 1.

R; hence these tuples can be thought of as having a timestamp of zero.

The timestamp field is filled in with the current time (from a system clock, or any other monotonicaly increasing source of timestamps) whenever a tuple is appended to S or T. For any relations A and B with timestamps as described, the DIFFERENCE, A - B is defined as all tuples a in A for which there is no tuple b in B such that

(1) DATA(a) = DATA(b)

and

(2) TIMESTAMP(a) < TIMESTAMP(b)

The definition of R UNION S is unchanged, except for the addition of a timestamp field in the result which contains either the timestamp of a tuple in S, or a zero timestamp for a tuple in R. If tuples with identical DATA appear in both R and S, the newer timestamp (from S) is chosen for tne result tuple.

In the above example, the timestamp of Eric's tuple in T would be newer than that of Eric's tuple in R (zero), but would be older than the timestamp of Eric's tuple in S; hence, (R UNION S) - T would be

equivalent to S, and W would contain the re-appended tuple.

2.3. A New Problem

The addition of timestamps solves the problem of appending deleted tuples. However, this solution is not free from problems. Consider the case of a second level hypothetical relation, W' = (W UNION S') - T', as shown in figure 2. Suppose Eric was given a 20 percent raise in W' at timestamp 10 which caused the indicated entries in S' and T'. Since no updates have occurred in W, S and T are empty. Now suppose a user gives Eric a 50 percent raise in W at timestamp 20, which results in the entries for S and T shown in figure 3. According to the algorithm above, W' would contain two tuples for Eric, one with salary 15,000, and one with salary 12,000. The problem is that the tuple in T' no longer functions to exclude Eric from W UNION S' and hence an unwanted Eric tuple is present.

There are at least two choices for the proper semantics for W' under this update pattern:

R

name	salary
eric	10000

S

name	salary	t-stamp

T

name	salary	t-stamp

S'

name	salary	t-stamp
eric	12000	10

T'

name	salary	t-stamp
eric	10000	10

Figure 2, Eric's 20% raise in W'.

R

name	salary
eric	10000

S

name	salary	t-stamp
eric	15000	20

T

name	salary	t-stamp
eric	10000	20

S'

name	salary	t-stamp
eric	12000	10

T'

name	salary	t-stamp
eric	10000	10

Figure 3, Eric's 50% raise in W.

1) Eric's salary is set to the latest value, in this case the 15,000 from W.

2) Eric's salary is set to 12,000, corresponding to the original update of W'.

We choose to follow the latter choice, and specify the following semantics:

> Once a tuple has been changed at level N, changes at levels < N cannot affect tuples at levels >= N.

2.4. A New Solution

These semantics can be guaranteed by the addition of a tuple identifier, and modification of the DIFFERENCE operator. A tuple identifier, TNAME, must be given to each tuple in R. Each tuple inserted into W (and thereby added to S) must also be given an identifier. Then, any inserts to S or T, which are used to replace or delete a tuple in W, must be marked with the identifier for the original tuple in R or S which they replace or delete. For any relations A and B with timestamps

and TNAMES as described, the DIFFERENCE, A - B is defined as all the tuples a in A for which there is no tuple b in B such that

(1) TNAME(a) = TNAME(b)

and

(2) TIMESTAMP(a) < TIMESTAMP(b)

To guarantee that our chosen update semantics hold, tuples in A - B are given timestamps of zero. Hence, at a second level, each tuple in S' and T' will have a newer timestamp than its corresponding tuple in W.

In our example the identifier of all of the five Eric tuples from figure 3 will be identical. Since the timestamp of the tuple in W is treated as being older than that of the tuple in T', only Eric's tuple from S' will be contained in W'.

A similar method is proposed in [KATZ82], to solve this problem.

3. A MECHANISM

Given these modifications to S, T and the DIFFERENCE operator, an HR of the form W = (R UNION S) - T no longer has its original conceptual simplicity. Moreover, support for HR's becomes considerably more complex than simply implementing UNION and - as valid operators in a DBMS. Consequently, we have designed a mechanism based on differential file techniques which builds on the above developments. The goal is to provide a single-pass algorithm with proper semantics that will support arbitrary cascading of HR's. The next two sections describe our data structure and algorithm in detail.

