Representing Databases in Frames

Ey-Chih Chow

Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, California 94304

Abstract

Three methods for representing data in a relational storage system with an in-core frame-based system are experimented with and reported upon. Tradeoffs among these three representational methods are sizes of databases, times for loading data, and performance of queries. Essentially, these methods differ in ways of capturing relationships among frames. The three different ways of capturing such relationships are via links (pointers), symbolic names (keys), or both. Results of the experiments shed light on efficient interfacing of databases with frame-based systems.

1 Introduction

Frame-based systems have become popular in building expert systems [Fikes and Kehler, 1985]. A current research topic along these lines is how to efficiently hook disk-based database systems together with in-core frame-based systems to extend the capabilities of both systems [Abarbnel and Williams, 1986]. As a step toward this outcome, this paper discusses the performance aspects of ways of representing data in relational storage systems using frames.

By allowing slots to be pointers to other frames and to be multiple-valued [Stefik, 1979], frames are able to capture relationships among objects (frames) effectively. However, because putting pointers like these on disk leads to too many disk accesses, relationships among objects (tuples) in relational database systems are expressed instead via keys or other symbolic identifiers [Chamberlin *et al.*, 1981].

To investigate the tradeoffs between using pointers or keys with in-core databases, a conventional relational database benchmark, **Fasttrack** [Chow, 1986] and [Chow and Cate, 1986], was adopted to compare three in-core frame-based alternatives used to represent the benchmark database. Experiments involving this comparison were conducted in terms of HP-RL [Rosenberg, 1983] and [1986b], an in-house frame-based expert-system toolkit running at Hewlett-Packard Laboratories. Results show that representing databases via pointers can improve the performance of queries involving joins. However, sizes of databases with such pointers are larger than those with only keys or other symbolic names. Obviously, the larger the databases, the longer the loading time needed. Additional findings were that evaluating database queries using the very general query-handling mechanism in HP-RL, under the environment of NMODE on top of HP-9000/300 UNIX[1] workstation [1986a], suffered from low hit ratios. Furthermore, with the same environment, evaluating complex database queries, i.e. queries involving large amounts of relational data and joins, it is easy to incur garbage collection. An integrated system, combining HP-RL and Iris [Fishman *et al.*, 1987], a next-generation database system with underlying relational storage and processing being developed at Hewlett-Packard Laboratories, is then proposed and is being further investigated to take advantage of Iris' ability to efficiently handle data of disk-based databases.

Section 2 describes three different HP-RL database designs for **Fasttrack**. In Section 3, we discuss the performance tradeoffs of these three designs. Section 4 suggests a way to use both the Iris database management system and HP-RL to take advantage of their strengths. Finally, we give a brief summary of our observations in Section 5.

[1] UNIX is a trademark of AT&T Bell Laboratories.

```
(new-instance {dealers}
 :name     D0
 :slots ((dlrid   :v  1000)
         (name    :v  "eddie")
         (phone   :v  "4087258111")
         (misc    :v  "Misc Data")
         (daddr   :v  "19420 Homestead Ave.")
         (has     :vs ({OU0}{OU1}{OU2}))
         (receive :vs ({OR0}{OR1}{OR2}))
         ))
```

(a) An instance of dealers

```
(new-instance {outlets}
 :name    OU0
 :slots ((oaddr        :v  "1180 Lochinvear Ave.")
         (zip_code     :v  0)
         (owned_by     :v  {D0})
         (competed_for :vs {C0}{C1501}{C3001})
         ))
```

