Case study **15**

Directory management

Objective: The contents of storage media, such as hard disks and flash disks, both internal and external, are organized into a hierarchical structure made up of directories and files.

This chapter shows that, when such structures are stored in a database, processes can be designed easily to examine directories, to analyze their contents, to describe their evolution and to discover potential problems. In particular, small applications will be developed to extract statistics, to display the structure and contents of a directory, do identify and describe potentially duplicate files and directories within a root directory or between two directories.

The problem of fast clone detection, that is, of set of files that have *exactly* the same contents, is also analyzed and solved.

Keywords: directory structure, tree modeling, tree analysis, statistics, tree evolution, duplicate files, clone detection, secure hashing, SHA256, database performance, CTE, recursive queries

15.1 Of files and directories

In this chapter, we will deal with the way the contents of storage media (internal/external disks and Flash memory for example) are organized. Each of these media provides a space in which *files* of any kind are permanently stored. When their number are getting larger, we find it convenient to collect these files in *directories* according to some logical rules. A directory generally contains files but may also contain directories (of which the former is the *parent*), that, in turn, may contain other files and directories. The storage medium can be seen as a special directory.

Directory or folder?

> *Directories*, the kind of container we just discussed about may also be called *folders*. Actually, Unix and Unix-like taxonomies favor the abstract term *directory* (as well as MS-DOS) while Windows and Mac OS have long introduced the more practical term *folder*. In this chapter, the terms *folder* and *directory* will be used interchangeably.[1]

Seen as a whole, the contents of a directory is structured as a *tree*, of which the latter is the root. Each node is either a directory or a file. Each edge represents the inclusion relation between a directory and each of its children node. The root of the tree is always a directory and its leaves are either files or empty directories.

Figure 15.1 shows a typical aspect of the standard file explorer of MS Windows. File DIRECTORY.sql is selected in folder Chapter-48. This folder is located in parent folder SQLfast-Tutorials, itself contained in folder Scripts of parent folder SQLfast-std, at the root of internal disk D:/. The graphical representation of the left panel shows clearly the tree structure of the directories.

Character string 'DIRECTORY.sql' is the *local name* of this file, that is, its name within its immediate parent folder. Its *full name* also comprises the names of all its direct and indirect parents:

```
D:/SQLfast-std/Scripts/SQLfast-Tutorials/Chapter-48/DIRECTORY.sql
```

Files have some interesting properties, such as their *extension*, their *size*, the time they were *last updated* or *last accessed*.

15.2 Examining directories

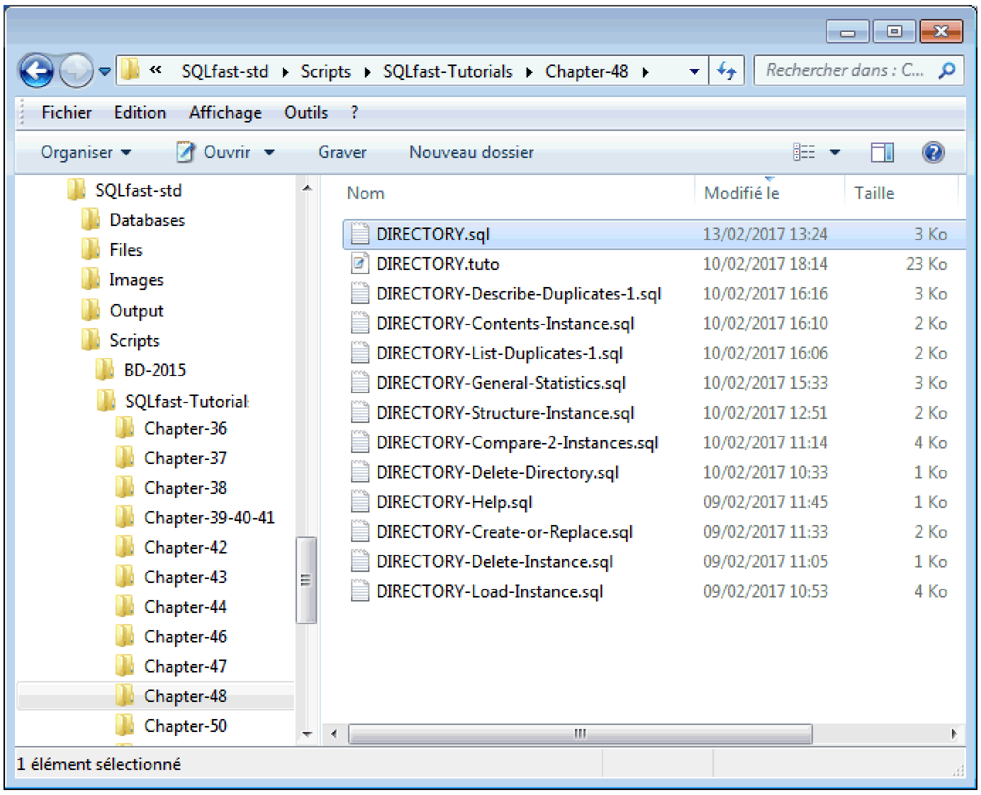
Directories surely are a convenient and intuitive way to organize the many files we store on our disk. However, after a while, they also become a real headache when the number and the variety of these files are getting unmanageable. Files are modified, their successive versions are kept, sometimes saved in more than one copy, quite often scattered across one or several disks. Directories and files are not always

1. https://msdn.microsoft.com/en-us/library/bb513869.aspx

named in a pertinent way, so that these names may give little hints about their contents.

Hence the need to better control the collection of our files distributed among the internal hard disk(s) and all these external hard disks and USB flash keys. In particular, displaying and analyzing their contents is the first task, for which we will develop some useful but simple tools.

Since we are going to examine and query trees, we could just reuse the scripts we created for the *Kings of France* application (Chapters 35 and 36). However, directory trees have quite different properties and suggest specific processes, that make it worth to develop a new approach.



