(860821)

VLB ARRAY MEMO No. 569

CSIRO Division of Radiophysics

To: VLBA Memo Series

From: Martin Ewing 15 August 1986

Subj: Relational Databases and VLBA Operations, Revisited

In an earlier memo (Memo I, of 7 August 1986), I outlined some data "relations" that could be used in the operation of the VLBA. Here, I report on my initial contacts with the VAX/VMS "Rdb" database product.

I am new to the field of databases on big machines, but my perspective is probably similar to many other radio astronomy computer people. I hope my impressions will have some value in assessing the practicality and desirability of using Rdb. I have not looked at competing products, so my remarks probably apply to the class of relational database products, rather than specifically to Rdb.

General Impressions

One does not approach DEC's db products lightly. There is a fearsome collection of manuals and acronyms to deal with when you first wet your toes. Typical Fortran / Pascal / C scientific users will have had no contact with any of this. However, the documentation and help files are quite good, and there are even prepared demonstrations (similar to EDTCAI) to bring along the novice.

I doubt that the typical VLBA scientist or engineer will want to use the interactive interfaces (Datatrieve and RDO) in their full generality. The command structures are enormously powerful while still English-like, but they do not have "Macintosh" clarity. A user can not sit down with these systems productively without having quite a bit of previous experience.

DEC seems to intend that general data entry and queries be handled through various forms- and screen-management packages and/or user-written programs. The typical manual data entry process, for example, would be handled by a package that puts up predefined "fill-in-the-blanks" screens on VT100 terminals, tests user responses for various validity constraints, and stores values in the database.

The Rdb system can be used interactively by experienced people through Datatrieve and RDO. RDO, in particular, is probably the best way to set up database structures and to try out various command sequences. However, for production use it will often be best to use the program-callable entries of Rdb.

Rdb is supplied with language-sensitive precompilers for Fortran, Pascal, Cobol, etc. These translate embedded Rdb commands into references to Rdb entry points. You have the ability to interleave Fortran and Rdb commands quite flexibly. Thus it is straightforward to access the database from VLBA monitor and control programs, correlator programs, or from specialized engineering test programs. I suspect this would be the most common way to utilize Rdb in the VLBA. My test programs described below use this interface.

Applicability

In Memo I, I set out some relations that would describe the VLBA operations. Most kinds of data, it turns out, are relatively static: tape and user indexes, antenna configurations, etc. It is clear that a general database system like Rdb is quite useable for these.

One concern is whether Rdb is really called for: is the problem large enough? DEC's literature says that their smaller (and friendlier) Datatrieve system (which can be used with its own data structures separately from Rdb) is more appropriate for files with less than about 5,000 records. Most static VLBA files will be smaller than this size; only telescope instructions, logs, and soruce catalogs could be much larger. I would say that the total number of records in the VLBA database would likely be over the 5,000 mark and the extra flexibility of embedding Rdb commands in Fortran make it relatively attractive for this application. Presumably, Datatrieve would also be purchased to handle casual report generation. (It may actually be bundled with Rdb; I don't know.)

The biggest uncertainty from my viewpoint has been whether the performance level of Rdb or Datatrieve would be high enough so that any large fraction of the monitor and control data could be contained. To gain some quantitative feel, I have made a couple of small experiments on CSIRO Radiophysics VAX 11/750.

Tests

Test 1: Signal Tape Inventory. As a typical chore for Rdb, I selected the "Signal Tape Inventory" relation described in Memo I. The attached listings show the relation and field definitions from Rdb. I wrote a small Fortran program with Rdb calls, also attached, which would load the database with 1,000 records of "typical" data. The objective was to see what sort of resources would be required.

First, a note on the test computer. This is a VAX-11/751 (an OEM variant of the 750 purchased with an Intergraph CAD/CAM system) with 8 MB of memory and 12–15 users during the period of the tests. It is half of a VAX cluster sharing an Systems Industries disk sytem with Fujitsu Eagle drives, which contained the Rdb files. The 750's at Radiophysics (like most VAXes I know) are generally considered to be terribly overloaded.

