

Case study 12

Kings of France - Part 2

Objective. In this chapter, we continue the exploitation of the KINGS database through more advanced tree processing applications, based notably on recursive scripts. The central concept from which most of these applications will derive is the transitive closure of table BRANCH, which comprises all direct and indirect ancestor/descendant couples. From it, we will build queries that count the descendants of a member, others that display the hierarchy of these descendants in various graphical way and a transitive reduction query that recovers the contents of table MEMBER from its closure. The last application, tree projection, extracts from table MEMBER a subset in which only kings appear.

Keywords. genealogy, tree, cyclic data structure, transitive closure, transitive reduction, tree projection, recursive CTE, recursive query, tree drawing, tree traversal, depth-first traversal, breadth-first traversal, SQLdraw.

12.1 The *ancestor/descendant* relationships

We will first build a very simple data set that tells us who is a descendant of whom. We know from the base data that ROBERT II is a descendant of HUGUES CAPET (actually his son) and that HENRI I is a descendant of ROBERT II. It is up to us to infer that HENRI I is a descendant of HUGUES CAPET (actually his grandson). We

can go on by identifying all the descendants of HUGUES CAPET, then those of ROBERT II, and so on (Figure 12.1).

The full set of these ancestor/descendant relationships is called the **transitive closure** of table MEMBER. More precisely, we should call it *the transitive closure of the set of (Father,PiD) couples extracted from MEMBER table*¹. Drawing a transitive closure may prove particularly painful, even for a small initial set of couples. For instance, that of MEMBER comprises 850 couples!

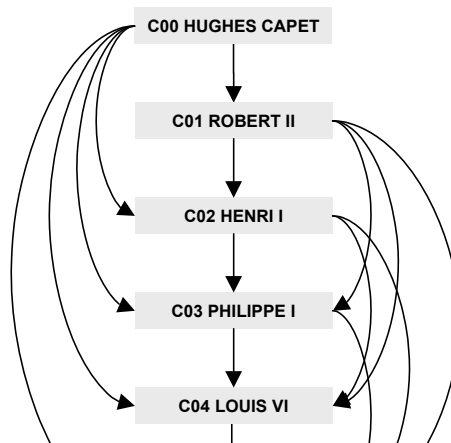


Figure 12.1 - Transitive closure: who is a descendant of whom?

Ancestor	Descendant
B00	B01
B00	B02
...	...
C00	C01
C00	C02
C00	C03
C00	C04
...	...
C01	C02
C01	C03
C01	C04
...	...
C02	C03
C02	C04
...	...
V14	V17

Figure 12.2 - The transitive closure of table MEMBER (excerpts)

1. In graph theory, the *transitive closure* **T** of a *binary relation* **B** (that is, a set of couples) is the set defined as follows: (1) it comprises all the couples of **B**, (2) if couples (a,b) and (b,c) are in **T**, then couple (a,c) also is in **T**.

It would be nice to *automatically* generate all these ancestor/descendant relationships from the base data recorded in table MEMBER. The result would look like the data table of Figure 12.2.

This is a good job for a recursive query, or, more precisely, a recursive CTE. In Script 12.1, CTE **PATHS** is defined by two queries:

1. the **initialization query**, that stores the couples (Father,PiD) of the members, except of those who have no father:

```
select Father,PiD from MEMBER where Father is not null
```

2. the **recursive query**, that adds to the CTE an additional subset, defined by joining the current contents of the CTE with the (Father,PiD) couples from table MEMBER:

```
select P.Ancestor, M.PiD
from   PATHS P, MEMBER M
where  P.Descendant = M.Father
```

To make the query easy to read, we have stored the definition of the CTE in variable Closure. The main query of the CTE prints all the couples of the closure.

```
set Closure = select Father,PiD from MEMBER
              where  Father is not null
              union all
              select P.Ancestor, M.PiD
              from   PATHS P, MEMBER M
              where  P.Descendant = M.Father;

with recursive PATHS(Ancestor,Descendant) as ($Closure$)
select Ancestor,Descendant
from   PATHS
order by Ancestor,Descendant;
```

Script 12.1 - Generating the closure of table MEMBER

Despite its simplicity, the closure is a powerful concept that can be used to solve in a simple way a great variety of problems.

Before going on, we will refine it a little bit by adding two derived pieces of information: the list of members forming the **path** between the ancestor and the descendant of each couple of the closure, and the **length** of this path. For instance, the path of couple (C01,C04) is .C01.C02.C03.C04. and its length is 3.