Nom	Modifié le	Taille
DIRECTORY.sql	13/02/2017 13:24	3 Ko
DIRECTORY.tuto	10/02/2017 18:14	23 Ko
DIRECTORY-Describe-Duplicates-1.sql	10/02/2017 16:16	3 Ko
DIRECTORY-Contents-Instance.sql	10/02/2017 16:10	2 Ko
DIRECTORY-List-Duplicates-1.sql	10/02/2017 16:06	2 Ko
DIRECTORY-General-Statistics.sql	10/02/2017 15:33	3 Ko
DIRECTORY-Structure-Instance.sql	10/02/2017 12:51	2 Ko
DIRECTORY-Compare-2-Instances.sql	10/02/2017 11:14	4 Ko
DIRECTORY-Delete-Directory.sql	10/02/2017 10:33	1 Ko
DIRECTORY-Help.sql	09/02/2017 11:45	1 Ko
DIRECTORY-Create-or-Replace.sql	09/02/2017 11:33	2 Ko
DIRECTORY-Delete-Instance.sql	09/02/2017 11:05	1 Ko
DIRECTORY-Load-Instance.sql	09/02/2017 10:53	4 Ko

Figure 15.1 - Screen shot of Windows explorer

15.3 Modeling the problem

We intend to store in a database the description of the nodes of a directory in order to query it in various ways. However, further examination of the problem shows that some of its aspects may be more complex than thought at first glance. In particular,

1. considered statically, a directory may contain duplicate files (and directories)
2. directories generally evolve in time: files are updated, deleted, created, renamed, moved from one place to another one and duplicated
3. file duplication also appears among several directories, particularly if we create backup versions on different devices.

Addressing these spatial and temporal aspects of directories leads us to identify three main concepts:

1. *Directory*: a directory is uniquely identified by its name throughout all the devices we intend to manage. This concept is time independent: directory D:/SQLfast on our internal hard disk has been created several years ago and still exists, though with different contents. To distinguish these directories from those found in their composition, we call them *root directories*.
2. *Instance of directory*: the state of a root directory at a definite point in time (a *date-time*); in other words, a *snapshot* of a directory. All the instances of a directory have distinct date-time.
3. *Node*: the state of an element, either a directory or a file, in an instance of a directory. A node belongs to one instance. Therefore, a file that has been captured in two instances is represented by two different nodes. Deciding whether two nodes describe the same file must be done based on their properties (name, size, last update time) and position in the tree of their instances.

Let us illustrate and refine these concepts with an example.

We want to describe two root directories: **SQLfast** and **SQLfast-std**, located at the root of partition D of our internal hard disk. We decide to name them Local-D:/SQLfast and Local-D:/SQLfast-std. Why not just D:/SQLfast? Because this name alone may prove ambiguous if, later, we decide to record the contents of another computer in which a directory named D:/SQLfast also exists. So, we distinguish the real system name of a root directory (D:/SQLfast) from its *nickname* we assign to it to resolve potential name ambiguities (Local-D:/SQLfast). Nicknames are unique in the scope we define while system name are may not be.

We record the state of SQLfast-std on January 6th (twice), 9th and 10th, therefore creating four instances of it. Same for SQLfast, to which three instances are associated.

For each instance, we record (the description of) all the directories and files that composed its root directory at the time the instance was captured. These directories (root included) and files constitute the nodes of this instance. Two successive instances of a directory are likely to have a different composition.

15.4 Data structures

The implementation of these three concepts is straightforward. Root directories are recorded in table DIRECTORY, their instances are recorded in table INSTANCE and their components in table NODE. In each table, the rows are assigned a unique arbitrary number.

Table DIRECTORY records the system name (as their full path) of root directories (DirName), their unique nicknames (DirNick) and assigns them a technical primary key (DirID).

Table INSTANCE records all the instances of each root directory (column DirID) at a time point (InstDate and InstTime, in ISO format), down to the second. Each instance also is assigned a technical primary key (InstID).

Table NODE records the description of all the components of each instance (column InstID). A component is either a directory (NType = 'D') or a file (NType = 'F'). The root directory is itself a component of all its instances. Each node is assigned a unique Id (NodeID).

Each node has a parent node (ParentID), which is the directory node in which it is located, except root directories, which have no parent (this is their distinctive property).

A node has a full name (FullName), that is, the unique path starting from the device to its local name. The local name of a node (LocalName) is its name within its immediate parent directory. The prefix of a node (Prefix) is the full name of its immediate parent directory. Local name and prefix derive from the decomposition the full name (FullName).

Four properties are specific to file nodes:

- their extension, derived from their local name (Extension)
- their size, in Bytes (Size)
- the last time they were updated (LastUpdate)
- the last time they were accessed without modification (LastAccess)

Two derived properties synthesize the overall structure of instances:

- the level of a node within its instance tree (Level); root directory nodes are assigned level 0 and the level of a node is that of its parent + 1
- the suite of nodes from the root directory to each node, those included (Path)

Column Hash, that will be useful to identify clone files, will be explained later. These data structures are translated in SQL-DDL in Script 15.1.

Figures 15.2 and 15.3 show excerpts of the data that represent the example described above.

- Table DIRECTORY stores (the description of) root directories D:/SQLfast (nickname Local-D:/SQLfast) and D:/SQLfast-std (nickname Local-D:/SQLfast-std).