(b) An instance of outlets

Figure 1: Instances of classes **dealers** and **outlets** in schema #1

2 Fasttrack Database in Frames

The conventional database benchmark, **Fasttrack**, basically includes a simple sales-record system and some test queries. The system consists of dealers that may operate several outlets. Outlets have competitors, determined by matching zip code. Customers order products from dealers. A typical execution creates 500 dealers, 100 products, 1500 orders, and 1000 outlets, and contains information about 4500 competitors. The relational schema [Date, 1986] of this database is as follows:

```
dealers (dlrid, name, phone, misc, daddr)
outlets (dlrid, zip, oaddr)
orders (dlrid, prod, qty, date)
products (prod, price, desc)
competitors (zip, compid, cratio, prodtyp)
```

where **dlrid**, **prod**, and **zip** are primary keys of relations **dealers**, **products**, and **outlets** respectively. Note that, in the above schema, relationships among entities are represented via keys. For example, relationships between dealers and outlets are represented via **dlrid**s

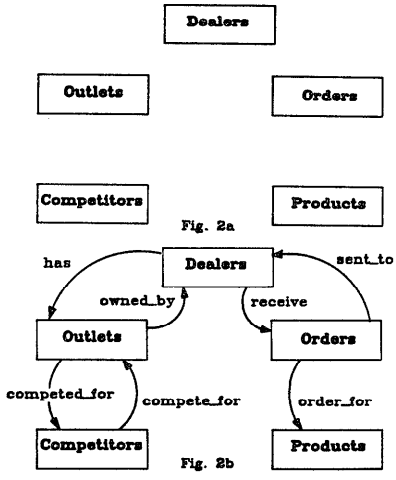

Figure 2: Pointers vs. keys in the Fasttrack database

in relations **outlets** and **dealers** respectively, where **dlrid** is a foreign (primary) key of relation **outlets** (**dealers**.)

In terms of frames, the above database can be represented by three alternative schemata. Each of these three schemata has the same five classes of frames (or objects): dealers, orders, outlets, products, and competitors. However, schema#1 defines relationships among frames with pointers only. With this schema, a pair of instances of classes **dealers** and **outlets** respectively are shown in Figure 1a and 1b. In this figure, instances of frames OUn, n = 0,1,2, are frames of outlets, ORn, n = 0,1,2, are frames of orders, D0 is a frame of dealear, and Cn, n = 0,1501,3001 are frames of competitors. Curely brackets of frames denote pointers to the frames. Finally, facet **value** (**values**) of a slot is denoted by **v** (**vs**).

In this way, relationships between dealers and outlets, for example, are represented via pointers of slots **has** and **owned_by** in instances of classes **dealers** and **outlets** respectively. Note that slot **has** is multiple-valued. Schema#2 defines relationships among frames with only keys as the above relational representation. With this schema, relationships between dealers and outlets are represented via keys of slots **dlrids** in both classes **outlets** and **dealers**, where both **dlrids** are single-valued. Schema#3 defines relationships among frames with redundant information, using both keys and pointers. This schema is used to investigate the possible performance improvement due to such redundant information. Note that the notion of keys adopted in schema #2 and schema #3 are borrowed from disk-based databases. Scheme #1 is a more typical representation for a knowledge representation language such as HP-RL. The three schemata are shown in Appendix I.

Frames which use foreign keys to relate to other frames can be deemed to have some implicit pointers among these frames. Namely, foreign keys can be viewed as a kind of (slightly indirect) pointer. With this viewpoint, schema #2, i.e. frames with keys only, can be conceptually shown in Figure 2a, where pointers are implicit.

Frames which with pointers to other frames, on the other hand, can be viewed as extended semantic networks [Rich, 1983] whose nodes are frames themselves and whose arcs are pointers to other frames. Schemata #1, i.e. frames with pointers to other frames, can, therefore, be shown in Figure 2b, where pointers are explicit. The meaning of pointers in Fig. 2b is that dealers **have** outlets and **receive** orders, orders are **sent_to** dealers and **order_for** products, outlets are **owned_by** dealers and are **competed_for** by competitors, and competitors **compete_for** outlets. Note that each pointer of a frame pointing to another frame can be represented as a slot of the former frame.

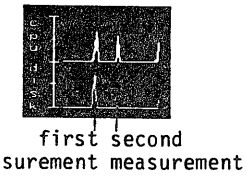

Figure 3: CPU and disk usages in processing an HP-RL query

3 Performance Tradeoffs

This section discusscs performance tradeoffs among the above three schemata. The discussions are divided into three parts: performance of queries, database sizes, and loading.

