Relationship Searches on Family Databases: Theory and Practice

PART II. THE PRACTICE

We now put to practice the theories of part I, using dBASEII (a trademark of Ashton-Tate Corp.). We must implement the theories efficiently. One implementation, soon to be shown, originally took 18 minutes to answer a query. After modifying it for better efficiency, the same query was reduced to 2 minutes.

We call our database FAMILY. The relevant fields, for the simpler queries, are all 4-digit numeric codes: PERSON, FATHER, MOTHER.

Person	Father	Mother
127	18	1037
128	347	92
129	2	212

However, our database may have any additional fields we desire: Name, Birthdate, Birthplace, etc.

Is X related to Y?

We used four steps to answer this question:

1. Create 2 sets: one with Martha Johnston in it and one with Solomon Butson in it.
2. Find the records of the parents of each person, and add them to that person's set.
3. Check to see if anyone in Martha Johnston's ancestral set is also in Solomon Butson's ancestral set. If so, quit: the answer is "Yes, they are related."
4. Repeat steps 2 and 3 until there are no more parents to be found. If that happens, then they have no known blood relationship.

The functions we need to carry out these steps are:

```
Answer Query.
  Step 1.
    Make an index.
    Determine who X and Y are.
    Make Ancestral Sets for X and Y.
  Step 2.
    Generate parents into Ancestral Sets.
  Steps 3 and 4.
    Check for Match in Ancestral Sets.
    Print Results.
```

That is, we will need computer routines, or modules, to do each of the six functions. Then, we can put those routines together to answer the query.

The main module, the one that puts the others together, is called RELATE. We code it in dBASEII as follows. [NOTE: Lower case words are variables. Upper case words are dBASEII reserved words. To save space, some long commands have been broken into multiple lines. These are identified by the one-space indention of the second line.

```
*******************************************
*              RELATE.CMD
*          "Is X related to Y?"
*******************************************
CLEAR
SET TALK off
SET ECHO off
SET FORMAT TO screen
STORE "Is X related to Y?" TO mode
*
DO makeindx
DO getxy
DO makesets
DO WHILE .NOT. related .AND.
 (xgenend   xprevend .OR. ygenend   yprevend)
    STORE gens + 1 TO gens
*
    IF xgenend   xprevend
        STORE "X" TO setname
        DO generate
    ENDIF
    IF ygenend   yprevend
        STORE "Y" TO setname
        DO generate
    ENDIF
*

    DO checkit
ENDDO
*
DO printans
RETURN
```

The six routines, or modules, are preceded by asterisks:

MAKEINDX
GETXY
MAKESETS
GENERATE
CHECKIT
PRINTANS

GENERATE and CHECKIT are in a DO/ENDDO loop. This loop repeats the two functions (recall step 4 of our theoretical procedure) until either a relationship is found or we run out of ancestors for both X and Y.

GENERATE runs only if no relation has yet been found and if there is the possibility of finding any more ancestors. That is, if X has run out of ancestors but Y has not, then there is no need to go looking for more ancestors of X. The variables XGENEND, XPREVEND, and SETNAME will be explained in detail below.

We will examine each of the six modules in detail in the following sections.

Module MAKEINDX

MAKEINDX creates a PERSON code index for our FAMILY database. We will use this index, called XPERSON, in other modules to make the searching efficient.

The following is the advanced version of MAKEINDX. The only commands needed to make the index are:

```
    USE family
    INDEX ON person TO xperson
```