– Table INSTANCE stores three instances of Local-D:/SQLfast and four instances of Local-D:/SQLfast-std.
– The contents of table NODE is fairly large, so that one node only is shown in a `showData` box (Figure 15.3).

```
create table DIRECTORY(
       DirID      integer not null primary key autoincrement,
       DirName    varchar(1024) not null),
       DirNick    varchar(1024) not null unique);

create table INSTANCE(
       InstID     integer not null primary key autoincrement,
       DirID      integer not null
                  references DIRECTORY on delete cascade,
       InstDate   date not null,
       InstTime   time not null,
       unique (DirID,InstDate,InstTime));

create table NODE(
       NodeID     integer not null primary key autoincrement,
       InstID     integer not null
                  references INSTANCE on delete cascade,
       NType      char(1) not null,
       ParentID   integer
                  references NODE on delete set null,
       FullName   varchar(2048),
       Prefix     varchar(2048),
       LocalName  varchar(256),
       Extension  varchar(32),
       Size       integer,
       LastUpdate char(19),
       LastAccess char(19),
       Path       varchar(2048),
       Level      integer,
       Hash       char(64);
```

Script 15.1 - The tables of database DIRECTORY.db

This node describes a file (NType = F) named DIRECTORY.sql (LocalName), located in directory D:/SQLfast-std/Scripts/SQLfast-Tutorials/Chapter-48 (Prefix). Its node Id (NodeID) is 15813 and it belongs to instance 8 of root directory D:SQLfast (ParentID).
It is at level 4 from its root directory (column Level) and it can be reached (column Path) from the root directory node (NodeID = 14400) by following intermediate nodes 14897, 15163 and 15801. The prefix letter, D or F, is a reminder of the nature of the node. Column Hash is not shown.

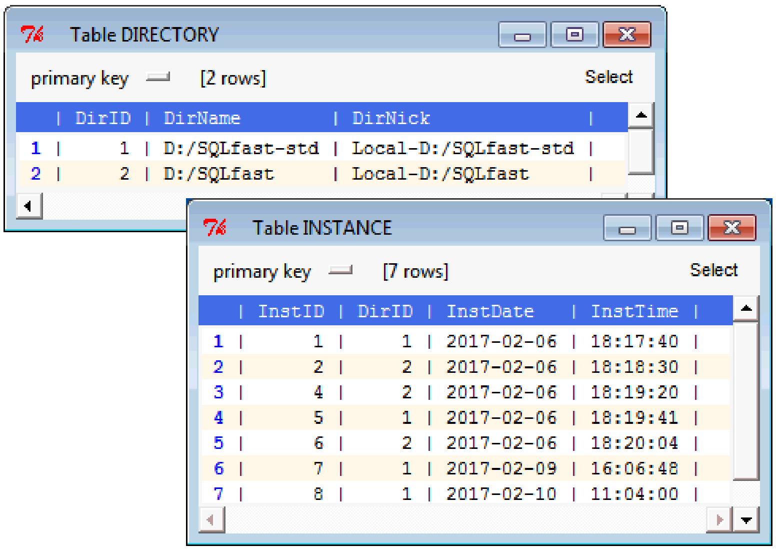


Table DIRECTORY

primary key [2 rows] Select

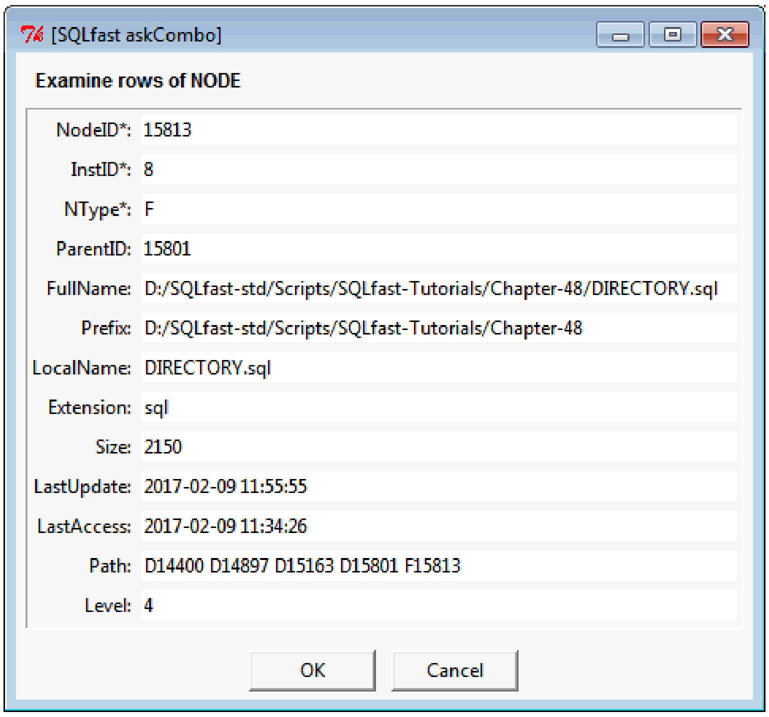
```
  | DirID | DirName          | DirNick                |
1 |     1 | D:/SQLfast-std   | Local-D:/SQLfast-std   |
2 |     2 | D:/SQLfast       | Local-D:/SQLfast       |
```

Table INSTANCE

primary key [7 rows] Select

```
  | InstID | DirID | InstDate   | InstTime |
1 |      1 |     1 | 2017-02-06 | 18:17:40 |
2 |      2 |     2 | 2017-02-06 | 18:18:30 |
3 |      4 |     2 | 2017-02-06 | 18:19:20 |
4 |      5 |     1 | 2017-02-06 | 18:19:41 |
5 |      6 |     2 | 2017-02-06 | 18:20:04 |
6 |      7 |     1 | 2017-02-09 | 16:06:48 |
7 |      8 |     1 | 2017-02-10 | 11:04:00 |
```

Figure 15.2 - DIRECTORY and INSTANCE tables



[SQLfast askCombo]

Examine rows of NODE

NodeID*:	15813
InstID*:	8
NType*:	F
ParentID:	15801
FullName:	D:/SQLfast-std/Scripts/SQLfast-Tutorials/Chapter-48/DIRECTORY.sql
Prefix:	D:/SQLfast-std/Scripts/SQLfast-Tutorials/Chapter-48
LocalName:	DIRECTORY.sql
Extension:	sql
Size:	2150
LastUpdate:	2017-02-09 11:55:55
LastAccess:	2017-02-09 11:34:26
Path:	D14400 D14897 D15163 D15801 F15813
Level:	4

Figure 15.3 - Composition of a node (here script file DIRECTORY.sql)

15.5 Loading a directory instance

Statement `dirFileNamesAll-r` is used to extract recursively (suffix `-r`) the contents of a directory. Script fragment 15.2 lets the user select a root directory (`selectDirectory`) and stores its content in file names.txt.

```
set path = D:/;
selectDirectory path = $path$;
dirFileNamesAll-r $fileDirectory$/names.txt = $path$;
```