3.1 Performance of Queries

We used six queries to test the above three schemata. In terms of SQL [Date86], these six queries against the relational schema mentioned in section 2 are as follows:

```
q1.  select phone from dealers where dlrid = 1260
q2.  select dlrid from outlets where zip = 1260
q3.  select oaddr,zip from outlets where dlrid = 1370
q4.  select cratio,compid from outlets,competitors
     where outlets.dlrid = 1150 and outlets.zip =
     competitors.zip
q5.  select dealers.name,products.desc from
     dealers,orders,products where
     orders.qty between 10 and 20 and
     dealers.dlrid = orders.dlrid and
     products.prod = orders.prod
q6.  select prod,price,desc from products
```

Queries **q1**, **q2** and **q3** include selections and projections. Queries **q4** and **q5** are join queries. The selectivity factor of the join in **q4** is 0.07%. The selectivity factors of the two joins in **q5** are 0.2% and 1% respectively. Query **q6** is a query to retrieve all the data in relation **prod**.

As in **Prolog** [Warren, 1981], forms of queries in HP-RL affect their efficiency of retrieval tremendously. Optimum ways to express the above six test queries in HP-RL have been designed based on our inspection of the underlying data statistics and structures of schemata. We avoid joins as much as possible in expressing queries. Optimum ordering of joins of queries are chosen according to the underlying data statistics and the nature of the query-evaluation algorithm in HP-RL. We also take advantage of possible pointers between objects to express joins. The three groups of translated HP-RL test queries for the corresponding three schemata are shown in Appendix II.

Measurements were based on running queries two consecutive times and were made for each query right after garbage collection. An interesting feature of response times for simple queries is that after garbage collection, much useful code and data for query evaluation is not really in core and needs to be paged in. This incurs a very low hit ratio. Therefore, the elapsed time for a query at the first measurement is affected by the size of the underlying database and is much longer than that at the second measurement. For example, the elapsed time to process **q1** of schema #1 at the first measurement is a factor of 30 slower than at the second measurement. Figure 3 shows snapshots of usage of system resources in processing an HP-RL query. There is a big I/O peak the first time a query is processed and almost no I/O the second time. The hit ratio is not as low if a query

	HP-RL Schema#1		HP-RL Schema#2		HP-RL Schema#3		Iris
	1st	2nd	1st	2nd	1st	2nd	
q1	2.44	0.08	4.04	0.08	6.96	0.08	9
q2	8.68	0.74	5.74	0.75	9.14	0.80	4
q3	4.36	0.28	6.18	0.94	4.28	0.30	3
q4	3.86	0.86	13.00	7.32	6.32	1.10	30
q5	587.40	665.62	744.42	813.76	595.68	688.70	57
q6	15.36	10.18	16.06	10.12	15.22	10.96	30

Table 1: Response times of test queries (sec)

HP-RL schema #1		HP-RL schema #2		HP-RL schema #3	
1st	2nd	1st	2nd	1st	2nd
7.24	3.16	458.92	498.36	8.12	3.30

Table 2: Response times of the 1st answer of q5 (sec)

is not posed right after garbage collection. Very often, response times of ad hoc queries are 2 - 3 times that of the corresponding second measurements mentioned above. The above phenomenon will be hidden when much garbage collection is involved in query evaluation, that is, when there are many joins and duplicate answers of a query. For example, processing q5 of schema #1 the second time takes even longer than the first time by a factor of 1.13. This is because the first measurement is made right after garbage collection, which makes this measurement have less garbage collection than the second one. The HP-RL command `solve-all` suffers from garbage collection in processing joins involving large amounts of data because it is likely to create many temporary frames in such processing. There is another relatively specialized command in HP-RL, `fast-solve-all`, which partly solves the above problem but does not relieve it entirely.