The results:

a. Disk space. There is a disk space overhead of some 500 blocks per database. But, since one only needs one "VLBA" database to hold all the relations of Memo I, this overhead will not be serious. (Actually, for security one might place personnel or proposal review data in a separate database, but no matter.) The VLBA database, loaded with 1,000 tape records takes some 1,300 disk blocks (0.65 Mbyte), mostly in one file. The disk space consumption is broken down in an attached listing.

b. Memory. My experience is that Datatrieve, RDO, and their ilk always want as much main memory as is available. They would always run at 1,100–1,200 pages on our system. Small programs calling Rdb run at 900–1,000 pages.

c. Resources required to load 1,000 records:

	Total	Per Record
Elapsed Time:	737. s	0.74 s
CPU Time:	496. s	0.20 s
Buffered IO:	2,032.	2.0
Direct IO:	9,750.	9.8
Page Faults:	647.	–

Test 2: Bulk Monitor Data. To test the extreme monitor data capacity of an Rdb system, I used the "segmented stream" facility. This is a method for storing relatively large amounts of data (up to 64K bytes) in an Rdb field. Rdb carries the data as part of a normal record, but the contents of the stream segments is arbitrary binary or ASCII data. Rdb does not attempt to evaluate this data; it cannot be used directly for searching, etc. The user's program is completely responsible for the internal format.

I wrote a program to store 200 4K byte ASCII segments in a "monitor-test" relation. Each segmented stream consisted of two 4K segments, making 100 Rdb records in all.

The results:

1. Disk Space. The Rdb file expanded to nearly 3,300 blocks (1.7 Mbyte) after the data were loaded.

2. Other Resources. With 17 users on the system, the following statistics were recorded:

	LOADING		READING	
	Total	Per Record	Total	Per Record
Elapsed Time:	372. s	1.9 s	30. s	0.15 s
CPU Time:	67. s	0.34 s	11. s	0.055 s
Buffered IO:	208	1.0	7	0.035
Direct IO:	2,464	12.3	357	1.79
Page Faults:	731	–	377	–

Note that preallocation of disk file space is required to achieve this timing for loading. At first I had not preallocated enough storage, and the loading process took about twice the times indicated above.

Conclusions

I was somewhat surprised that the record loading time was only 0.2–0.3 CPU seconds on the VAX 11/751. For the VLBA monitor and logging data application, I would suspect that the normal processing time would be dominated by loading (writing) time. Engineering data would be retrieved in sporadic ways depending on the problem at hand. (I assume that data for real-time display would be creamed off the incoming stream before going into Rdb.) In terms of data rate, a VAX 750 could more-or-less keep up with a 96 Kbit/s (12 Kbyte/s) incoming rate, much greater than the expected rates, even including real-time fringe checking.

Rdb obviously requires more CPU time, disk activity, and disk space for storing data than a Fortran unformatted write statement and a corresponding "flat file." However, it does provide the benefit of great versatility in accessing the data once stored. I have not worked up any benchmarks for data inquiries, since these would be difficult to define and because I do not regard inquiries as being the likely bottleneck for Rdb. Nor have I looked carefully at disk space utilization.

In all, I feel that VMS Rdb (and possibly its competitors) should not be excluded from further consideration for VLBA database operations. I believe that it *could* be used for the whole VLBA database implementation–from antenna monitoring to payroll data. Whether a general database system *should* be adopted is a question for another forum.