3.1. The Differential Relation

Each hypothetical relation W, built on top of a real or hypothetical relation B, has an associated differential file D, which contains all columns from B plus plus five additional fields. For example, the differential relation D for the base relation R from Section 2 is shown in figure 4. "Name" and "salary" are the attributes from R. The fields "mindate" and "maxdate" are both timestamps. "Mindate" is exactly the timestamp as defined above, while "maxdate" is another timestamp to be explained in section 4.2. The fields "level" and "tupnum" are used to identify the tuple which this tuple replaces or causes deletion of. Each hypothetical relation is assigned a level number as indicated in figure 5. All real relations are at level zero, and an HR built from a real relation is assigned a level of one. Then an HR built on top of a level one HR is given a level of two. Here the column "level" identifies the level number of a particular tuple, while the column "tupnum" is a unique identifier at that level. Together "tupnum" and "level" comprise the unique identifier, TNAME, of a tuple. Values for "tupnum" are just a sequentially allocated integers. The last field in D, "type," marks what form of update the tuple represents; thus, it has three values, APPEND, REPLACE, and DELETE.

The following examples will illustrate the use of these extra

name	c12
salary	i4
mindate	i4
maxdate	i4
tupnum	i4
level	i1
type	i1

Figure 4.

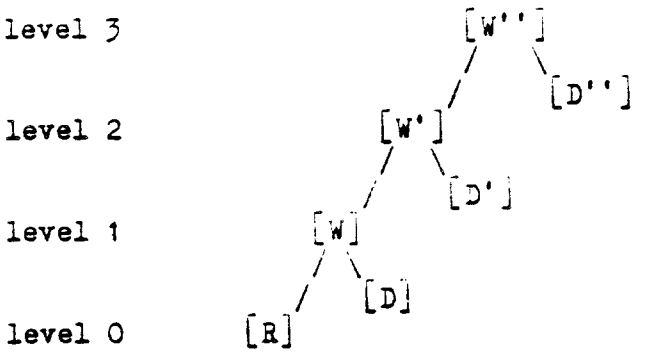

Figure 5.

fields. A precise algorithm is presented in Section 3.2.

Suppose the relation R has the following data:

name	salary
fred	4000
sally	6000

tupnum of this tuple is 0
tupnum of this tuple is 1

Figure 6.

Initially W is identical to R, and D is empty.

Running the following QUEL command:

```
append to W (name = "nancy", salary = 5000)
```

would cause a single tuple to be inserted into D as follows:

name	salary	mindate	maxdate	tupnum	level	type
nancy	5000	30	**	0	1	APPEND

Figure 7.

The 30 stored in "mindate" is simply the current timestamp, and the "type" is clearly APPEND. Since there is no corresponding tuple at level 0, which the tuple replaces, the fields "level" and "tupnum" are

set to identify the tuple itself (i.e. "level" = 1, "tupnum" = 0)

Suppose we now change the salary of Sally as follows:

```
range of w is W
replace w (salary = 8000) where w.name = "sally"
```

After this update, D looks like:

name	salary	mindate	maxdate	tupnum	level	type
nancy	5000	30	**	0	1	APPEND
sally	8000	40	**	1	0	REPLACE

Figure 8.

"Mindate" is 40, the current timestamp. The tuple which we are replacing in R has an identifier of (level = 0, tupnum = 1) (see figure 6).

Suppose we delete the tuple just replaced:

```
delete w where w.name = "sally"
```

The resulting form of D is:

name	salary	mindate	maxdate	tupnum	level	type
nancy	5000	30	**	0	1	APPEND
sally	8000	40	**	1	0	REPLACE
		50	**	1	0	DELETE

Figure 9.

Since this operation is a delete and "name" and "salary" are no longer important, they are set to null. "Tupnum" and "level" are the same as in in figure 8, since they refer to the same tuple.

Suppose we now replace the tuple appended above; eg:

```
replace w (name = "billy") where w.name = "nancy"
```

The resulting form of D is:

name	salary	mindate	maxdate	tupnum	level	type
nancy	5000	30	**	0	1	APPEND
sally	8000	40	**	1	0	REPLACE
	0	50	**	1	0	DELETE
billy	5000	60	**	0	1	REPLACE

Figure 10.