The extended version is shown in Figure 12.3 and is generated by Script 12.2. The path is built with the technique elaborated in Chapter 19 of the Tutorial, each Id being surrounded by commas.

Ancestor	Descendant	Len	Path
B00	B01	1	.B00.B01.
B00	B02	2	.B00.B01.B02.
...
C00	C01	1	.C00.C01.
C00	C02	2	.C00.C01.C02.
C00	C03	3	.C00.C01.C02.C03.
C00	C04	4	.C00.C01.C02.C03.C04.
...
C01	C02	1	.C01.C02.
C01	C03	2	.C01.C02.C03.
C01	C04	3	.C01.C02.C03.C04.
...
C02	C03	1	.C02.C03.C04.
C02	C04	2	.C02.C03.C04.
...
V14	V17	1	.V14.V17.

Figure 12.3 - The extended transitive closure of table MEMBER (excerpts)

```

set Closure = select Father,PiD,1,'.'||Father||'.'||PiD||'.'
               from MEMBER where Father is not null
               union all
               select P.Ancestor,M.PiD,P.Len+1,
                      P.Path||M.PiD||'.'
               from CTE_PATHS P,MEMBER M
               where P.Descendant = M.Father;

with recursive
  PATHS (Ancestor,Descendant,Len,Path) as ($Closure$)
select Ancestor,Descendant,Len,Path
from PATHS order by Len,Ancestor,Descendant;

```

Script 12.2 - Extending the closure of table MEMBER - CTE-based

Since we will use this CTE in several queries, it would be a good idea to transform it into a real SQL view.² We change the name of the CTE for CTE_PATHS so that the view can be named PATHS (Script 12.3).

To illustrate the exploitation of the closure, we suggest three application queries. Script 12.4 checks if a member is an ancestor of another one.

2. Or in a temporary table with appropriate indexes, if run time matters.

```

set Closure = <cftr. Script 12.2>;

create temporary view PATHS as
  with recursive
    CTE_PATHS (Ancestor,Descendant,Len,Path) as ($Closure$)
  select Ancestor,Descendant,Len,Path
  from CTE_PATHS;

```

Script 12.3 - Transformation of the CTE into an actual SQL view

```

ask A,B = Ancestor:|Descendant:;
extract C = select count(*) from PATHS
           where Ancestor = '$A$' and Descendant = '$B$';
if ($C$ = 1) showMessage $A$ is an ancestor of $B$;

```

```

ask A,B = Ancestor:|Descendant:;
select case when count(*) = 1
           then 'Yes, $A$ is an ancestor of $B$'
           else 'No, $A$ is not an ancestor of $B$'
        end as "Is $A$ an ancestor of $B$?"
from PATHS
where Ancestor = '$A$' and Descendant = '$B$';

```

Script 12.4 - Two scripts that check if member A is an ancestor of member B

Script 12.5 displays all the grandfather/grandson couples from the hierarchy.

```

select Ancestor,Descendant from PATHS where Len = 2;

```

Script 12.5 - Grandfather-grandson relationships

Finally, Script 12.6 checks whether member B is between member A and member C in the same branch without being any of them. The last condition examines whether the intermediate member (B) appears in the path from A to C. Both the first and the last characters (dots) are excluded from the search to prevent B from being found at the head or at the end of the path.

We could also check whether some paths include A, B and C in this order:

```

path like '%.$A$%.$B$%.$C$%'

```

```
ask A,B,C = A:|B:|C:;
extract N = select count(*) from PATHS
           where Ancestor = '$A$' and Descendant = '$C$'
           and Path like '.*$B$.*';
write $N$;
```

Script 12.6 - Are members A, B and C in the same branch?

12.2 Loop-based computing of the transitive closure

Loop-based algorithms can be less elegant than recursive queries but they allow more complex processing on each row of the resultset. They may also be preferred by programmers less familiar with recursive CTE.

Table PATHS, that will contain the paths to build, is initialized with the *Father-Son* couples extracted from table MEMBER. The next set of paths, built by a join between table PATHS and MEMBER, will be first stored in working table LAST, then moved table PATHS. Tables PATHS and LAST have the same structure:

```
PATHS (Ancestor, Descendant, Length, Path);
LAST  (Ancestor, Descendant, Length, Path);
```

However, this join is not applied to the full contents of PATHS, but only to the subset just added by the previous iteration (otherwise, the loop would run forever). This is controlled by variable length: initialization stores paths of length 1, the first iteration of the loop adds paths of length 2, the next iteration adds paths the length 3, and so on until all the *Father-Son* couples of table MEMBER have been exhausted, that is, when table LAST is empty.