The rest of the code in the advanced module gives the inquirer the option of bypassing the somewhat time consuming creation of an index, if he or she knows that a correct, up-to-date index already exists.

```
*******************************************
*               MAKEINDX.CMD
*******************************************
IF FILE("xperson.ndx")
    STORE f TO makeindx
    ERASE
    @  1,11 SAY "RELATIONSHIP SEARCH"
@  3, 1 SAY "-------------------------------"
@  5, 1 SAY "This search requires that an    "
@  6, 1 SAY "index be made for the FAMILY    "
@  7, 1 SAY "database. There is an index on "
@  8, 1 SAY "the diskette. Thus you have the"
@  9, 1 SAY "option "of using that index     "
@ 10, 1 SAY "(which may not be up-to-date)  "
@ 11, 1 SAY "or creating a new index (which "
@ 12, 1 SAY "is time-consuming).            "
@ 14, 1 SAY "Enter a 'y' to create an index."
    @ 14,33 GET makeindx
@ 20, 1 SAY "-------------------------------"
    READ
    IF makeindx
        USE family
        INDEX ON person TO xperson
    ENDIF
ELSE
    USE family
    INDEX ON person TO xperson
ENDIF
RELEASE makeindx
RETURN
```

Module GETXY

GETXY asks the inquirer who he or she wants to inquire about and then makes this available to the other modules as X and Y. The simplest version, shown below, merely asks you to enter X and then enter Y.

I have implemented a more advanced version that allows you to enter all or part of the first and/or last name. It then scans the database and gives you the data about everyone who met those requirements. You then tell it which one of the people it found is the one you want. Optionally, if you simply enter a number for person X, it tells you who that person is. And it let's you confirm that it is the person you want before it does the relationship search.

The advanced version is too complex for this article. Thus, the following is the simple version:

```
*******************************************
*               GETXY.CMD
*******************************************
STORE 0000 TO x
STORE 0000 TO y
ERASE
@  3, 1 SAY "Enter the numbers or names of  "
@  4, 1 SAY "the persons.                   "
@  5, 1 SAY "X"
@  5, 3 GET x
@  6, 1 SAY "Y"
@  6, 3 GET y
READ
RETURN
```

Module MAKESETS

MAKESETS creates the Ancestral Sets for X and Y. It initializes the sets by copying in the data (PERSON, FATHER, and MOTHER codes) about X and Y.

This is an important module for improving the efficiency of the overall process. Do not use the dBASEII COPY for setting the initial values in X and Y. COPY is not efficient for that. Use APPEND and REPLACE instead.

To gain even more efficiency, we limit the Ancestral Sets to only those fields needed to answer the query. If the person, father, and mother codes are the only fields needed to determine a relationship, why make the computer use the entire FAMILY database record, with names, dates, and places? So, we set up model databases, called RELATE1X and RELATE1Y, for the Ancestral Sets. Then we COPY the structures of the models to create our X and Y Ancestral Sets.

```
*******************************************
*               MAKESETS.CMD
*******************************************
STORE 0000 TO gens
STORE 0000 TO xprevend
STORE 0001 TO xgenend
STORE 0000 TO yprevend
STORE 0001 TO ygenend
STORE f TO related
*************************
USE family INDEX xperson
*
STORE STR(x,4) TO srcharg
FIND &srcharg
SELECT SECONDARY
USE relate1x
COPY STRUCTURE TO x
USE x
APPEND BLANK
REPLACE person WITH p.person
REPLACE father WITH p.father
REPLACE mother WITH p.mother
USE
SELECT PRIMARY
*
STORE STR(y,4) TO srcharg
FIND &srcharg
SELECT SECONDARY
USE relate1y
COPY STRUCTURE TO y
USE y
APPEND BLANK
REPLACE person WITH p.person
REPLACE father WITH p.father
REPLACE mother WITH p.mother
USE
SELECT PRIMARY
*************************
RELEASE recnum, srcharg
RETURN
```

Module GENERATE

GENERATE is the first module to be in the DO/ENDDO loop. It is repeated many times. So, we must look at it closely. Efficiency here will have a great effect on the efficiency of the overall query.

GENERATE finds the father's and mother's records, if there are any. It then adds these records to the Ancestral Sets. GENERATE also controls a record counter, which is very important. Let's take a closer look at the Ancestral Sets and see why this is important.

The Ancestral Set for X will have a record for X, followed by a record for his or her father, followed by a record for X's mother, etc. The Ancestral Set for X will look like this, assuming records for all relevant ancestors are actually in our FAMILY database. Recall that each record has three fields: PERSON, FATHER, and MOTHER codes.