"Tupnum" and "level" identify the original "nancy" tuple (see figure 7 above). At this point, R is unchanged, and W looks like:

name	salary
fred	4000
billy	5000

unchanged
billy replacing nancy

3.2. The Algorithm

There are two parts to the algorithm for supporting hypothetical relations: accessing an HR, and updating an HR.

3.2.1. Accessing Hypothetical Relations

The algorithm for deriving a level N hypothetical relation W from a base relation R and a collection of differential relations D1, ..., DN is a one pass algorithm which starts with the highest level differential relation and proceeds by examining each tuple, passing through each lower level, and finally passing through the level 0 base relation. Figure 11 shows this processing order more clearly. MaxLevel is the level N of the relation H.

An auxiliary data structure, which will be called "seen-ids," is maintained during the execution of this algorithm. This data structure has one associated update routine, "see(level, tupnum)", and a boolean retrieval function, "seen(level, tupnum)". The routine see(level,

```
FOR physlevel := MaxLevel DOWN TO 0 DO
BEGIN
    WHILE (there are tuples at level physlevel) DO
    BEGIN
        tuple := get-next-tuple(physlevel);

        examine-and-process-tuple(tuple, physlevel);
    END
END.
```

Figure 11.

tupnum) inserts a TNAME into the data structure if it nas not been seen before, while seen(level, tupnum) returns the value TRUE if <level, tupnum> is in seen-ids, FALSE otherwise.

The examine-and-process-tuple routine takes one or both of the following actions: it can "accept" the tuple for inclusion in H and it can call the routine "see" to place the identifier in "seen-ids". The choice of actions is dictated by Table 1.

	level0	newest	seen	type	action accept	samelevel	action see
1	yes	------	yes	-------	no	---------	no
2	yes	------	no	-------	yes	---------	no
3	no	no	------	-------	no	---------	no
4	no	yes	yes	-------	no	---------	no
5	no	yes	no	DELETE	no	yes	no
6	no	yes	no	REPLACE	yes	yes	no
7	no	yes	no	APPEND	yes	yes	no
8	no	yes	no	DELETE	no	no	yes
9	no	yes	no	REPLACE	yes	no	yes

Table 1, Processing criteria for HR's.

In applying table 1, to a particular tuple t, "level0" is a boolean condition which is "yes" if physlevel from figure 11, is zero, "no" otherwise. A tuple t at physlevel N is "newest" if (as in Section 2.4) there is no tuple tb at level N such that

(1) (t.level = tb.level and t.tupnum = tb.tupnum)
and
(2) ta.mindate < tb.mindate.

A tuple t has been "seen" when the pair <t.level, t.tupnum> has already been entered into "seen-ids". Fast tests for "newest" and "seen" are presented in Sections 4.2 and 4.3. The "type" of tuple t is t.type. "Samelevel" is a boolean field to indicate if physlevel is the same as t.level. The examining and processing of a tuple is shown in figure 12.

To demonstrate this processing we will generate W from D and R in figures 6 - 10. The starting configuration is shown in figure 13. Processing starts with MaxLevel = 1 and physlevel = 1 in the differential relation D; hence, for all of this level, level0 will be false. Tuple (1) is not "newest", since tuple (4) has the same identifier, and a higher mindate. Since level0 is false, the tuple corresponds to line (3) of table 1, and the tuple is neither "accepted" nor "seen."

Tuple (2) is not "newest" either, because tuple (3) has the same identifier, and a higher mindate, and so it also corresponds to line (3) of table 1, and is neither "accepted" nor "seen."

Tuple (3) is "newest," because the only other tuple at this physlevel with the same identifier, tuple (2) has a smaller mindate. It has not been "seen," since seen-ids is empty and type is DELETE. We now determine "samelevel" by comparing the level field with physlevel. Both are 1, so "samelevel" is true and line (5) is applied. Hence, the tuple is neither "accepted" nor "seen".