RDO Database Listing

The following is a listing of an interactive session using RDO to list the structure of the VLBA test database. The underlined parts are typed by the user.

```
show version
Current version of RDO is:  Rdb/VMS V1.1-0
Underlying versions are:
Database with db_handle  V   in file  vlba
        Rdb/VMS V1.1-0
        RDB V1.0

show database
Database with db_handle  V   in file  vlba

show relations
User Relations in Database with db_handle  V
     MONITOR_TEST
     SIGNAL_TAPE_INVENTORY
     SIGNAL_TAPE_LOG
     VENDORS

show fields for monitor-test
 Fields for relation  MONITOR_TEST
     MON_ID                           signed longword scale  0
       based on global field  NUMERIC
     MON_TIME                         Date
```

```
       based on global field  DATE
     MON_DATA                         segmented string
                                        segment_length 4096
       based on global field  BIG_DATA

show fields for signal-tape-inventory
 Fields for relation  SIGNAL_TAPE_INVENTORY
     STAPE_ID                         text size is  10
     VENDOR_ID                        text size is  8
     PRODUCT                          text size is  32
       based on global field  TEXT32
     LENGTH                           signed longword scale  0
       based on global field  NUMERIC
     ACQUISITION_DATE                 Date
       based on global field  DATE
     N_SHIP_CYCLES                    signed longword scale  0
       based on global field  NUMERIC
     N_REC_PASSES                     signed longword scale  0
       based on global field  NUMERIC
     LAST_SHIP_DATE                   Date
       based on global field  DATE
     LAST_SHIP_ORIG                   text size is  6
       based on global field  STATION_ID
     LAST_SHIP_NO                     signed longword scale  0
       based on global field  NUMERIC
     LAST_SHIP_RCVD                   Date
       based on global field  DATE
     LAST_SHIP_DEST                   text size is  6
       based on global field  STATION_ID
     CUR_CONDITION                    text size is  8
     CUR_LOCATION                     text size is  6
       based on global field  STATION_ID

show fields for signal-tape-log
 Fields for relation  SIGNAL_TAPE_LOG
     STAPE_ID                         text size is  10
     START_TIME                       Date
       based on global field  DATE
     STOP_TIME                        Date
       based on global field  DATE
     QUALITY_LEVEL                    text size is  8

show fields for vendors
 Fields for relation  VENDORS
     VENDOR_ID                        text size is  8
     NAME                             text size is  32
       based on global field  TEXT32
     TELEPHONE                        text size is  17
       based on global field  PHONE_NO
     TELEX                            text size is  16
       based on global field  TELEX_NO
     ADDR_1                           text size is  20
       based on global field  ADDR_LINE
     ADDR_2                           text size is  20
```

```
       based on global field  ADDR_LINE
     ADDR_3                          text size is  20
       based on global field  ADDR_LINE
     ADDR_4                          text size is  20
       based on global field  ADDR_LINE
     ZIP                             text size is  10
     CONTACT                         text size is  32
       based on global field  TEXT32
```

analyze page

```
Space utilization analysis - completed at 14-AUG-1986 13:10:54.49
480 data pages, each page is 2 blocks long
Available data storage area is 78 percent utilized

-- %used --- #data pages -----------------------------------------------------

90 -100%        224 |========================================================
80 - 90%         97 |========================
70 - 80%         41 |==========
60 - 70%          0 |
50 - 60%          3 |
40 - 50%          6 |=
30 - 40%         54 |=============
20 - 30%          0 |
10 - 20%          1 |
 0 - 10%         54 |=============
```

analyze relations

```
                                                        Avg           %     %
                                                       Rec in   %  Total  Used
Record Name                     Occurrences Bytes Used  Bytes  Frag Space Space
-----------                     ----------- ---------- ------  ---- ----- -----
MONITOR_TEST                            100       2800     28    0    32    43
RDB$CONSTRAINTS                           0          0      0    0     0     0
RDB$CONSTRAINT_RELATIONS                  0          0      0    0     0     0
RDB$DATABASE                              1        597    597    0    11    17
RDB$FIELDS                               75      30225    403    0    74    90
RDB$FIELD_VERSIONS                      126      10332     82    0    59    73
RDB$INDEX_SEGMENTS                       22       1584     72    0    27    34
RDB$INDICES                              17       1615     95    0    28    35
RDB$RELATIONS                            14       1232     88    0    21    29
RDB$RELATION_FIELDS                     126      56826    451    0    88    94
RDB$VIEW_RELATIONS                        0          0      0    0     0     0
SIGNAL_TAPE_INVENTORY                  1000     125000    125    0    87    92
SIGNAL_TAPE_LOG                           0          0      0    0     0     0
VENDORS                                   2        408    204    0     7    13
                                ----------- ----------
                                       1483     230619
```

```
analyze indexes

                                  Index  Index   Length  Duplicate  Duplicate
Index Name                        levels nodes   used    nodes      length used
----------                        ------ -----   ------  ---------  -----------
RDB$CON_CONSTRAINT_NAME_NDX           1      1        0          0            0
RDB$CR_CONSTRAINT_NAME_NDX            1      1        0          0            0
RDB$CR_REL_NAME_NDX                   1      1        0          0            0
RDB$FIELDS_NAME_NDX                   2      9     2352          0            0
RDB$VER_REL_ID_VER_NDX                1      1      142         20         3868
RDB$NDX_SEG_NAM_FLD_POS_NDX           2      3      736          0            0
RDB$NDX_NDX_NAME_NDX                  2      3      547          0            0
RDB$NDX_REL_NAME_NDX                  1      1      345          6          552
RDB$REL_REL_ID_NDX                    1      1       86          0            0
RDB$REL_REL_NAME_NDX                  2      3      460          0            0
RDB$RFR_REL_FLD_NAMES_NDX             3     24     5606          0            0
RDB$RFR_SRC_FLD_NAME_NDX              2     10     2291         31         2852
RDB$VIEW_REL_NAME_NDX                 1      1        0          0            0
RDB$VIEW_VIEW_NAME_NDX                1      1        0          0            0
STAPE_ID_NDX                          2     18     6433          0            0
STAPE_LOCATION_NDX                    2     17     6343          0            0
VENDOR_INDEX                          1      1       27          0            0

show indexes
User Indexes in Database with db_handle  V

 Indexes for relation  SIGNAL_TAPE_INVENTORY
STAPE_ID_NDX                      with field  STAPE_ID
                                  No duplicates allowed
STAPE_LOCATION_NDX                with field  CUR_LOCATION
                                  No duplicates allowed

 Indexes for relation  VENDORS
VENDOR_INDEX                      with field  VENDOR_ID
                                  No duplicates allowed
```