Based on these principles, we build Script 12.7, which extracts the same data as Script 12.2. It is worth noticing that this iterative algorithm is quite similar to the internal mechanism of recursive CTE, though likely to be less efficient.

12.3 Counting descendants

Through view PATHS created by Script 12.3 one can easily count the descendants of each member:

```
select Ancestor, count(*) as Descent
from PATHS
group by Ancestor
order by Descent desc, Ancestor;
```

```

insert into PATHS select Father,PiD,1,
                        '.'||Father||'.'||PiD||'.'
                        from MEMBER where Father is not null;
for length = [1,99];
  insert into LAST
  select P.Ancestor, M.PiD, $length$+1, P.Path||'.'||M.PiD
  from PATHS P, MEMBER M
  where P.Descendant = M.Father
  and P.Length = $length$;
  extract lastN = select count(*) from LAST;
  if ($lastN$ = 0) exit;
  insert into PATHS select * from LAST;
  delete from LAST;
endfor;

```

Script 12.7 - Extending the closure of table MEMBER - Loop-based

To provide a complete answer, we also have to add, through the **union** operator, members who have no descendants:

```

select PiD as Ancestor, 0 as Descent
from MEMBER
where PiD not in (select Ancestor from PATHS)

```

This query (lines [2-9] in Script 12.8) returns a minimalist two column resultset that is not particularly attractive. We will improve it by adding the name of the member and by inserting the data into a nice sentence format like that of Figure 12.4.

```

+-----+
| [C00] HUGHES CAPET, King of France, has 35 royal descendants |
| [C01] ROBERT II, King of France, has 34 royal descendants    |
| [C02] HENRI I, King of France, has 33 royal descendants      |
| [C03] PHILIPPE I, King of France, has 32 royal descendants   |
| ...                                                         |
| [V02] JEAN II, King of France, has 11 royal descendants      |
| [V03] CHARLES V, King of France, has 10 royal descendants    |
| [B09] HENRI IV, King of France, has 7 royal descendants      |
| [B10] LOUIS XIII, King of France, has 6 royal descendants    |
| [B11] LOUIS XIV, King of France, has 4 royal descendants     |
| [C10] PHILIPPE IV, King of France, has 4 royal descendants   |
| [V13] FRANCOIS I, King of France, has 4 royal descendants    |
| ...                                                         |
| [V16] CHARLES IX, King of France, has 0 royal descendants   |
| [V17] HENRI III, King of France, has 0 royal descendants     |
+-----+

```

Figure 12.4 - An expressive presentation of the data

In addition, we will consider kings only: we count the *kings* in the descent of each *king* of France. Hence the query of Script 12.8, which deserves a bit of explanation

This query is built as a join of view KING [1] with the former expression [2-9] to extract additional information and to limit the members to those who are kings. Counting kings only in their descent is obtained by condition [4]. So, both ancestors and descendants are kings of France.

```

select ' [|||Ancestor|||] ' |||K.Name|||', King of France, has
      ' |||Descent|||' royal descendants'
from KING K,
      (select Ancestor, count(*) as Descent
       from PATHS
       where Descendant in (select PiD from KING)
       group by Ancestor
       union
       select PiD as Ancestor, 0 as Descent
       from MEMBER
       where PiD not in (select Ancestor from PATHS)) DC
where DC.Ancestor = K.PiD
order by Descent desc, Ancestor;

```

Script 12.8 - Generating the description of kings with the number of descendants

12.4 Showing the descendants of a member

In the next project, we will produce an **indented list** of the **descendants** of a member. Each son is displayed with a *two space* right shift + a dash character to make the hierarchy explicit. We would like something like the list of Figure 12.5.

Descent of LOUIS IX, roi de France:

```

- PHILIPPE III, Roi de France
- PHILIPPE IV, Roi de France
  - LOUIS X, Roi de France
    - JEAN I, Roi de France
  - PHILIPPE V, Roi de France
  - CHARLES IV, Roi de France
- Charles, Comte de Valois
- PHILIPPE VI, Roi de France
  - JEAN II, Roi de France
    - CHARLES V, Roi de France
      - CHARLES VI, Roi de France
        - CHARLES VII, Roi de France
          - LOUIS XI, Roi de France
            - CHARLES VIII, Roi de France
  - Louis I, Duc d'Orléans
    - Charles, Duc d'Orléans
      - LOUIS XII, Roi de France
    - Jean, Comte d'Angoulême