Tuple (4) is also "newest," has not been "seen," and type is REPLACE. Comparing level and physlevel, we find "samelevel" is false, since the level field is 0, and physlevel is still 1. hence, (9) is the

```
examine-and-process-tuple(t, physlevel)
BEGIN
    level0     : BOOLEAN;
    newest     : BOOLEAN;
    seen       : BOOLEAN;
    type       : (APPEND, REPLACE, DELETE);
    samelevel  : BOOLEAN;

    level0 := (physlevel = 0);

    IF level0 then
    BEGIN
        newest := NULL;

        seen := seen(t.level, t.tupnum);

        type := NULL;

        samelevel := TRUE;
    END ELSE
    BEGIN
        newest := is_newest(t.mindate, t.level, t.tupnum);

        seen := seen(t.level, t.tupnum);

        type := t.type;

        samelevel := (t.level = physlevel);
    END;

    IF table-accept(level0, newest, seen, type) THEN
        accept-tuple(t);

    IF table-see(level0, newest, seen, type, samelevel) THEN
        see(t.level, t.tupnum);
END;
```

Figure 12, processing a tuple.

D

	name	salary	mindate	maxdate	tupnum	level	type
1	nancy	5000	30	**	0	1	APPEND
2	sally	8000	40	**	1	0	REPLACE
3		0	50	**	1	0	DELETE
4	billy	5000	60	**	0	1	REPLACE

R

	name	salary	
5	fred	4000	tupnum of this tuple is 0
6	sally	6000	tupnum of this tuple is 1

seen-ids = {}

Tuples "accepted"

name	salary

Figure 13, Initial structures for processing W.

correct line in table 1, and the tuple is both "seen" and "accepted". At this point, W and seen-ids look like:

name	salary
billy	5000

seen-ids = {<0, 1>}

Physlevel now changes to 0, "level0" becomes true, and we start to scan the base relation. Only lines (1) and (2) of table 1 are relevant differing in the value of "seen". To check whether a tuple has been "seen," at level 0, we look for the pair <level, location> in seen-ids. For tuple (5) this pair is <0, 0> (see figure 6) which is not in seen-ids. Hence, line (2) of table 1 is applied and we "accept" the tuple. The pair <level, location> for tuple (6) is <0, 1>, which is in seen-ids. The corresponding line is (1), so the tuple is not "accepted," and

is not "seen." We have reached the end of our scan, and have generated the relation W as follows.

name	salary
billy	5000
fred	4000

3.2.2. Updating Hypothetical Relations

All updates to an HR of level N require appending tuples to the differential relation DN at level N. The contents of the different fields in the appended tuple are specified as follows:

(A) For APPENDS and REPLACES, The data columns of DN, are filled with new data. For DELETES, the fields are NULL.

(B) Mindate, is assigned the current timestamp. (Maxdate is discussed in Section 4.2.)

(C) For APPENDS, tupnum and level are set to self-identify the inserted tuple. For DELETEs and REPLACEs tupnum and level identify the target tuple being deleted or replaced.

(D) Type is the type of the update, APPEND, DELETE or REPLACE.

4. IMPLEMENTATION

An implementation of HR's was done within the INGRES DBMS [STON76]. In order to create an HR, the following addition to QUEL was made:

DEFINE HYPREL newrel ON baserel

Once an HR has been defined, it can be updated and accessed just like an ordinary relation. Since, "baserel" can be either a regular relation, or an HR, an unlimited number of levels is allowed.

4.1. Modifications

Within the INGRES access methods, a relation is accessed first by a call to "find" which sets the range for a scan of tuples, and then "get" is called repeatedly to access each tuple in this range. It is within "get" that most of the HR algorithm is implemented. "Get" returns tuples from each differential relation, and finally the tuples from the base relation. The routines which perform REPLACES, DELETES, and APPENDs are also modified to initialize and append the appropriate tuples to the differential relation.

4.2. Newest

If tuples were appended to a differential relation at one end, and the relation were scanned from the other direction, it would be possible to tell when a tuple was the "newest" for a particular identifier by the fact that it was the first one encountered. Unfortunately, INGRES appends tuples and scans relations in the same direction. In order to be able to tell from a single pass whether a tuple is "newest", an additional timestamp field "maxdate" was added. When a tuple is appended, maxdate is set to infinity. When the tuple is REPLACED or DELETED at the same level, maxdate is updated. Thus a tuple is the "newest" if the time of the current scan is between mindate and maxdate.