Table 1 shows response times for the six queries in HP-RL schemata, listing both the first and second measurements. Performance of the queries in Iris is also attached, where the numbers are measured without index. Queries q3, q4, and q5 take longer in schema #2 than in schema #1 and #3. This shows that defining objects to link through pointers is faster than naming keys of objects. Time savings of q5 in schema #1 and #3 against schema #2 are only about 26%. This is because most of the time spent in processing this query is used to eliminate duplicate answers. In q5, by replacing `solve-all` with `solve` to measure times spent in getting the first answer, we find that this query in schema #1 or #3 is faster than in schema #2 by a factor of 60. This measurement is shown in Table 2.

Note that, unlike schema #1, in schema #3 q2 avoids a join because each instance of `outlets` has the key of its dealer. But q2 takes about the same time in schema #1 and #3. Therefore, it is not necessary to keep redundant information the way schema #3 does.

Now, we compare the performance of the HP-RL queries with that of the corresponding Iris queries. For a reasonably high hit ratio, HP-RL is superior to Iris in processing simple database queries such as q2 (HP-RL: at most 0.80s vs. Iris: with index 3.8s.) For an exceptional low hit ratio, however, Iris performs better than HP-RL in processing database queries (for q2, HP-RL: at least 5.74s vs. Iris: no index 4s.)

In addition, with pointers as in schema #1, HP-RL is able to handle simple relational joins, i.e., joins not involving too many duplicates and data, better than Iris does (for q4, HP-RL: 0.86s vs. Iris: with index 5s.) Iris, however, is much better than HP-RL at handling complex relational joins, i.e., joins involving many duplicates as well as large amounts of data (for q5, HP-RL: $\approx$ 600s vs. Iris: no index 57s.) Essentially, this is because HP-RL retrieval commands like `solve-all` and `fast-solve-all` are intended to handle much more general kinds of data than Iris does. Therefore, in the specific environment of databases, HP-RL suffers from garbage collection during relational joins and eliminating duplicates.

3.2 Database Sizes

Sizes of the three HP-RL databases with the schemata mentioned in section 2 depend on how the slot is implemented. For HP-RL default slot declarations, slots have three facets: `daemon`, `value`, and `comment`. In this case, sizes of the database are 2.73 Mbytes for schema #1, 2.31 Mbytes for schema #2, and 3.15 Mbytes for schema #3. However, for database-oriented frames like those of `Fasttrack`, only one facet, i.e. `value`, for each slot is enough. This can be achieved with an HP-RL command to override the default declaration. With this kind of slot implementation, sizes of database are 1.57 Mbytes for schema #1, 1.31 Mbytes for schema #2, and 1.70 Mbytes for schema #3.

The database of schema #1 is bigger than that of schema #2 because, although keys of objects are not used to represent relationships among objects in schema #1, information associated with such keys still needs to be kept. For example, key `zip_code` of class `outlets` of schema #1 cannot be ignored without losing information. Schema #3 uses redundant information of both pointers and keys in representing relationships. Therefore, the associated database is the biggest of the three HP-RL databases.

	sequence #1	sequence #2
products	19.84s	21.14s
orders	924.10s	912.80s
dealers	683.70s	365.52s
competitors	5066.78s	5153.76s
outlets	1175.56s	2179.30s
TOTAL	131min & 29 sec	143min & 52 sec

Table 3: Loading times of schema #1 (elapsed time)

3.3 Loading

Due to frequent garbage collection, loading times of HP-RL databases depend on sizes of free dynamic heap spaces where the `Fasttrack` database is stored. The size of the dynamic heap of the HP-RL configuration used in these measurements is 5.82 mbytes with 4.56 mbytes free. In addition, to make the database smaller, each slot of each frame to be loaded in this measurement allows only one facet, i.e., `value`.