Script 15.2 - Selecting a root directory and recording its contents in file names.txt

Then, the user is invited to assign a nickname to the root directory, either by entering a new name (the path of the directory just selected is suggested) in which case both a new root directory and its first instance are created, or by selecting one among those already recorded, in which case a new instance is created for this root directory (Script 15.3).

```
set rnn = $path$;
ask-u rnn = [/bSelect/Enter the nickname of the root directory]
             Root directory nickname:[select DirNick
                                      from   DIRECTORY];
```

Script 15.3 - Getting the nickname of the selected root directory

15.5.1 Creating a directory

If this root directory already exists, its Id (dirID) is extracted, otherwise, a new DIRECTORY row is inserted (Script 15.4).

```
extract dirID = select DirID from DIRECTORY
                where  DirNick = '§rnn§';
if ('$SQLdiag$' = 'NONE');
   insert into DIRECTORY(DirName,DirNick)
          values ('§path§','§rnn§');
   extract dirID = select DirID from DIRECTORY
                   where  DirNick = '§rnn§';
endif;
```

Script 15.4 - Creating a root directory if needed

15.5.2 Creating the current instance

Now, we have enough information to record the information on the current instance in a row of table INSTANCE (Script 15.5). Variables dat and tim have preliminarily been set to the current date and time.

```
insert into INSTANCE(DirID,InstDate,InstTime)
             values ($dirID$,'$dat$','$tim$');
extract instID = select InstID from INSTANCE
                 where  DirID = $dirID$
                 and    InstDate = '$dat$'
                 and    InstTime = '$tim$';
```

Script 15.5 - Creating an instance of the root directory

15.5.3 Creating the nodes of the current instance

The next step is to store the nodes of this instance (Script 15.6). Their names are extracted from file names.txt. Each line is decomposed by function splitPath of library LFile. It returns five values: the *full name*, the *prefix*, the *local name* for a directory (empty for a file), the *local name* for a file (empty for a directory) and the *extension* (empty for a directory).

For files, additional informations on *size*, *last update time* and *last access times* are collected through function infoFile of library LFile.

```
for line = [file $namesFile$];
  function ful,pre,dir,fil,ext = LFile:splitPath {$line$};
  if ('§fil§' = '');
    insert into NODE(InstID,NType,FullName,Prefix,LocalName)
              values($instID$,'D','§ful§','§pre§','§dir§');
  else;
    function siz,cre,upd,acc = LFile:infoFile {$line$};
    insert into NODE(InstID,NType,FullName,Prefix,LocalName,
                     Extension,Size,LastUpdate,LastAccess)
              values($instID$,'F','§ful§','§pre§','§fil§',
                     '§ext§',$siz$,'$upd$','$acc$');
  endif;
endfor;
```

Script 15.6 - Creating the nodes of the current instance of the root directory

Filling column ParentID is straightforward: the full name of the parent of a node is the prefix of this node in the same instance. Hence the `update` query of Script 15.7.

The level of a node could be computed by a recursive query, but it is more simply derived from the number of symbol `'/'` in its full name, relative to that of its root directory (Script 15.7, last part).

```
update NODE
set    ParentID = (select NodeID from NODE
                   where  FullName  = NODE.Prefix
                   and    InstID  = $instID$)
where  InstID = $instID$;

compute NL = countInst('§path§','/');
update NODE set Level = countInst(FullName,'/') - $NL$
where  InstID = $instID$;
```

Script 15.7 - Computing the values of columns ParentID and Level

15.5.4 Computing the path of a node

The value of column Path of a node is the list of node Id's from the root directory (included) to this node (included). Computing paths in graphs in general, and in trees in particular, is a nice application of recursive queries also known as *recursive CTE* in SQL (see Chapter 36 for example).

In Script 15.8, CTE TREE *extracts* couples (ParentID, NodeID) from NODE and recursively computes the path of a node (starting from the root directory) from that of its parent, to which it appends its own node Id.

```
with TREE(FromID,ToID,Path) as (
   select ParentID,NodeID,cast(NodeID as char)
   from   NODE where ParentID is null
      union all
   select T.ToID,N.NodeID,T.Path||' '||cast(N.NodeID as char)
   from   TREE T, NODE N
   where  T.ToID = N.ParentID
   )
update NODE
set Path = (select Path from TREE where ToID = NODE.NodeID);
```

Script 15.8 - Computing node paths - Recursive version

The way node id's are formatted in the path is a bit raw. For example, the sequence <937,1038,11847,32> will be expressed as

```
'937 1038 11847 32'
```

which is not particularly elegant and, more important, will not serve the depth-first ordering role we intend to assign to it (see below). So, we suggest to replace expression

```
cast(NodeID as char)
```

by[2]

```
NType||frame(NodeID,$nd$,'>','0')
```

which obviously deserves some explanation!

Function `frame(s,nd,a,p)` returns string `s` framed in a `nd`-long string; `a` is the alignment ('<' = left, '>' = right, '^' = center) of `s` in the frame; unused positions are padded with characters `p`. The value of `nd` is computed as the number of digits of the largest node Id:

```
extract nd = select length(cast(max(NodeID) as char))
             from   NODE;
```

In addition, each node Id is prefixed with the nature of the node (D or F). The result becomes quite regular:

```
'D00937 D01038 D11847 F00032'
```

This algorithm can also be translated into an iterative script. Its logic is quite similar to that of the recursive version:

– first, the path of the root node is set to its own node Id,
– then, each iteration computes the path of the children of each node of the preceding level, until all levels have been processed (Script 15.9).

```
extract maxLevel = select max(Level) from NODE;
update NODE set Path = cast(NodeID as char)
where  Level = 0;
for iLev = [1,$maxLevel$];
   update NODE
   set Path = (select Path from NODE
               where  NodeID = NODE.ParentID)
                             ||' '||cast(NodeID as char)
   where Level = $iLev$;
endfor;
```

Script 15.9 - Computing node paths - Iterative version

Note that, for simplicity, both algorithms have been designed for a table NODE that stores one directory instance only. The fully functional scripts must restrict the processing to the current instance, that is, to those NODE rows that satisfy condition

```
where InstID = $instID$
```

2. Function `frame` transforms numeric values into character strings, so that casting them into `char` type is unnecessary.