```



```

- Charles, Comte d'Angoulême
  - FRANCOIS I, Roi de France
    - HENRI II, Roi de France
      - FRANCOIS II, Roi de France
      - CHARLES IX, Roi de France
      - HENRI III, Roi de France
- Robert, Comte de Clermont
  - Louis I, Duc de Bourbon
    - Jacques I, Comte de La Marche
      - Jean I, Comte de La Marche
        - Louis, Comte de Vendôme
          - Jean VIII, Comte de Vendôme
            - François, Comte de Vendôme
              - Charles, Duc de Vendôme
                - Antoine, Duc de Vendôme
                  - HENRI IV, Roi de France
                    - LOUIS XIII, Roi de France
                      - LOUIS XIV, Roi de France
                        - Louis, Grand Dauphin
                          - Louis, Petit Dauphin
                            - LOUIS XV, Roi de France
                              - Louis, Dauphin de France
                                - LOUIS XVI, Roi de France
                                - LOUIS XVIII, Roi de France
                                - CHARLES X, Roi de France
                              - Philippe I, Duc d'Orléans
                                - Philippe II, Duc d'Orléans
                                  - Louis, Duc d'Orléans
                                    - Louis Philippe I, Duc d'Orléans
                                      - Philippe Égalité, Duc d'Orléans
                                        - LOUIS PHILIPPE I, Roi de France

```

Figure 12.5 - Indented list of the descent of LOUIS IX

The members are listed according to the *depth first traversal* described in Section 19.6 (*Recursive programming*) of the Tutorial, from which we derive Script 12.9.

```

ask pid = Member:[!select PiD from MEMBER order by PiD];
extract Nam,Tit = select Name,Title
                  from MEMBER where PiD = '$pid$';

write-a Descent of $Nam$, $Tit$;

with recursive TREE(Ind,Father,PiD,Path) as
(select ' ','$pid$',PiD,'.' || '$pid$' || '.' || PiD || '.'
 from MEMBER where Father = '$pid$'
 union
 select T.Ind || ' ',T.PiD,M.PiD,T.Path || M.PiD || '.'
 from TREE T, MEMBER M
 where T.PiD = M.Father)
select Ind || '- ' || M.Name || ', ' || M.Title as ""
from TREE T, MEMBER M
where T.PiD = M.PiD
order by T.Path;

```

Script 12.9 - Extracting the descent of a member through a recursive query

Alternative computing through a loop-based script

The scripts 12.10 use another technique for generating this hierarchical text: a loop scans the resultset of the CTE described above and writes, for each of its row, a line of the tree.

```
for txt =
[with recursive TREE(Ind,Father,PiD,Path) as
(select '', '$pid$', PiD, '.' || '$pid$' || '.' || PiD || '.'
from MEMBER where Father = '$pid$'
union
select T.Ind || ' ', T.PiD, M.PiD, T.Path || M.PiD || '.'
from TREE T, MEMBER M
where T.PiD = M.Father
)
select Ind || '- ' || M.Name || ', ' || M.Title as ""
from TREE T, MEMBER M
where T.PiD = M.PiD
order by T.Path
];
write @s$txt$;
endfor;
```

Script 12.10 - Extracting the descent of a member through a loop-based script

12.5 Graphical representation of king genealogy

Instead of merely printing member data as a pure text as we did in the previous section, it would be nicer to **draw** the tree as we may find it in some web sites.

We can use the SQLdraw engine to present the data in a graphical way. We first generate an SQLdraw script file representing the tree, then we render it with the `showDrawing` statement.

The detail of the generation of the SQLdraw file is described in complement document `SQLfast-Case-Kings-of-France-Draw.pdf` available in the script directory of this case study.

Figure 12.6 shows a simple representation of the *Father-son* tree of the members. In Figure 12.7, an improved drawing allows a better understanding of the tree structure. In addition, king descriptors are written in blue, which is the natural color of (the blood of) kings of France!