There is a problem in loading frames with pointers. As a frame is loaded with reference to nonexistent frames, some partial frames, i.e., frames with only headers but no bodies, are created and echoed to the screen. Due to the expense of such echos, sequences that load objects with minimal numbers of echos require minimal time. The following two sequences of loading objects were tested:

```
sequence #1.  products, orders, dealers, competitors,
              outlets.
sequence #2.  products, orders, outlets, dealers,
              competitors.
```

Loading times of sequence #1 and #2 for schema #1 are shown in Table 3. Loading via sequence #2 is slower than sequence #1 because of the following reason. During the loading of outlets with sequence #2, instances of `competitors` are nonexistent. Accordingly, partial frames are created and are echoed to the screen. Since instances of `competitors` are much more numerous than those of the other four classes. Echos on loading via sequence #2 are more than those via sequence #1.

In the following discussions, loading times of databases of schema #1 and #3 were measured via loading sequence #1. Among the three HP-RL databases, the loading times increase (about 104 : 131 : 157 mins for schema #2 : schema #1 : schema #3) roughly in linear proportion to the increases of the corresponding database sizes (1.31 : 1.57 : 1.70 mbytes), with a ratio about 1. In addition to the size of the database, another reason that the database of schema #2 had the shortest loading time is that no partial frames (and therefore no

echos) are created in loading this database.

To summarize, of the three HP-RL schemata, schema #3 does not appear to have any advantage over schema #1. In contrast to schema #2, schema #1 trades good performance of queries for some memory space and extra loading time. Finally, Iris is able to handle complex joins better than HP-RL, although HP-RL is faster for simple queries.

4 Interfacing Databases with Frames

In this section, we propose a way to build high-performance knowledge base systems by combining Iris and HP-RL.

Iris is designed to handle large amounts of data in wide ranges of applications. Expert systems built in HP-RL, however, deal with a limited domain in a particular session. To achieve high performance using combined HP-RL and Iris, for a particular application domain, we retrieve and transform related Iris data of relational forms into frames and cache them in the HP-RL heap space during run time. Note that retrieval of data in a particular domain from Iris databases tends to involve complex joins. In addition, answers to queries from Iris often include duplicates. Such duplicates should be eliminated before cache. In the transformation of Iris data to frames, relationships among these frames should be represented by either keys, as in schema #2 mentioned in the previous sections or pointers, as in schema #1. Considerations involved in selecting one of these two alternatives are memory sizes, loading time and performance of queries. For example, if free space of HP-RL is sufficient and cached objects are dynamically preloaded to HP-RL, then performance of queries is the only consideration. In this case, using pointers to link objects (frames) is an appropriate way to gain efficiency.

Interfacing Iris with HP-RL in the above way tends to make Iris handle complex joins and HP-RL handle simple queries. This achieves high performance. The above discussion, however, does not address the issues of update and insert as data are cached back to Iris. This still needs to be investigated.

5 Conclusions

This paper uses some experimental results to describe the tradeoffs among three representations of databases in frames. Representing relationships among frames via pointers will get better performance on queries than will representation via keys or symbolic names. However, pointers will make the corresponding databases larger than will keys or symbolic names. The same benchmarks are used to compare the frame-based system, HP-RL, with the Iris database system, a next-generation database system with underlying relational storage and processing. Results show that for simple database queries, at a reasonable hit ratio, HP-RL tends to be faster than Iris because data for Iris queries are on disk. Because of its sophisticated buffer management strategy and relatively specialized code in handling data of conventional databases, however, Iris is able to handle queries involving complex relational joins much better than HP-RL does.

Acknowledgments

The author is grateful to Dan Fishman, Steve Rosenberg, Henry Cate, Tom Ryan, Reed Letsinger, Bill Stanton, Pierre Huyn and Alan Shepherd for most inspiring discussions and comments of this work. Charles Hoch, Jim Davis, Wendell Fields, and Randy Splitter have provided excellent and responsive support of the experimental environments. The author also thanks the referees for suggesting several important improvements to the paper.