15.6 Applications

The value of a database lies in the quality of the data but also on the problem solving application portfolio. We will develop some simple applications that focus on the analysis of a collection of directories in order to understand its structure and its contents on one hand, and to search them for potential duplicates on the other hand, a frequent problem of people who use computers for several years!

The applications described below just are suggestions that are intended to be adapted and completed to meet specific readers needs.

15.7 Application 1: general statistics

Some global statistics will give us a general picture on our directories (Script 15.10). Script 15.11 provides more detailed statistics on the instances of each root directory, notably providing the time range of these instances.

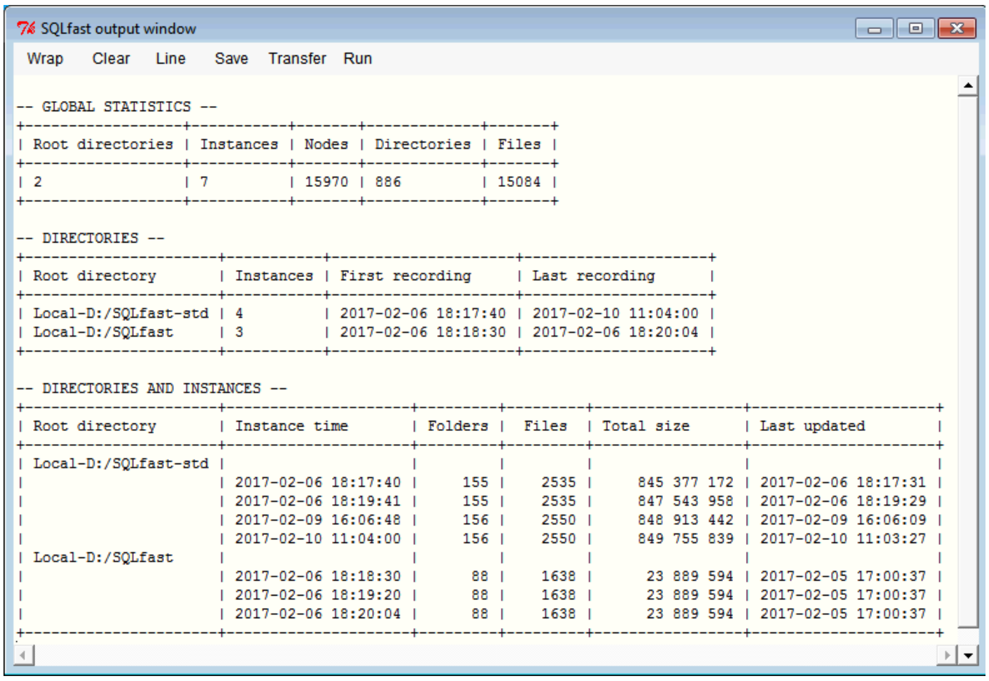
```
select
  (select count(*) from DIRECTORY) as "Root directories",
  (select count(*) from INSTANCE)  as Instances,
  (select count(*) from NODE)      as Nodes,
  (select count(*) from NODE where NType = 'D') as Directories
  (select count(*) from NODE where NType = 'F') as Files;
```

Script 15.10 - Displaying the size of the tables

```
select D.DirNick                    as "Root directory",
       count(*)                     as Instances,
       min(InstDate||' '||InstTime) as "First recording",
       max(InstDate||' '||InstTime) as "Last recording"
from   DIRECTORY D, INSTANCE I
where  D.DirID = I.DirID
group by D.DirID;
```

Script 15.11 - Statistics for each root directory

Excerpts of the result are shown in Figure 15.4, top two tables. This screen shot also shows the hierarchical instance description produced by Script 15.12. For each directory and for each of its instances, it provides the instance time, the number of directories, the number of files, the total file size and the most recent file update time.



```
-- GLOBAL STATISTICS --
+------------------+-----------+-------+-------------+-------+
| Root directories | Instances | Nodes | Directories | Files |
+------------------+-----------+-------+-------------+-------+
| 2                | 7         | 15970 | 886         | 15084 |
+------------------+-----------+-------+-------------+-------+

-- DIRECTORIES --
+----------------------+-----------+---------------------+---------------------+
| Root directory       | Instances | First recording     | Last recording      |
+----------------------+-----------+---------------------+---------------------+
| Local-D:/SQLfast-std | 4         | 2017-02-06 18:17:40 | 2017-02-10 11:04:00 |
| Local-D:/SQLfast     | 3         | 2017-02-06 18:18:30 | 2017-02-06 18:20:04 |
+----------------------+-----------+---------------------+---------------------+

-- DIRECTORIES AND INSTANCES --
+----------------------+---------------------+---------+---------+-----------------+---------------------+
| Root directory       | Instance time       | Folders |  Files  | Total size      | Last updated        |
+----------------------+---------------------+---------+---------+-----------------+---------------------+
| Local-D:/SQLfast-std |                     |         |         |                 |                     |
|                      | 2017-02-06 18:17:40 |     155 |    2535 |     845 377 172 | 2017-02-06 18:17:31 |
|                      | 2017-02-06 18:19:41 |     155 |    2535 |     847 543 958 | 2017-02-06 18:19:29 |
|                      | 2017-02-09 16:06:48 |     156 |    2550 |     848 913 442 | 2017-02-09 16:06:09 |
|                      | 2017-02-10 11:04:00 |     156 |    2550 |     849 755 839 | 2017-02-10 11:03:27 |
| Local-D:/SQLfast     |                     |         |         |                 |                     |
|                      | 2017-02-06 18:18:30 |      88 |    1638 |      23 889 594 | 2017-02-05 17:00:37 |
|                      | 2017-02-06 18:19:20 |      88 |    1638 |      23 889 594 | 2017-02-05 17:00:37 |
|                      | 2017-02-06 18:20:04 |      88 |    1638 |      23 889 594 | 2017-02-05 17:00:37 |
+----------------------+---------------------+---------+---------+-----------------+---------------------+
```