4.3. Seen-ids

The data structure, seen-ids is stored in a series of main memory bit-maps, one for each level. Thus to see a tuple with tupnum Y at level L, bit Y in bitmap L is set. The boolean function "seen(L, Y)" tests wnether the corresponding bit is set.

4.4. Optimization

If the base relation is organized as either a random hash structure or an ISAM structure, then the differential relations can be given a similar structure and a sequential scan of the differential relation avoided. To accomplish this, a correspondence must be established between the pages in a differential relation and those in the base relation. If a tuple would be placed on a certain page of the base relation, then the tuple in the hypothetical relation must be placed on the corresponding page in the differential relation.

To access a tuple in such a structured HR, the scan within each relation is restricted to those pages corresponding to the key of the query. For example, suppose the relation R(name, salary) is stored hashed on name and the differential relation D is stored likewise. Then, the query

```
range of w is W
retrieve (w.all) where w.name = "billy"
```

only requires accessing the appropriate hash bucket in both R and D.

There is one complication with this performance enhancement, which stems from the fact that a REPLACE command can change the hash key, and hence the page location of a tuple in a structured relation. For example, consider the following contents of R and D:

R

hashbucket	name	salary
1	suzy	3000
2	tandy	25

D

name	salary	other

Figure 14, R and D hashed on name.

Then, suppose we do the following REPLACE:

```
range of w is W
replace w (name = "tandy") where w.name = "suzy"
```

As a result, R and D would look like

R

hashbucket	name	salary
1	suzy	3000
2	tandy	25

D

name	salary	type	*
tandy	3000	REPLACE	

Figure 15, problematic hashed replace.

and the query:

```
retrieve (w.all) where w.name = "suzy"
```

would generate the result:

name	salary
suzy	3000

Despite the fact that we changed suzy's name, she appears in the result because the algorithm indicates searching hashbucket 1 of D, where there are no tuples, then searching hashbucket 1 of R, where susy's tuple appears. This tuple in hashbucket 1 of R is "accepted", because no tuples have been "seen." Unfortunately, the algorithm never searches hashbucket 0 of D to discover the correct tuple.

This problem can be solved by the addition of a fourth type of differential tuple, FORWARD. An additional FORWARD tuple is appended in hashed and ISAM differential relations whenever a REPLACE is done which inserts a tuple in a different hashbucket (or ISAM data page) than that of the target tuple. With this correction, D of figure 15 would look like:

hashbucket	name	salary	mindate	maxdate	tupnum	level	type
1		0	100	INFINITY	0	0	FORWARD
2	tandy	3000	100	INFINITY	0	0	REPLACE

Figure 16.

The processing of the query would then start in hashbucket 1 of D in figure 16, where a FORWARD tuple would be found, and the ordered pair <0, 0> would be added to seen-ids. Next, hashbucket 1 of R would be scanned, but since <0, 0> is in seen-ids, Suzy's tuple, tuple 0 of R, would not be accepted.

4.5. Functionality

With this refinement all QUEL commands have been made operational on HRs for any INGRES storage structure. Such HR's could be used as the basis for a crash recovery scheme as suggested in [STON81] with minor modifications to the our algorithms. Moreover, "snap-shots" of the state of an HR at any point in the past can be generated by setting the scan time to a time prior to the current time. Minor changes to the QUEL syntax would allow a user to run retrieval commands against an HR as of some previous point in time.

If at any time one wanted to make the changes in an HR permanent, he can use a series of QUEL statements to update the base relation using the information in the differential relations. Alternately, a simple utility could be constructed to perform the same function.

5. PERFORMANCE MEASUREMENT AND ANALYSIS

Our performance analysis is aimed at comparing the performance of standard QUEL commands on real relations versus the same ones on HRs and

our tests were run on a single user VAX-11/780. The following four commands are used to measure update performance for a real parts relation parts5000(pnum, pname, pweight, pcolor) of 5000 tuples stores as a heap. Baseparts will serve both as a real relation and an HR.

```
range of b is baseparts
range of p is parts5000

(a) append to baseparts (p.all)

(b) delete b

(c) replace b (weight = b.weight + 1000)

(d) replace b (pnum = b.pnum * 1000)
```

Table 2 indicates the results of running commands a) - c) first for a real baseparts relation of 5000 tuples stored as a heap and then for baseparts as an HR. In the latter case it consists of an empty differential relation, D and a 5000 tuple real relation, R stored as a heap. Command d) was not run in this situation because it should produce comparable results to command c) for unstructured relations. Notice that real and hypothetical relations perform comparably.