Appendix I. Fasttrack database in HP-RL

Schema #1

```
(define-class dealers ()
 :instance-slots
 ((dlrid        :declare (single-valued)   ;key of dealer;
                :type number)
  (name         :declare (single-valued)
                :type string)
  (phone        :declare (single-valued)
                :type string)
  (misc         :declare (single-valued)
                :type string)
  (daddr        :declare (single-valued)
                :type string)
  (has          :declare (multiple-valued) ;pointers to outlets
                :type {outlets})
  (receive      :declare (multiple-valued) ;pointers to orders
                :type {orders})
 ))
(define-class outlets ()
 :instance-slots
 ((oaddr        :declare (single-valued)
                :type string)
  (zip_code     :declare (single-valued)   ;key of outlets
                :type number)
  (owned_by     :declare (single-valued)   ;pointers to dealers
                :type {dealers})
  (competed_for :declare (multiple-valued) ;pointers to competitors
                :type {competitors})
 ))
(define-class orders ()
 :instance-slots
 ((qty          :declare (single-valued)
                :type number)
  (date         :declare (single-valued)
                :type string)
  (sent_to      :declare (single-valued)   ;pointers to dealers
                :type {dealers})
  (order_for    :declare (single-valued)   ;pointers to products
                :type {products})
 ))
(define-class products ()
 :instance-slots
 ((prod#        :declare (single-valued)   ;key of products
                :type number)
  (price        :declare (single-valued)
                :type number)
  (desc         :declare (single-valued)
                :type string)
 ))
(define-class competitors ()
 :instance-slots
 ((compid       :declare (single-valued)
                :type number)
  (cratio       :declare (single-valued)
                :type number)
  (prodtype     :declare (single-valued)
                :type string)
  (compete_for  :declare (single-valued)   ;pointers to outlets
                :type {outlets})
 ))
```

Schema #2

```
(define-class dealers ()
 :instance-slots
 ((dlrid      :declare (single-valued)   ;key of dealers
              :type number)
  (name       :declare (single-valued)
              :type string)
  (phone      :declare (single-valued)
              :type string)
  (misc       :declare (single-valued)
              :type string)
  (daddr      :declare (single-valued)
              :type string)
 ))
(define-class outlets ()
 :instance-slots
 ((dlrid      :declare (single-valued)   ;key of dealers
              :type number)
  (zip_code   :declare (single-valued)   ;key of outlets
              :type number)
  (oaddr      :declare (single-valued)
              :type string)
 ))
(define-class orders ()
 :instance-slots
 ((dlrid      :declare (single-valued)   ;key of dealers
```

```
            :type number)
  (prod     :declare (single-valued)   ;key of products
            :type number)
  (qty      :declare (single-valued)
            :type number)
  (date     :declare (single-valued)
            :type string)
 ))
(define-class products ()
 :instance-slots
 ((prod#    :declare (single-valued)   ;key of products
            :type number)
  (price    :declare (single-valued)
            :type number)
  (desc     :declare (single-valued)
            :type string)
 ))
(define-class competitors ()
 :instance-slots
 ((zip      :declare (single-valued)   ;key of outlets
            :type number)
  (compid   :declare (single-valued)
            :type number)
  (cratio   :declare (single-valued)
            :type number)
  (prodtype :declare (single-valued)
            :type string)
 ))
```

Schema #3

```
*** Slots of classes in this schema are combinations of slots of the
    corresponding classes in schema #1 and #2. ***
```