Figure 15.4 - General statistics

This script is a bit more complex than the queries developed so far. It comprises three main parts:

1. the **`from`** clause builds the union of two results sets, one producing the identification of root directories and the second one the description of instances. Since they must comprise the same number of columns, those of them that are not significant are left empty (string `''`).
2. the **`order by`** clause creates the hierarchical sequence that presents each directory immediately followed by its instances. To do so, we define special computed column **`sort`**. Its values are that of DirID in both tables, so that all DIRECTORY and INSTANCE rows are grouped, followed by a character that forces the DIRECTORY row to precede its INSTANCE rows: we naturally have chosen `'D'` for DIRECTORY and `'I'` for INSTANCE.
3. the **`select`** clause formats the values from the combined result sets and assigns them expressive names. Function `thousandSep(n,sep)` returns the character string formed by number `n`, in which character `sep` separates groups of three digits in the integral part of `n`. E.g., `thousandSep(1234567.45,',')` returns `'1,234,567.45'`.

```
select DirNick                      as "Root directory",
       Time                         as "Instance time",
       frame(Folders,7,'>',' ')  as Directories,
       frame(Files,7,'>',' ')    as " Files",
       frame(thousandSep(Size,' '),15,'>',' ') as "Total size",
       Last                         as "Last updated"
from
 (select DirNick,'' as Time,
         '' as Folders, '' as Files,'' as Size,
         '' as Last, cast(DirID as char)||'D' as Sort
  from DIRECTORY D
    union
  select '' as DirNick,InstDate||' '||InstTime as Time,
         (select count(*) from NODE N where NType = 'D'
          and  N.InstID = I.InstID) as Folders,
         (select count(*) from NODE N
          where  NType = 'F' and N.InstID = I.InstID) as Files,
         (select case NType when 'F' then sum(Size) else ''
                 end from NODE N
          where  N.InstID = I.InstID) as Size,
         (select max(LastUpdate) from NODE N
          where  NType = 'F' and N.InstID = I.InstID) as Last,
          cast(DirID as char)||'I' as Sort
  from INSTANCE I)
order by Sort;
```

Script 15.12 - Displaying the directory/instance hierarchy

15.8 Selecting a directory instance

All the following applications will carry out an analysis of a definite directory instance (or two instances for some of them). In order to alleviate the code of these applications, we could create specific scripts to select this instance and to get the nickname of the directory. We could also use a lighter technique: storing common patterns as code fragments in SQLfast variable. These patterns are shown in Script 15.13. Selecting an instance is now easy:

```
ask inst = Instance:[!$listInstances$];
```

and extracting the directory name of instance `$inst$` is quite straightforward as well:

```
extract iName = $instName$;
```

```
set listInstances
    = select D.DirNick||' ('||I.InstDate
             ||' '||I.InstTime||')' as Name,
             I.InstID
      from   DIRECTORY D,INSTANCE I
      where  D.DirID = I.DirID
      order by Name;

set instName
    = select DirNick||' ('||InstDate||' '||InstTime||')'
      from   DIRECTORY D, INSTANCE I
      where  InstID = $inst$ and D.DirID = I.DirID;
```

Script 15.13 - Two useful common patterns to select instances

15.9 Application 2: Structure of a directory instance

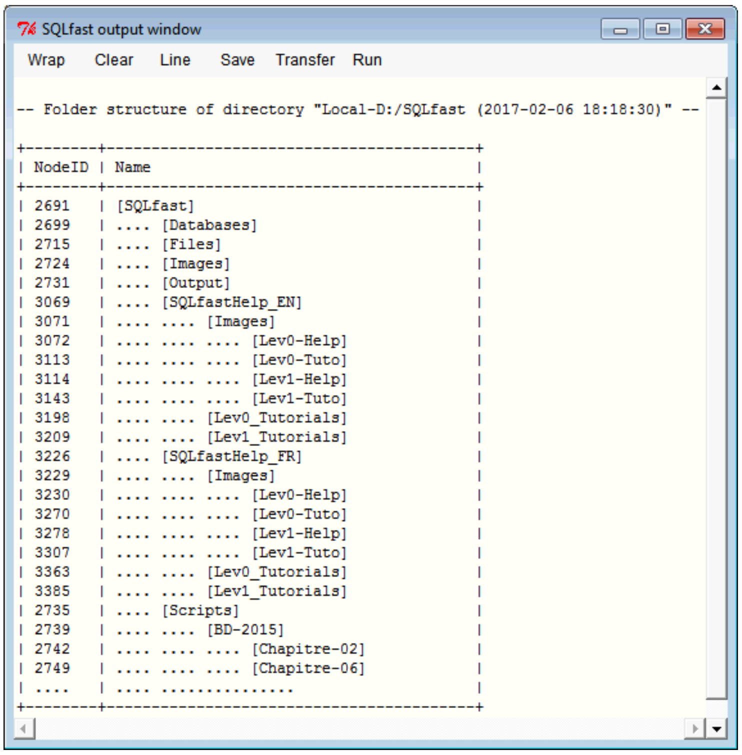
We call *structure of an instance* the list of internal directories that shows, through indentation, the inclusion relationships. It represents graphically the tree structure of these directories (Figure 15.5).