Record	Description
1	Record for X
2	Record of Father of X
3	Record of Mother of X
4	Record of Father of 2
5	Record of Mother of 2
6	Record of Father of 3
7	Record of Mother of 3

Of course, not all ancestors for any person will be in our database. There will be some cases like this:

Record	Description
1	Record for X
2	Record of Mother of X
3	Record of Father of 2
4	Record of Father of 3
5	Record of Mother of 3

How do we keep track of where we are at any time? How do we know when we have run out of ancestors to add to an Ancestral Set? One way is to keep two record counters for each Ancestral Set. We have already seen these counters in the main module, RELATE. The first counter, XPREVEND for Set X, keeps track of the last record number of the previous generation. The other counter, XGENEND, keeps track of the absolute highest record number on the Ancestral Set.

For example, suppose we have the first Ancestral Set, with 7 records, shown above. We are ready to generate the next generation of ancestors of X. Module RELATE checks to see that we actually added some ancestors to the Set for the previous generation. It sees that XPREVEND was 3 (X, X's father and X's mother) and that XGENEND is now 7. Thus, we know that we did add some records for the last generation. So, we can now look for the parents of these latest added ancestors. (If we had not found any ancestors in our previous generation, both counters would be the same, 7.) GENERATE now takes over and does all the work for the next generation, including changing XPREVEND and, if any more ancestors are found, XGENEND.

GENERATE also makes use of dBASEII symbolic variables. We could have had a separate GENERATE module for X and one for Y. They would be the same, except for the references to the variables, counters, and Ancestral Sets. The use of the symbolic variable &SETNAME allows us to have just one GENERATE module. RELATE1 passes either an X or a Y into GENERATE via the SETNAME field. GENERATE does the rest.

```
*********************************************
*                GENERATE.CMD
*********************************************
STORE "&setname" + "genend"   TO genfld
STORE "&setname" + "prevend" TO prevfld
STORE &prevfld                      TO start
STORE &genfld                       TO finish
STORE &genfld                       TO &prevfld
USE &setname
SELECT secondary
USE family INDEX xperson
SELECT primary
DO WHILE start   finish
   STORE start + 1 TO start
************
   GOTO start
   IF father   0
      STORE STR(father,4) TO srcharg
      SELECT secondary
      FIND &srcharg
      IF #   0
         SELECT primary
         APPEND BLANK
         REPLACE person WITH s.person
         REPLACE father WITH s.father
         REPLACE mother WITH s.mother
         STORE &genfld + 1    TO &genfld
      ELSE
         SELECT primary
      ENDIF
   ENDIF
************
   GOTO start
   IF mother   0
      STORE STR(mother,4) TO srcharg
      SELECT secondary
      FIND &srcharg
      IF #   0
         SELECT primary
         APPEND BLANK
         REPLACE person WITH s.person
         REPLACE father WITH s.father
         REPLACE mother WITH s.mother
         STORE &genfld + 1    TO &genfld
      ELSE
         SELECT primary
      ENDIF
   ENDIF
ENDDO
SELECT secondary
USE
SELECT primary
USE
RELEASE start, finish, recnum, srcharg
RELEASE genfld, prevfld
RETURN
```

Module CHECKIT

CHECKIT is also in the DO/ENDDO loop and is thus repeated several times to answer the query. CHECKIT does a relational JOIN of Ancestral Sets X and Y, creating a new set Z, if there are any matches. That is, if - and only if - anyone is both an ancestor of X and an ancestor of Y, then that person will be included in Set Z, the Set of common ancestors of X and Y. To find out if there is a relationship, we simply look at Set Z and see if there are any records in it. If there are, then we have found a relationship between X and Y. So, we store the value t (for true) in the field called RELATED. Otherwise, we leave RELATED as it was.

```
*********************************************
*                CHECKIT.CMD
*********************************************
USE x
SELECT secondary
USE y
```

```
SELECT primary
IF FILE(z)
   DELETE FILE z
ENDIF
JOIN TO z FOR person=s.person
 FIELDS person,father,mother
SELECT secondary
USE
SELECT primary
USE z
IF #    0
   STORE t TO related
ENDIF
USE
RETURN
```