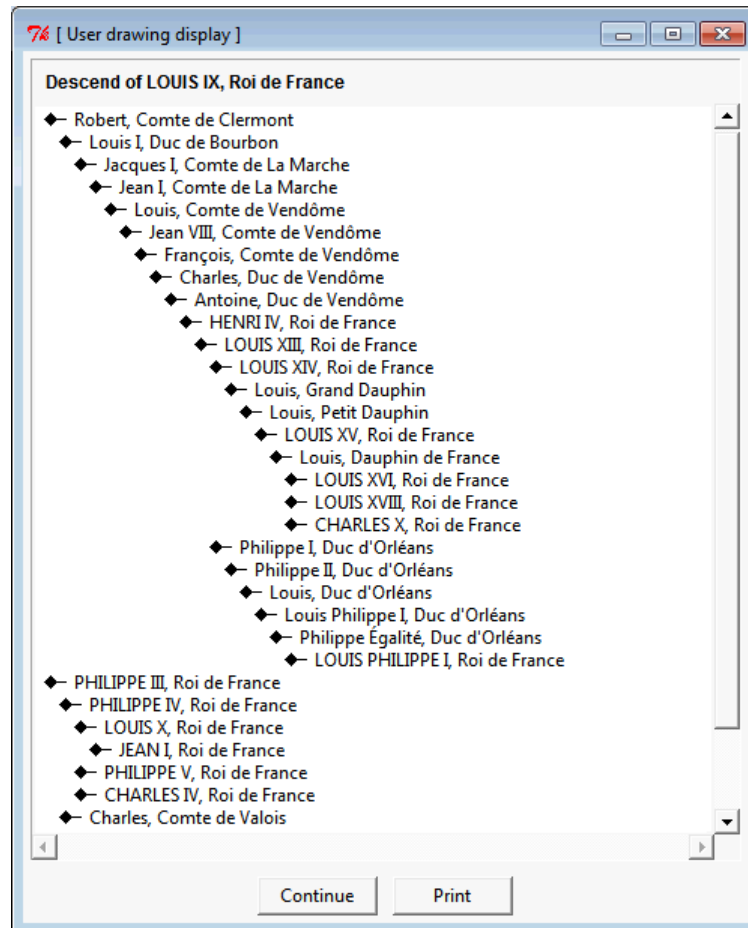


Figure 12.6 - The descent of King LOUIS IX (excerpt) - Simple version

12.6 Recovering the source data from their closure

This is a basic exercise that seems to have no immediate practical interest, but that will prove quite useful for the project discussed in the next section. The problem can be formulated as:

how can we extract from the 850 rows of the closure of table MEMBER the source data recorded in MEMBER?

More precisely, we would like to select the couples (Father,PiD) of the closure that also belong to table MEMBER. This operation, that removes all the couples that can be built by composing two other couples, is called **transitive reduction**.

Printed 5/6/23



1. If the closure is complemented with the length information (column **Len**), the selected couples satisfy condition ($Len = 1$), this is clearly the simplest and most efficient procedure.
2. A couple (M1,M3) of the closure is selected if there is **no** member M2, different from M1 and M3, such that couples (M1,M2) and (M2,M3) are in the closure.
3. A couple (M1,M3) of the closure is selected if there is **no** pair of couples (Ma,Mb) and (Mc,Md) in the closure such that $M1 = Ma$, $Mb = Mc$ and $Md = M3$.

Printed 5/6/23

```

set Closure = select PiD,PiD from MEMBER
              union all
              select P.Ancestor, M.PiD
              from   CTE_PATHS P, MEMBER M
              where  P.Descendant = M.Father;

create view PATHS as
with recursive CTE_PATHS (Ancestor, Descendant) as ($Closure$)
select Ancestor, Descendant from CTE_PATHS
where Ancestor <> Descendant;

select P.Ancestor, P.Descendant                                [1]
from   PATHS P
where  not exists (select 1 from PATHS Pab, PATHS Pcd
                  where  P.Ancestor = Pab.Ancestor
                  and     Pab.Descendant = Pcd.Ancestor
                  and     Pcd.Descendant = P.Descendant)
order by P.Ancestor, P.Descendant;

```

Script 12.11 - Transitive reduction of a closure: retrieving the source Father/Son relationships

12.7 Extracting the hierarchy of kings

This project is more concrete and can find applications in various actual situations. The goal is to extract from table MEMBER a subtree that comprises *all the kings and only them*. All the members who are not king do not appear in the hierarchy. Figure 12.8 shows what we want to extract. The links no longer represent the *father/son* relationships but rather the *closest king ancestor/king* relationships.

Selecting the kings among the members is easy, but combining the *father/son* links between them to produce *ancestor/descendant* links seems a bit more tricky. When *non-king* members appear between two kings, they are skipped and these kings are connected by a direct link.

Let us first observe that the *ancestor/descendant* links we are looking for are in the closure of MEMBER. Moreover, if we select in this closure the couples of two kings, we get the links we are looking for, but also those links that are combinations of the latter. In short, the full set of the *ancestor/descendant* links form the *closure* of the desired set of couples.

We consider the example of path **.B11.B12.B13.B14.B15.B16.**, illustrated in Figure 12.9. This path includes three kings, namely LOUIS XIV (B11), LOUIS XV (B14) and LOUIS XVI (B16).