Rdb Programming Examples

LOADER. This is the benchmark program used to load the signal-tape-inventory relation with "typical" data. It is used as input to RDBFOR, the Rdb Fortran precompiler. Lines beginning "&RDB&" are commands to the precompiler; they are nearly identical with RDO interactive commands.

```
        program loader
        implicit        none
        character*10    tapeno
        character*8     vendno
        character*32    product
        integer         length, nship, npass, waybillno
        real*8          adate, ldate, rcvddate
        character*8     origstn, deststn
        character*6     location
        character*8     condition
        integer         itrans, ntrans, iseed, ran3
        parameter       (ntrans = 1000)
        character*8     vendors(3), stations(3), conditions(3)
        character*32    products(3)
        character*80    err_message

        data    iseed/1234567/
        data    vendors/'Vendor A', 'Vendor B', 'Vendor C'/
        data    products/'Long slimy blue tape',
        1               'Lenghthy green sticky variety',
        2               'Slippery brown elastic snapping'/
        data    stations/'VLA NM', 'OWNVLY', 'SCROIX'/
        data    conditions/'Good', 'New', 'Poor'/

&RDB&   INVOKE DATABASE V = PATHNAME 'VLBA'
        open(unit=6, name='SYS$OUTPUT', status='NEW',carriagecontrol='LIST')
        write(6,*) 'Hello.'
        call lib$init_timer
        do itrans = 1, ntrans
&RDB&   START-TRANSACTION READ-WRITE
&RDB&           RESERVING SIGNAL-TAPE-INVENTORY FOR SHARED WRITE
        if (mod(itrans,100).eq.0) then
                write(6,10) itrans
10              format(' Record =', i5)
                call lib$show_timer
                end if
        write(tapeno,1) itrans
1       format('STAPE',i5.5)
        vendno = vendors(ran3(iseed))
        product = products(ran3(iseed))
        length = 5000 * ( ran(iseed) + 1.0)
        call sys$gettim(adate)
        nship = ran3(iseed)**2
        npass = nship*45
        origstn = stations(ran3(iseed))
        deststn = stations(ran3(iseed))
        ldate = adate
        rcvddate = adate
        waybillno = abs(1000*ran(iseed))
        condition = conditions(ran3(iseed))
        write(location,2) ntrans - itrans + 1
2       format('BN',i4.4)
&RDB&   STORE X IN SIGNAL-TAPE-INVENTORY USING
&RDB&    ON ERROR
                goto 999
&RDB&    END-ERROR
&RDB&   X.STAPE-ID = TAPENO;
&RDB&   X.VENDOR-ID= VENDNO;
&RDB&   X.PRODUCT  = PRODUCT;
&RDB&   X.LENGTH   = LENGTH;
&RDB&   X.ACQUISITION-DATE = ADATE;
&RDB&   X.N-SHIP-CYCLES    = NSHIP;
&RDB&   X.N-REC-PASSES     = NPASS;
```

```
&RDB&    X.LAST-SHIP-DATE    = LDATE;
&RDB&    X.LAST-SHIP-ORIG    = ORIGSTN;
&RDB&    X.LAST-SHIP-NO      = WAYBILLNO;
&RDB&    X.LAST-SHIP-RCVD    = RCVDDATE;
&RDB&    X.LAST-SHIP-DEST    = DESTSTN;
&RDB&    X.CUR-CONDITION     = CONDITION;
&RDB&    X.CUR-LOCATION      = LOCATION;
&RDB&    END-STORE
&RDB&    COMMIT
         end do
         call lib$show_timer
         call exit

999      call lib$sys_getmsg(%ref(rdb$status),
         1       ,%descr(err_message))
         type *,err_message
         type *, 'Error - rollback'
&RDB&    ROLLBACK
         end

         integer function ran3(iseed)
         implicit        none
         integer         iseed, i
         i = 10000*ran(iseed)
         ran3 = mod(i, 3) + 1
         return
         end
```