Appendix II. Test queries in HP-RL

Schema #1

```
query 1
(solve-all '({D260} phone ?x)
           :returns '?x)
query 2
(solve-all '(and (?x zip_code 1260) (?x owned_by ?y) (?y dlrid ?z))
           :type '((?x {outlets}) (?y {dealers}))
           :returns '?z)
query 3
(solve-all '(and ({D370} has ?x) (?x oaddr ?y) (?x zip_code ?z))
           :type '((?x {outlets}))
           :returns '(?y ?z))
query 4
(solve-all '(and ({D150} has ?x) (?x competed_for ?y)
                 (?y cratio ?z) (?y compid ?u))
           :type '((?x {outlets}) (?y {competitors}))
           :returns '(?z ?u))
query 5
(solve-all '(and (and (?y qty ?z) ^(>= ?z 10) ^(<= ?z 20))
                 (?y order_for ?w) (?w desc ?u) (?y sent_to ?x)
                 (?x name ?v))
           :type '((?y {orders}) (?x {dealers}))
           :returns '(?v ?u))
query 6
(solve-all '(and (?x prod# ?y) (?x price ?z) (?x desc ?u))
           :type '((?x {products}))
           :returns '(?y ?z ?u))
```

Schema #2

```
query 1
(solve-all '({D260} phone ?x)
           :returns '?x)
query 2
(solve-all '(and (?x zip_code 1260) (?x dlrid ?y))
           :type '((?x {outlets}))
           :returns '?y)
query 3
(solve-all '(and (?x dlrid 1370) (?x oaddr ?y) (?x zip_code ?z))
           :type '((?x {outlets}))
           :returns '(?y ?z))
query 4
(solve-all '(and ({D150} dlrid ?w) (?x dlrid ?w) (?x zip_code ?y)
                 (?z zip ?y) (?z cratio ?u) (?z compid ?v))
           :type '((?x {outlets}) (?z {competitors}))
           :returns '(?u ?v))
query 5
(solve-all '(and (?x qty ?z) (and ^(>= ?z 10) ^(<= ?z 20))
                 (?x prod ?u) (?v prod# ?u) (?x dlrid ?y)
                 (?w dlrid ?y) (?w name ?w1) (?v desc ?v1))
           :type '((?x {orders}) (?v {products}) (?w {dealers}))
           :returns '(?w1 ?v1))
query 6
(solve-all '(and (?x prod# ?y) (?x price ?z) (?x desc ?u))
           :type '((?x {products}))
           :returns '(?y ?z ?u))
```

Schema #3

```
*** Queries 1,3,4,5 and 6 in this schema are of the same forms as
    those in schema #1.  Query 2 in this schema, on the other
    hand, is of the same form as that in schema #2. ***
```

References

[1986a] *HP 9000 Series 300 NMODE User's Guide.* Hewlett Packard Company, 1986.

[1986b] *HP-RL Reference Manual.* Hewlett Packard Laboratories, September 1986.

[Chamberlin *et al.*, 1981] D. Chamberlin *et al.* A history and evaluation of system R. *Communications of the ACM*, 24(10), October 1981.

[Fishman *et al.*, 1987] D. Fishman *et al.* Iris: an object-oriented dbms. *ACM Transactions on Office Information Systems*, 5(2), April 1987.

[Abarbnel and Williams, 1986] R. Abarbnel and M. Williams. A relational representation for knowledge bases. In *First International Conference on Expert Database Systems*, 1986.

[Chow, 1986] E. Chow. *Iris and HPRL.* Technical Report STL-TM-86-13, Hewlett Packard Laboratories, October 1986.

[Chow and Cate, 1986] E. Chow and H. Cate. *Performance Evaluation for IRIS Version 1.0.* Technical Report STL-TM-86-13, Hewlett Packard Laboratories, October 1986.

[Date, 1986] C. Date. *An Introduction to Database Systems.* Volume 1, Addison-Wesley, fourth edition, 1986.

[Fikes and Kehler, 1985] R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9), September 1985.

[Rich, 1983] E. Rich. *Artifical Intelligence.* McGraw-Hill, 1983.

[Rosenberg, 1983] S. Rosenberg. HPRL: a language for building expert systems. In *Proc. IJCAI*, 1983.

[Stefik, 1979] M. Stefik. An examination of a frame-structured representation system. In *Proc. IJCAI*, 1979.

[Warren, 1981] D. Warren. Efficient processing of interactive relational database queries expressed in logic. In *Proc. of 7th Int. Conf. on VLDB*, 1981.