Module PRINTANS

PRINTANS looks at the value of the variable called RELATED. If the variable is 't' (for true), PRINTANS prints a message that there is a relation between X and Y. Otherwise, it says there was no relation found on the database. This is aother module that can have simple or complex versions. This version is simple.

```
*******************************************
*              PRINTANS.CMD
*******************************************
ERASE
?
?
STORE "Is          ("+STR(x,4)+") "+xname
 TO text1
? text1
STORE "related to ("+STR(y,4)+") "+yname+"?"
 TO text2
? text2
IF related
   STORE "                YES"
    TO text3
ELSE
   STORE "                 NO"
    TO text3
ENDIF
? text3
ERASE
RETURN
```

That does it for the simplest query, "Is X related to Y?" Now let's look at how to implement the more difficult queries.

Who Are the Common Ancestors of X and Y?

This query is now easy to implement. The only module that must change is PRINTANS. If there is a relation, PRINTANS must do more than print "YES".

For each person in Set Z, the common ancestors of X and Y, PRINTANS must FIND that person's record in our FAMILY database. Then PRINTANS must print the ancestor's name, which is in the FAMILY database record. That's all there is to the change.

What Is the Relationship of X to Y?

This one is not so easy. But, thanks to the modular nature of our program, it is easy to see what must be changed. Let's recall what our procedure was:

1. Create 2 sets: one with Martha Johnston in it and one with Solomon Butson in it. To each record add a field for counting the generation number. For Martha and Solomon, give this field a value of 0 (zero).

2. Find the records of the parents of each person, and add them to that person's set. In each record added, include the field for the generation number. Set the value to 1 more than the value of the field for the child.

3. Check to see if anyone in Martha Johnston's ancestral set is also in Solomon Butson's ancestral set. If so, compute and print their relationship and quit.

4. Repeat steps 2 and 3 until there are no more parents to be found. If that happens, then they have no known blood relationship.

What modules do not need changing? MAKEINDX and GETXY do not have anything to do with telling us what the relation is. Nor do they have anything to do with finding out what the relation is. So they can be used as they are.

PRINTANS, on the other hand, must obviously change. It has to tell us the name of the relationship and not just that there is one or who the common ancestors are. So, it will need to be a lot smarter. Recall the relationship table from our theoretical discussion in part I. PRINTANS must be able to use that table.

In order to use the table, PRINTANS must know how many generations X is from the common ancestor and how many generations Y is from the common ancestor. The most efficient way to keep track of that is to add a field, call it GENX for X and GENY for Y, to the Ancestral Sets. So, we must change our model databases. Call the new ones RELATE3X and RELATE3Y. And we must change any modules that use the Ancestral Sets.

MAKESETS, GENERATE, and CHECKIT must be changed to include references to the new generation count fields. MAKESETS must set the initial value of GENX and GENY to zero. GENERATE must store the new values of GENX and GENY. This will be for the parents, 2 for the grandparents, etc. And CHECKIT must include the two new fields in the FIELDS part of the JOIN command.