Script 48.14 that produces this structure is straightforward. The generation of the sequences of dots that create the indentation is based on column Level of table NODE (function `repeat(s,n)` returns a character string formed by **n** time string **s**). To make directory names distinguishable from those of files[3], the former are surrounded by square brackets.

```
ask inst = Instance:[!$listInstances$];
extract iName = $instName$;
write-ab -- Directory structure of directory "$iName$" --;
select NodeID,
       repeat('.... ',Level)
     ||case when NType = 'F'
            then LocalName
            else '['||LocalName||']'
       end as Name
from   NODE
where InstID = $inst$ and NType = 'D'
order by Path;
```

Script 15.14 - Displaying the directory structure of an instance

3. Not particularly striking in this case, but will be useful in the next report!



```
-- Folder structure of directory "Local-D:/SQLfast (2017-02-06 18:18:30)" --

+--------+---------------------------------------+
| NodeID | Name                                  |
+--------+---------------------------------------+
| 2691   | [SQLfast]                             |
| 2699   | .... [Databases]                      |
| 2715   | .... [Files]                          |
| 2724   | .... [Images]                         |
| 2731   | .... [Output]                         |
| 3069   | .... [SQLfastHelp_EN]                 |
| 3071   | .... .... [Images]                    |
| 3072   | .... .... .... [Lev0-Help]            |
| 3113   | .... .... .... [Lev0-Tuto]            |
| 3114   | .... .... .... [Lev1-Help]            |
| 3143   | .... .... .... [Lev1-Tuto]            |
| 3198   | .... .... [Lev0_Tutorials]            |
| 3209   | .... .... [Lev1_Tutorials]            |
| 3226   | .... [SQLfastHelp_FR]                 |
| 3229   | .... .... [Images]                    |
| 3230   | .... .... .... [Lev0-Help]            |
| 3270   | .... .... .... [Lev0-Tuto]            |
| 3278   | .... .... .... [Lev1-Help]            |
| 3307   | .... .... .... [Lev1-Tuto]            |
| 3363   | .... .... [Lev0_Tutorials]            |
| 3385   | .... .... [Lev1_Tutorials]            |
| 2735   | .... [Scripts]                        |
| 2739   | .... .... [BD-2015]                   |
| 2742   | .... .... .... [Chapitre-02]          |
| 2749   | .... .... .... [Chapitre-06]          |
| ....   | .... ...............                  |
+--------+---------------------------------------+
```

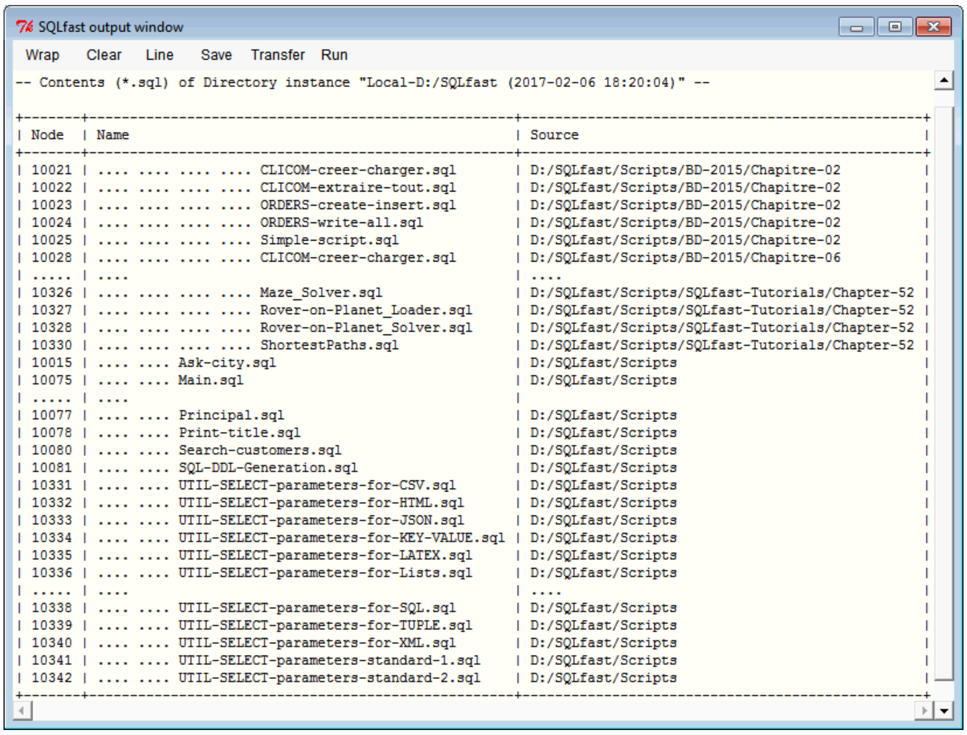
Figure 15.5 - Structure of directory SQLfast-std (excerpts)

The correct tree structure of the report is defined by the `order by` clause, that implements, through the values of column Path, the *depth-first* traversal order of the node tree, according to which the children of a node N are *visited* before visiting the next sibling of N and its children.

15.10 Application 3: Examining the contents of an instance

This application is an extension of the previous one. The file and directory local names are presented with the same indentation technique while an additional column indicates the prefix of each node.