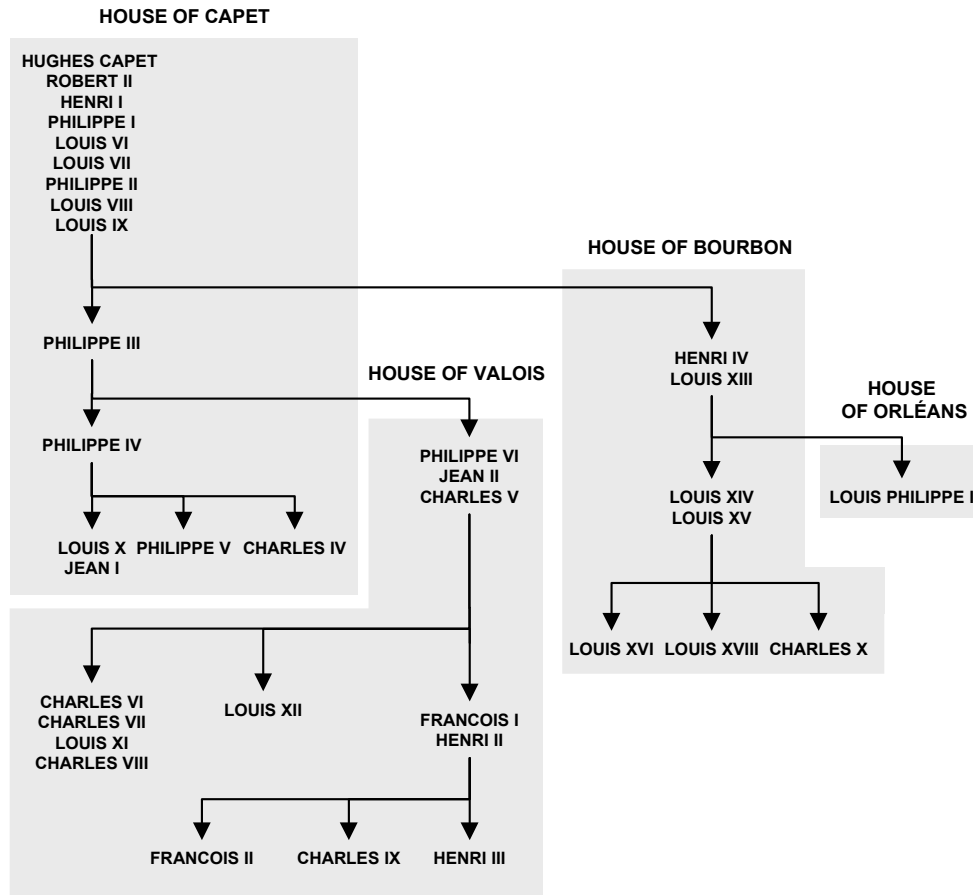


Figure 12.8 - The hierarchy reduced to the kings

The left side schema shows the fragment of the closure of MEMBER between B11 and B16. The central schema shows the kings and the links connecting them. The right side schema is the subset of these links we are looking for. We call the result the **projection** of MEMBER on its king members.

These observations provide us with a means to extract what we want:

- We select KK, the set of the couples of the closure of MEMBER in which both members are kings (central schema of Figure 12.9).
- We extract from KK the couples that cannot be built by composing two other couples (left side schema of Figure 12.9). In other words, we extract the *transitive reduction* of KK, a problem we have addressed in the previous section.

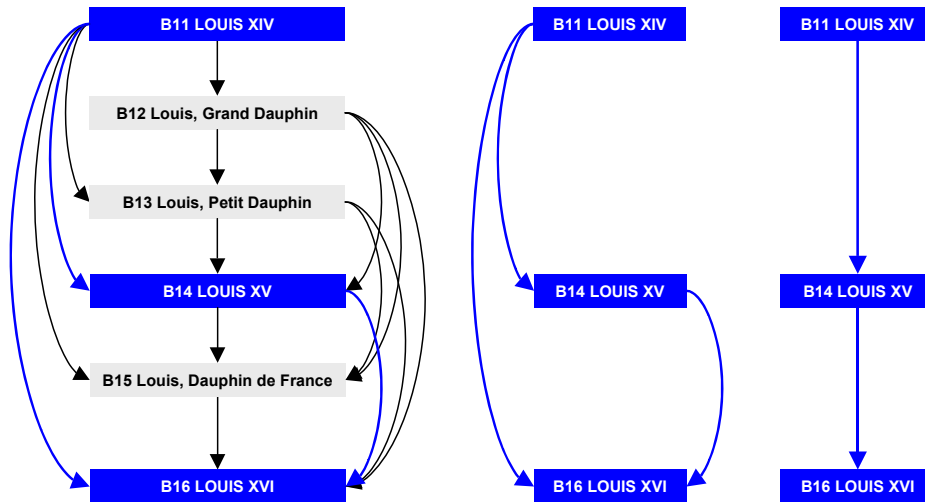


Figure 12.9 - Fragment of the closure of MEMBER (left), of its projection on kings (center) and of the transitive reduction (right)