Here's how these three modules look after the changes. We'll consider PRINTANS in more detail below.

```
*******************************************
*              MAKESET3.CMD
*******************************************
STORE 0000 TO gens
STORE 0000 TO xprevend
STORE 0001 TO xgenend
STORE 0000 TO yprevend
STORE 0001 TO ygenend
STORE f TO related
************************
USE family INDEX xperson
STORE STR(x,4) TO srcharg
FIND &srcharg
SELECT SECONDARY
USE relate3x
COPY STRUCTURE TO x
USE x
APPEND BLANK
```

```
REPLACE person WITH p.person
REPLACE father WITH p.father
REPLACE mother WITH p.mother
REPLACE genx WITH 0
USE
SELECT PRIMARY
*
STORE STR(y,4) TO srcharg
FIND &srcharg
SELECT SECONDARY
USE relate3y
COPY STRUCTURE TO y
USE y
APPEND BLANK
REPLACE person WITH p.person
REPLACE father WITH p.father
REPLACE mother WITH p.mother
REPLACE geny WITH 0
USE
SELECT PRIMARY
************************
RELEASE recnum, srcharg
RETURN

*********************************************
*                  GEN3.CMD
*********************************************
STORE "gen"+"&setname"          TO gennum
STORE "&setname" + "genend"  TO genfld
STORE "&setname" + "prevend" TO prevfld
STORE &prevfld              TO start
STORE &genfld               TO finish
STORE &genfld               TO &prevfld
USE &setname
SELECT secondary
USE family INDEX xperson
SELECT primary
DO WHILE start   finish
    STORE start + 1 TO start
************
    GOTO start
    IF father   0
        STORE STR(father,4) TO srcharg
        SELECT secondary
        FIND &srcharg
        IF #    0
            SELECT primary
            APPEND BLANK
            REPLACE person  WITH s.person
            REPLACE father  WITH s.father
            REPLACE mother  WITH s.mother
            REPLACE &gennum WITH gens
            STORE &genfld + 1   TO &genfld
        ELSE
            SELECT primary
        ENDIF
    ENDIF
************
    GOTO start
    IF mother   0
        STORE STR(mother,4) TO srcharg
        SELECT secondary
        FIND &srcharg
        IF #    0
            SELECT primary
            APPEND BLANK
            REPLACE person  WITH s.person
            REPLACE father  WITH s.father
            REPLACE mother  WITH s.mother
            REPLACE &gennum WITH gens
            STORE &genfld + 1   TO &genfld
        ELSE
            SELECT primary
        ENDIF
    ENDIF
ENDDO
```

```
    SELECT secondary
    USE
    SELECT primary
    USE
    RELEASE start, finish, recnum, srcharg,gennum
    RELEASE genfld, prevfld
    RETURN

*********************************************
*                  CHECKIT3.CMD
*********************************************
USE x
SELECT secondary
USE y
SELECT primary
IF FILE(z)
    DELETE FILE z
ENDIF
JOIN TO z FOR person=s.person
 FIELDS person,father,mother,genx,s.geny
SELECT secondary
USE
SELECT primary
USE z
IF #    0
    STORE t TO related
ENDIF
USE
RETURN
```

PRINTANS will use a special module to figure out what the relationship is. It is that module, called RELATION that is of interest. The only real change to PRINTANS is to add a line that says

```
    DO relation
```

at the appropriate place.

So let's look at RELATION. It is going to be the biggest module we will have to deal with. It will look at GENX and GENY for the common ancestor we are working with. Using those two fields, it will print the relationship. To refresh our memories, let's look again at the relationship table.

X is the ... of Y.

		X		
Y	0	1	2	3
0	Self	Child	Grand Child	1st Great Grand Child
1	Parent	Full/Half Sibling	Niece/ Nephew	Grand Niece/ Nephew
2	Grand Parent	Aunt/ Uncle	1st Cousin	1st Cous, 1 Time Removed
3	1st Great Grand Parent	Grand Aunt/Uncl	1st Cous, 1 Time Removed	2nd Cousin

RELATION works a CASE at a time. It does the first CASE whose requirements it meets and skips the rest. There are five CASEs altogether.

First of all, if GENX = 0, then X is a direct ancestor of Y. So the first CASE group is for either GENX = 0 or GENY = 0.

The next CASE is when GENX and GENY are both 1. Look at the table. This means they are full or half siblings.

The next CASE is that one of them is 1, but not the other one. This is the Aunt / Uncle or Niece / Nephew situation. We need to figure out how many "Great"'s to put in front of it.

In the CASE where GENX and GENY are equal and greater than 1, we are on the table's diagonal. That means X and Y are "un-removed" cousins: 1st Cousins, 2nd Cousins, etc. All we have to do is calculate the cousin number, which is one less than the generation number.