LSEG. This program was used to load the large segmented stream fields in the monitor-test relation.

```
         program lseg
         implicit        none
         real*8          adate
         integer         i, itrans, ntrans, irecord
         parameter       (ntrans = 100)
         character*80    err_message
         character*4096  text

&RDB&    INVOKE DATABASE V = PATHNAME 'VLBA'
         open(unit=6, name='SYS$OUTPUT', status='NEW',carriagecontrol='LIST')
         write(6,*) 'Hello.'
         call lib$init_timer
         do i = 1, 4096
                 text(i:i) = char(mod(i,128))
                 end do
         irecord = 0
         do itrans = 1, ntrans
&RDB&    START-TRANSACTION READ-WRITE
&RDB&            RESERVING MONITOR-TEST FOR EXCLUSIVE WRITE
         if (mod(itrans,100).eq.0) then
                 write(6,10) itrans
10               format(' Record =', i5)
```

```
                call lib$show_timer
                end if
        call sys$gettim(adate)
&RDB&   CREATE-SEGMENTED-STRING SSTR.
                irecord = irecord + 1
                write(text(1:5),5) irecord
5               format(i5.5)
&RDB&   STORE SEG IN SSTR USING
&RDB&           SEG.RDB$VALUE = text
&RDB&   END-STORE
                irecord = irecord + 1
                write(text(1:5),5) irecord
&RDB&   STORE SEG IN SSTR USING
&RDB&           SEG.RDB$VALUE = text
&RDB&   END-STORE
&RDB&   STORE X IN MONITOR-TEST USING
&RDB&    ON ERROR
                goto 999
&RDB&    END-ERROR
&RDB&           X.MON-ID = ITRANS;
&RDB&           X.MON-TIME = ADATE;
&RDB&           X.MON-DATA = SSTR;
&RDB&   END-STORE
&RDB&   END-SEGMENTED-STRING SSTR
&RDB&   COMMIT
        end do
        write(6,6) irecord
6       format(i5,' records written.')
        call lib$show_timer
        call exit

999     call lib$sys_getmsg(%ref(rdb$status),
        1       ,%descr(err_message))
        type *,err_message
        type *, 'Error - rollback'
&RDB&   ROLLBACK
        end
```

RSEG. This program reads back the data loaded by LSEG.

```
        program rseg
        implicit        none
        real*8          adate
        integer         i, itrans, ntrans, slen, id, is
        integer         iseg, irecord
        parameter       (ntrans = 100)
        character*80    err_message
        character*4096  sval

&RDB&   INVOKE DATABASE V = PATHNAME 'VLBA'
        open(unit=6, name='SYS$OUTPUT', status='NEW',carriagecontrol=
     +  'LIST')
        write(6,*) 'Hello.'
        call lib$init_timer
        i = 0
&RDB&   START-TRANSACTION READ-ONLY
&RDB&   START-STREAM SM USING X IN MONITOR-TEST
        do itrans=1,ntrans+1
&RDB&     FETCH SM
&RDB&           AT END
                        goto 888
&RDB&           END-FETCH
&RDB&   START-SEGMENTED-STRING SS USING T IN X.MON-DATA
        do iseg=1,2
&RDB&   GET slen = T.RDB$LENGTH;
&RDB&       sval = T.RDB$VALUE;
&RDB&       id = X.MON-ID;
&RDB&       adate = X.MON-TIME;
&RDB&   END-GET
                i = i + 1
                read(sval(1:5),11) irecord
11              format(i5)
                if (irecord.ne.i) type *, 'Nrec, val read:', i, irecord
        end do
&RDB&   END-SEGMENTED-STRING SS
        end do
&RDB&   END-STREAM SM
        type *, 'I fell out of the loop.'
888     type *, i, ' Records read.'
        call lib$show_timer
        end
```