To test retrieval performance we ran query (e) for four different compositions of baseparts, including

```
range of b is baseparts
(e) retrieve (m = max(b.weight))
```

a 10 tuple real relation, a 10000 tuple real relation, a 10 tuple HR and a 10000 tuple HR. The hypothetical relations had sizes of differential

query	operation	relation-type	cputime	elapsed
(a)	append	regular	24.47 secs	32 secs
(a)	append	hypothetical	26.57 secs	36 secs
(b)	delete	regular	24.38 secs	26 secs
(b)	delete	hypothetical	19.78 secs	25 secs
(c)	replace	regular	26.03 secs	28 secs
(c)	replace	hypothetical	25.03 secs	35 secs

Table 2, updates on 5000 tuples unstructured.

query	operation	relation-type	cputime	elapsed
(a)	append	regular	74.68 secs	268 secs
(a)	append	hypothetical	64.82 secs	226 secs
(b)	delete	regular	20.15 secs	31 secs
(b)	delete	hypothetical	21.32 secs	37 secs
(c)	replace	regular	42.32 secs	47 secs
(c)	replace	hypothetical	40.97 secs	59 secs
(d)	replace	regular	91.33 secs	345 secs
(d)	replace	hypothetical	89.63 secs	422 secs

Table 3, updates on 5000 tuples, hashed on salary.

relations, D, varying from 0 to 200% of the size of the R. Tables 4 and 5 show the results of these tests.

relation type	size of D	cputime	elapsed time
regular	-	0.16 secs	1 sec
hypothetical	0%	0.20 secs	1 sec
hypothetical	50%	0.26 secs	1 sec
hypothetical	100%	0.26 secs	1 sec

Table 4, Query (e) run with 10 tuple base.

relation type	size of D	cputime	elapsed time
regular	-	11.88 secs	13 secs
hypothetical	0%	13.86 secs	15 secs
hypothetical	10%	14.40 secs	15 secs
hypothetical	25%	15.22 secs	16 secs
hypothetical	50%	16.73 secs	18 secs
hypothetical	100%	18.60 secs	21 secs
hypothetical	200%	21.58 secs	30 secs

Table 5, Query (e) run with 10000 tuple base.

Query (e) was also run against a second level HR based on a first level HR with 50% of its tuples replaced. The results of this test are in table 6.

Lastly, we ran query (f) against a baseparts relation hashed on pnum.

```
range of p is parts5000
range of h is RELATION

(f) retrieve (p.weight, h.weight) where p.pnum = h.pnum
```

In this case table 7 compares performance where RELATION is either a 5000 tuple real relation hashed on pnum, or a 5000 tuple HR hashed on

hypothetical relation level	size of D	cputime	elapsed time
1	50	16.73 secs	18 secs
2	0%	17.35 secs	18 secs
2	10%	17.73 secs	19 secs
2	25%	18.52 secs	19 secs
2	50%	18.78 secs	21 secs
2	100%	20.75 secs	24 secs

Table 6, Query (e) 10000 tuples, 2 levels.

pnum, with 50% of its tuples replaced. Parts5000 is an unstructured

Query (f)

relation	type	cputime	elapsed
hashparts	regular	131 secs	5.85 minutes
hhashparts	hypothetical	185 secs	9.88 minutes

Table 7, hashed access results.

5000 tuple relation.

Two comments are appropriate about the numbers in Table 7. First, notice that INGRES is I/O bound in both tests and elapsed time substantially exceeds CPU time. The reasons include the particular query processing tactic chosen for this query and the fact that a substantial amount of data is printed on the output device. The second point is that joins on hypothetical relations are less than a factor of two slower than those on real relations.

Thus we can see that the performance of INGRES using hypothetical relations in many types of query is never worse than a factor of its level number and usually much better. We assume that for more complex queries involving an HR, the same general result would hold.

6. CONCLUSIONS

We have described a mechanism for supporting HR's which is shown to overcome the problems of previous proposals. We have described an implementation of HR's and provided performance data to show that performance of HR's is in general no worse than a factor of one per level of HR. Moreover, in most cases, performance is considerably better than

this.