Finally, there is the CASE where GENX and GENY are not equal but are both greater than 1. This is a computation to figure out how many times removed the cousins are, which is the absolute difference between GENX and GENY.

So, we have broken the problem into five subproblems. And now we have only mathematical computations, such as how many "Great"'s, how many times removed, or what number of cousin to calculate from GENX and GENY. We're ready to see how it looks in dBASEII.

```
*********************************************
*           RELATION.CMD
*  "What is the relationship of X to Y?"
*********************************************
DO CASE
*******************
 CASE genx = 0 .OR. geny = 0
   IF geny = 0
     STORE "Child" TO relation
     STORE genx-2     TO greats
   ELSE
     STORE "Parent" TO relation
     STORE geny-2     TO greats
   ENDIF
   IF greats   0
     STORE " " TO grand
   ELSE
     IF greats = 0
       STORE "Grand "  TO grand
     ELSE
       IF greats = 1
         STORE "Great Grand " TO grand
       ELSE
         IF greats   1
 STORE STR(greats,2)+" Great Grand " TO grand
         ENDIF
       ENDIF
     ENDIF
   ENDIF
   ? "is the",grand,relation
*******************
 CASE genx = 1 .AND. geny = 1
   ? "is the Full or Half Sibling"
*******************
 CASE genx = 1 .OR. geny = 1
   IF geny = 1
     STORE "Niece/nephew" TO relation
     STORE genx-3     TO greats
   ELSE
     STORE "Aunt/Uncle  " TO relation
     STORE geny-3     TO greats
   ENDIF
   IF greats   0
     STORE " " TO grand
   ELSE
     IF greats = 0
       STORE "Grand "  TO grand
     ELSE
       IF greats = 1
         STORE "Great Grand " TO grand
       ELSE
         IF greats   1
 STORE STR(greats,2)+" Great Grand " TO grand
         ENDIF
       ENDIF
     ENDIF
   ENDIF
   ? "is the",grand,relation
*******************
 CASE genx = geny
   STORE genx - 1 TO cousin
   ? "is the",STR(cousin,2),"Cousin"
*******************
 CASE genx   1 .AND. geny   1
   IF genx   geny
     STORE geny - 1    TO cousin
     STORE genx - geny TO removed
   ELSE
     STORE genx - 1    TO cousin
     STORE geny - genx TO removed
   ENDIF
? "is the",STR(cousin,2),"Cousin",
  STR(removed,3),"times removed"
 OTHERWISE
   ? "Program Bug in Relation.CMD--No Case"
   SET ECHO on
   SET STEP on
*******************
ENDCASE
RELEASE grand,greats,relation,cousin,removed
RETURN
```

It isn't so bad if you look at the logical CASEs. The only new (to our discussion) feature of dBASEII it uses is the STR function. STR converts numbers to character strings. We use it to save space on our printed answers. For example, STR(cousin,2) prints the number in the variable cousin as a 2-digit number.

Who Are the People Relating X to Y?

This one is the hardest of them all. In each record added to the Ancestral Sets, we must not only keep track of how many generations this ancestor is from X or Y. We must actually keep the complete chain of people linking X or Y to this ancestor. That means a change to our model databases to add a new field called CHAIN. So, we must change MAKESETS, GENERATE, and CHECKIT, since these all look at the fields of the Ancestral Sets. And obviously, we must change PRINTANS to print the result.

But there is really only one module that needs any significant change. After all, the only change to PRINTANS is to FIND the names of the common ancestors, using the CHAIN of person codes. And MAKESETS and CHECKIT only need minor changes to deal with the new CHAIN fields.

The real work is in GENERATE. It must add the links to the chain. But even this turns out not to be so bad. We add the following to the module:

```
STORE STR(person,4)+&chain TO genchain
```

This lengthens the CHAIN by putting the newest link on the front.