Once again, there are several ways to solve the transitive reduction problem, each based on properties such as the following:

1. A couple (K1,K3) of KK, with path P13, is selected if there is **no** king K2, different from K1 and K3, that appears in P13. This is the simplest and the more efficient procedure, but it requires the recording of the paths.
2. A couple (K1,K3) of KK is selected if there is **no** pairs of couples (Ka,Kb) and (Kc,Kd) in the closure of MEMBER such that K1 = Ka, Kb = Kc and Kd = K3.
3. A couple (K1,K3) of KK is selected if there is **no** king K2, different from K1 and K3, such that couples (K1,K2) and (K2,K3) are in the closure of MEMBER.

Script 12.12 is based on the first idea. The main query joins each component of the couples of PATHS with its respective MEMBER [1,3]. Condition [2] discards the initialization couples. Conditions [4] force the elements of the couples to be kings. Condition [5] discards the couples, the path of which includes another king.

Figure 12.10 shows the result of the execution of this script while Figure 12.11 presents this information in a graphical way.

```

set Closure = select PiD,PiD,'.'||PiD||'.' from MEMBER
              union all
              select P.Ancestor, M.PiD, P.Path||M.PiD||'.'
              from   PATHS P, MEMBER M
              where  P.Descendant = M.Father;

with recursive PATHS(Ancestor,Descendant,Path) as ($Closure$)
select P.Ancestor,K1.Name,P.Descendant,K3.Name,P.Path
from   MEMBER K1, PATHS P, MEMBER K3
where  P.Ancestor <> P.Descendant
and    P.Ancestor = K1.PiD and P.Descendant = K3.PiD
and    K1.Title = 'Roi de France'
and    K3.Title = 'Roi de France'
and not exists (select count(*) from KING K3
               where P.Path like '%.'||K3.PiD||'._%')
order by P.Ancestor,P.Descendant;

```

Script 12.12 - Extracting the royal hierarchy

Ancestor	Name	Descendant	Name	Path
B09	HENRI IV	B10	LOUIS XIII	.B09.B10.
B10	LOUIS XIII	B11	LOUIS XIV	.B10.B11.
...				
B11	LOUIS XIV	B14	LOUIS XV	.B11.B12.B13.B14.
B14	LOUIS XV	B16	LOUIS XVI	.B14.B15.B16.
B14	LOUIS XV	B17	LOUIS XVIII	.B14.B15.B17.
B14	LOUIS XV	B18	CHARLES X	.B14.B15.B18.
C00	HUGHES CAPET	C01	ROBERT II	.C00.C01.
C01	ROBERT II	C02	HENRI I	.C01.C02.
C02	HENRI I	C03	PHILIPPE I	.C02.C03.
C03	PHILIPPE I	C04	LOUIS VI	.C03.C04.
C04	LOUIS VI	C05	LOUIS VII	.C04.C05.
C05	LOUIS VII	C06	PHILIPPE II	.C05.C06.
C06	PHILIPPE II	C07	LOUIS VIII	.C06.C07.
C07	LOUIS VIII	C08	LOUIS IX	.C07.C08.
...				
C08	LOUIS IX	C09	PHILIPPE III	.C08.C09.
C09	PHILIPPE III	C10	PHILIPPE IV	.C09.C10.
C09	PHILIPPE III	V01	PHILIPPE VI	.C09.V00.V01.
C10	PHILIPPE IV	C11	LOUIS X	.C10.C11.
C10	PHILIPPE IV	C12	PHILIPPE V	.C10.C12.
C10	PHILIPPE IV	C13	CHARLES IV	.C10.C13.
C11	LOUIS X	C14	JEAN I	.C11.C14.
V01	PHILIPPE VI	V02	JEAN II	.V01.V02.
V02	JEAN II	V03	CHARLES V	.V02.V03.
...				
V13	FRANCOIS I	V14	HENRI II	.V13.V14.
V14	HENRI II	V15	FRANCOIS II	.V14.V15.
V14	HENRI II	V16	CHARLES IX	.V14.V16.
V14	HENRI II	V17	HENRI III	.V14.V17.