ACKNOWLEDGEMENT

This research was supported by the Advanced Research Project Agency under contract #N00039-C-0235.

REFERENCES

[AGRA82] Agrawal, R. and DeWitt, D. J., "Updating Hypothetical Data Bases," Unpublished working paper.

[KATZ82] Katz, R. and Lehman, T., "Storage Structures for Versions and Alternatives," University of Wisconsin - Madison, Computer Sciences Technical Report #479, July 1982.

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

[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.

[STON80] Stonebraker, M. and Keller, K., "Embedding Expert Knowledge and Hypothetical Data Bases Into a Data Base System," Proc. 1980 ACM-SIGMOD Conference on Management of Data, Santa Monica, Ca., May 1980

[STON81] Stonebraker, M., "Hypothetical Data Bases as Views," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., June 1982.

8. Statement of Work

8.1. Operating Systems, Distributed Computing, and Programming Systems

- We will implement the interprocess communication and large file access enhancements to UNIX and make them available as part of the Berkeley Software Distributions. We will have a substantially complete experimental version of the system with the large file access enhancements documented in a technical report by March 1982. We will have a substantially complete experimental version of the system with the interprocess control enhancements documented in a technical report by June 1982. We will have a complete system including the large file access and interprocess control enhancements ready for distribution by September 1982.
- We will implement a UNIX-based distributed computing environment in the context of a network of personal workstations and larger computers not necessarily under a common administration. We will create experimental versions of various components of such an environment and document them with technical reports throughout the contract period. We will have an experimental version of the distributed system documented in a technical report by March 1983. We will have an initial version of a distributed system ready for distribution by September 1983.
- We will construct a table driven code generator which takes input from the first pass of the Portable C Compiler, the Fortran 77 compiler and the Berkeley Pascal compiler. A technical report describing the implementation and comparing its output to that of the existing compilers will be provided by September 1982. We will explore techniques for improving the generated code, implement those which appear best, and examine their impact in another technical report by September 1983.

- We will explore basic issues related to distributed computing throughout the contract period and document our research results in technical reports.

8.2. Information Management in Design and Decision Support Systems

- We will investigate how to extend the class of data representations which can be processed by a relational database system. We will extend INGRES to allow multiple representations of data items using the techniques of descriptor based access methods and will document the result in a technical report by September 1982. We will introduce the notion of bins into INGRES to provide efficient processing of spatial data; the bins will be implemented using a generalization of secondary indices and will be documented by a technical report by March 1983. We will investigate using a relational database system as an AI programming tool by experimentally rewriting some existing AI programs to use a version of INGRES which has been enhanced to allow storing information which would have been stored using Lisp; the experiment will be described in a technical report by September 1983.
- We will explore the use of forms as an efficient interface for developing various applications of database systems. The specification of a form application development system will be provided as a technical report by June 1982. A prototype system will be developed and documented in a technical report by December 1982. During the remainder of the contract period we will build a variety of applications using the prototype system in order to evaluate its interface.

8.3. Interfaces and Graphics

- We will explore connection-based style of design including how to represent and manipulate connections graphically, how to hide the details of complex connections using the concept of bundles, and how to deal with geometrical constraints. We will measure relevant aspects of existing designs and design tools to provide a context for this research. These measurements will be documented in a technical report by June 1982. We will develop a simple connection-based design

system and describe it in a technical report by June 1983.

- We will study and build a prototype mathematical software environment based on workstations which communicate with remote computers. The workstations will be graphics based and will provide the user with an integrated interface. The large computer will provide a large scale algebraic/numerical computation environment for effective problem solving. The user interface to be provided by the workstation will be designed and spelled out in a technical report by September 1982. A working user interface to Macsyma provided via a workstation will be documented in a technical report by March 1983 and a system with interactive and graphical enhancements will be documented in a technical report by September 1983.

- We will conduct both theoretical and experimental research into the applicability of Beta-splines for curve and surface representation in computer graphics systems which allow the representation and modification of geometrical shapes. A basic experimental graphics facility for use from within the UNIX environment will be constructed and documented in a technical report by September 1982. A technical report evaluating subdivision techniques for Beta-splines will be provided by September 1983.