For example, suppose the chain for some person looks like this in Ancestral Set X:

```
2017013802220936
```

This means the person is 2017, who is the parent of 0138, who is the parent of 0222, who is the parent of 0936, who is X. When we add 2017's father 2209, his chain looks like this:

    22092017013802220936

Just to make sure you see the full effect on GENERATE, here is how the revised GENERATE will look:

```
*******************************************
*                GEN4.CMD
*******************************************
STORE "chain"+"&setname"       TO chain
STORE "gen"+"&setname"        TO gennum
STORE "&setname" + "genend"  TO genfld
STORE "&setname" + "prevend" TO prevfld
STORE &prevfld                TO start
STORE &genfld                 TO finish
STORE &genfld                 TO &prevfld
USE &setname
SELECT secondary
USE family INDEX xperson
SELECT primary
DO WHILE start   finish
    STORE start + 1 TO start
************
    GOTO start
    STORE STR(person,4)+&chain TO genchain
    IF father   0
        STORE STR(father,4) TO srcharg
        SELECT secondary
        FIND &srcharg
        IF #    0
            SELECT primary
            APPEND BLANK
            REPLACE person  WITH s.person
            REPLACE father  WITH s.father
            REPLACE mother  WITH s.mother
            REPLACE &gennum WITH gens
            REPLACE &chain   WITH genchain
            STORE &genfld + 1   TO &genfld
        ELSE
            SELECT primary
        ENDIF
    ENDIF
************
    GOTO start
    IF mother   0
        STORE STR(mother,4) TO srcharg
        SELECT secondary
        FIND &srcharg
        IF #    0
            SELECT primary
            APPEND BLANK
            REPLACE person WITH s.person
            REPLACE father WITH s.father
            REPLACE mother WITH s.mother
            REPLACE &gennum WITH gens
            REPLACE &chain   WITH genchain
            STORE &genfld + 1   TO &genfld
        ELSE
            SELECT primary
        ENDIF
    ENDIF
ENDDO
SELECT secondary
USE
SELECT primary
USE
RELEASE genfld, prevfld, start, finish
RELEASE recnum, srcharg,gennum,genchain,chain
RETURN
```

Other Relationships Beyond the Closest

Recall that to find relationships beyond the closest, we remove the common ancestors from the Ancestral Sets and resume the search. This requires no changes to the six main sub-modules.

Rather it means a change to the main module, RELATE, that DOes the sub-modules. We must put GENERATE and CHECKIT's DO/ENDDO loop into another bigger DO/ENDDO loop, along with PRINTANS and a new module. The big loop will be left only when there are no more ancestors for both X and Y or when the inquirer indicates he or she is ready to stop searching for more distant relationships.

The new module, we'll call it REMOVAL, takes those people who are in Set Z, the set of common ancestors, out of sets X and Y. The only exception is that X and Y must remain in their respective Ancestral Sets. Since REMOVAL and the changes to RELATE are fairly straightforward, their details are left to the reader.

CONCLUSION

Finding relationships is not a simple matter. But it is a process which can be efficiently run by a computer to save us time. Even if you do not want to program a relationship search, you should better understand what is involved in such a function.

We have explored the theory of finding blood relationships in a family database. We have put that theory into practice. We have seen to it that the implementation was efficient, as well as correct. We have implemented it in a modular manner, so that it can be most easily understood and modified.

There are issues remaining however. The most significant is the (apparently) far more complex problem of non-blood relationships. A little work with pencil and paper will show how quickly this problem explodes into a large one. But now we have a good base from which to deal with the remaining issues.

FAMILY TREE

A software package to help you use the Commodore 64 or VIC 20 (32K) as a dynamic system to control data on your family tree.

FEATURES

- *Unlimited Genealogies*
- *Fully indexed*
- *Easy editing and updating*
- *Search functions*
- *Family Record sheets*
- *4, 5, 6 Generation pedigrees*
- *Output to screen or printer*
- *Complete manual*

$49.95 CND — $39.95 US
(Ontario and Michigan residents add sales tax)

GENEALOGY SOFTWARE
phone 519-344-3990

P.O. Box 1151
Port Huron, MI
48061

1046 Parkwood Ave.
Sarnia, Ont.
N7V 3T9