Figure 12.10 - The royal hierarchy produced by Script 12.12

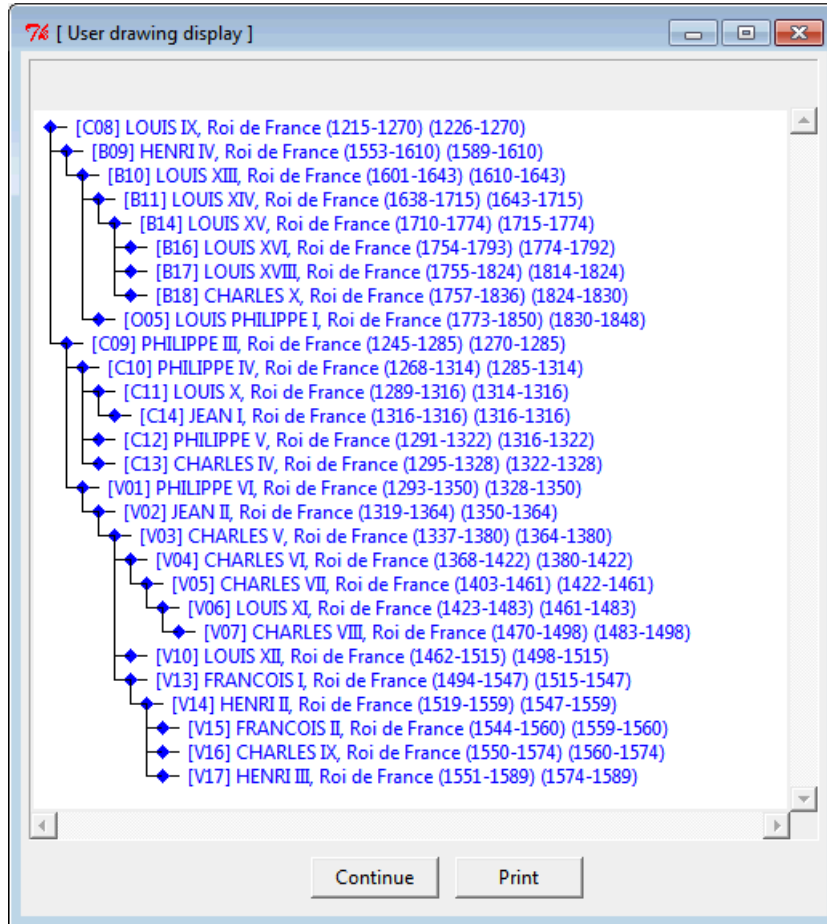


Figure 12.11 - The royal hierarchy based on ancestor/descendant relationships

12.8 The scripts

The algorithms and programs developed in this study are available as SQLfast scripts in directory **SQLfast/Scripts/Case-Studies/Kings-of-France**. Actually, they can be run from main script **Kings-MAIN.sql**, that displays the selection box of Figure 12.12.

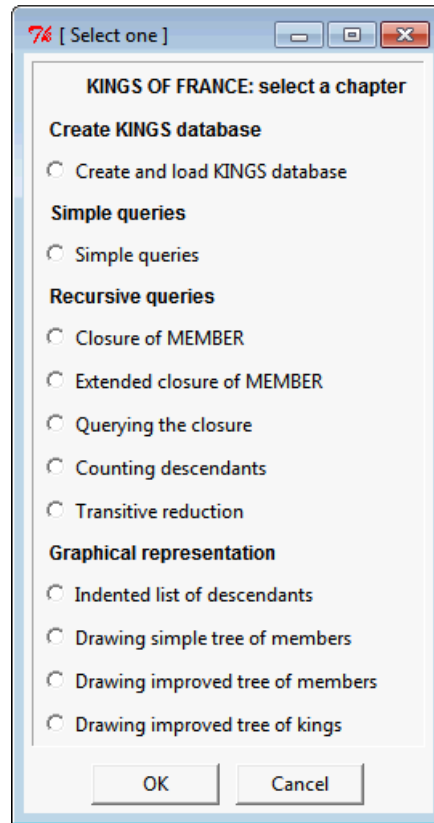


Figure 12.12 - Kings of France: selecting an application

These scripts are provided without warranty of any kind. Their sole objectives are to concretely illustrate the concepts of the case study and to help the readers master these concepts, notably in order to develop their own applications.