The contents of the directory can be limited to nodes of a certain type: directories only, files only, both (as in the previous report) or files with a definite extension. Figure 15.6 shows a report on *.sql scripts of the instance selected (Figure 15.7).



```
-- Contents (*.sql) of Directory instance "Local-D:/SQLfast (2017-02-06 18:20:04)" --

+-------+----------------------------------------------------+-------------------------------------------------+
| Node  | Name                                               | Source                                          |
+-------+----------------------------------------------------+-------------------------------------------------+
| 10021 | .... .... .... .... CLICOM-creer-charger.sql       | D:/SQLfast/Scripts/BD-2015/Chapitre-02          |
| 10022 | .... .... .... .... CLICOM-extraire-tout.sql       | D:/SQLfast/Scripts/BD-2015/Chapitre-02          |
| 10023 | .... .... .... .... ORDERS-create-insert.sql       | D:/SQLfast/Scripts/BD-2015/Chapitre-02          |
| 10024 | .... .... .... .... ORDERS-write-all.sql           | D:/SQLfast/Scripts/BD-2015/Chapitre-02          |
| 10025 | .... .... .... .... Simple-script.sql              | D:/SQLfast/Scripts/BD-2015/Chapitre-02          |
| 10028 | .... .... .... .... CLICOM-creer-charger.sql       | D:/SQLfast/Scripts/BD-2015/Chapitre-06          |
| ..... | ....                                               | ....                                            |
| 10326 | .... .... .... .... Maze_Solver.sql                | D:/SQLfast/Scripts/SQLfast-Tutorials/Chapter-52 |
| 10327 | .... .... .... .... Rover-on-Planet_Loader.sql     | D:/SQLfast/Scripts/SQLfast-Tutorials/Chapter-52 |
| 10328 | .... .... .... .... Rover-on-Planet_Solver.sql     | D:/SQLfast/Scripts/SQLfast-Tutorials/Chapter-52 |
| 10330 | .... .... .... .... ShortestPaths.sql              | D:/SQLfast/Scripts/SQLfast-Tutorials/Chapter-52 |
| 10015 | .... .... Ask-city.sql                             | D:/SQLfast/Scripts                              |
| 10075 | .... .... Main.sql                                 | D:/SQLfast/Scripts                              |
| ..... | ....                                               |                                                 |
| 10077 | .... .... Principal.sql                            | D:/SQLfast/Scripts                              |
| 10078 | .... .... Print-title.sql                          | D:/SQLfast/Scripts                              |
| 10080 | .... .... Search-customers.sql                     | D:/SQLfast/Scripts                              |
| 10081 | .... .... SQL-DDL-Generation.sql                   | D:/SQLfast/Scripts                              |
| 10331 | .... .... UTIL-SELECT-parameters-for-CSV.sql       | D:/SQLfast/Scripts                              |
| 10332 | .... .... UTIL-SELECT-parameters-for-HTML.sql      | D:/SQLfast/Scripts                              |
| 10333 | .... .... UTIL-SELECT-parameters-for-JSON.sql      | D:/SQLfast/Scripts                              |
| 10334 | .... .... UTIL-SELECT-parameters-for-KEY-VALUE.sql | D:/SQLfast/Scripts                              |
| 10335 | .... .... UTIL-SELECT-parameters-for-LATEX.sql     | D:/SQLfast/Scripts                              |
| 10336 | .... .... UTIL-SELECT-parameters-for-Lists.sql     | D:/SQLfast/Scripts                              |
| ..... | ....                                               | ....                                            |
| 10338 | .... .... UTIL-SELECT-parameters-for-SQL.sql       | D:/SQLfast/Scripts                              |
| 10339 | .... .... UTIL-SELECT-parameters-for-TUPLE.sql     | D:/SQLfast/Scripts                              |
| 10340 | .... .... UTIL-SELECT-parameters-for-XML.sql       | D:/SQLfast/Scripts                              |
| 10341 | .... .... UTIL-SELECT-parameters-standard-1.sql    | D:/SQLfast/Scripts                              |
| 10342 | .... .... UTIL-SELECT-parameters-standard-2.sql    | D:/SQLfast/Scripts                              |
+-------+----------------------------------------------------+-------------------------------------------------+
```

Figure 15.6 - List of all the sql scripts of the instance selected

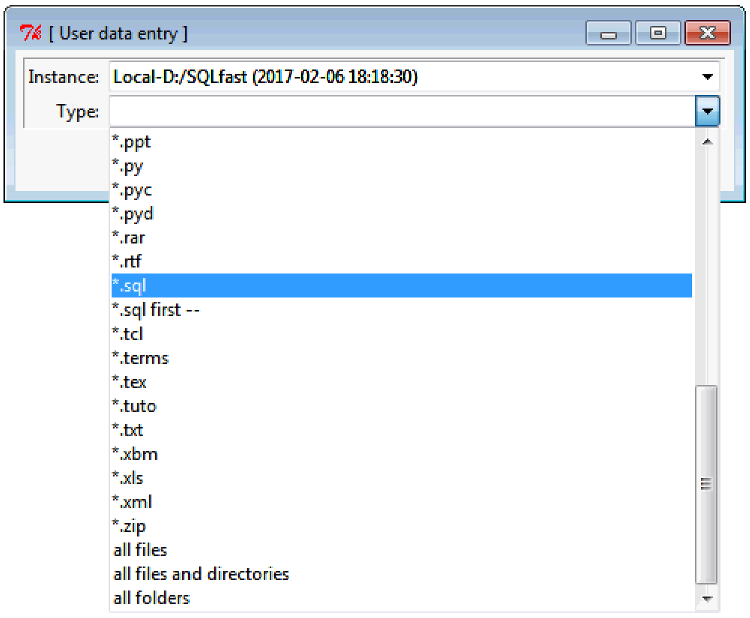
This report has been produced by Script 15.16, that comprises four main parts:

1. the dialogue box (Figure 15.7)
2. the sequence of `if ('$ext$'...)` statements that translate the node type into an SQL condition stored in variable extCond
3. extracting the name of the instance and writing the report title
4. the SQL query that produces the report itself.

To make the script simpler, we define pattern `listTypes`, that creates the list of the types of the file nodes in table NODE (Script 15.15).

15.11 Application 4: Duplicate nodes in a directory instance

Finding nodes that denote identical files in a root directory is one of the most important problem when managing a large collection of files. We first address the search of potentially duplicate files and directories in a root directory.



Instance: Local-D:/SQLfast (2017-02-06 18:18:30)
Type:

```
*.ppt
*.py
*.pyc
*.pyd
*.rar
*.rtf
*.sql
*.sql first --
*.tcl
*.terms
*.tex
*.tuto
*.txt
*.xbm
*.xls
*.xml
*.zip
all files
all files and directories
all folders
```

Figure 15.7 - Selecting the type of contents to display

```
set listTypes
    = select 'all files,all folders,...,'
             ||group_concat('*.'||Extension)
      from   (select distinct lower(Extension) as Extension
              from    NODE
              where   Extension <> ''
              order by lower(Extension));
```

Script 15.15 - Pattern that extracts the types of all the files recorded in table NODE

15.11.1 List of duplicate nodes in a directory instance

Through this small application, we want to know which nodes have the same name in a selected root directory. This indicator is not sufficient to declare them identical, but it is good starting point to evaluate redundancies in a directory.

Script 15.17 asks the user to select a directory and to specify the minimum multiplicity for which duplicate names must be reported.

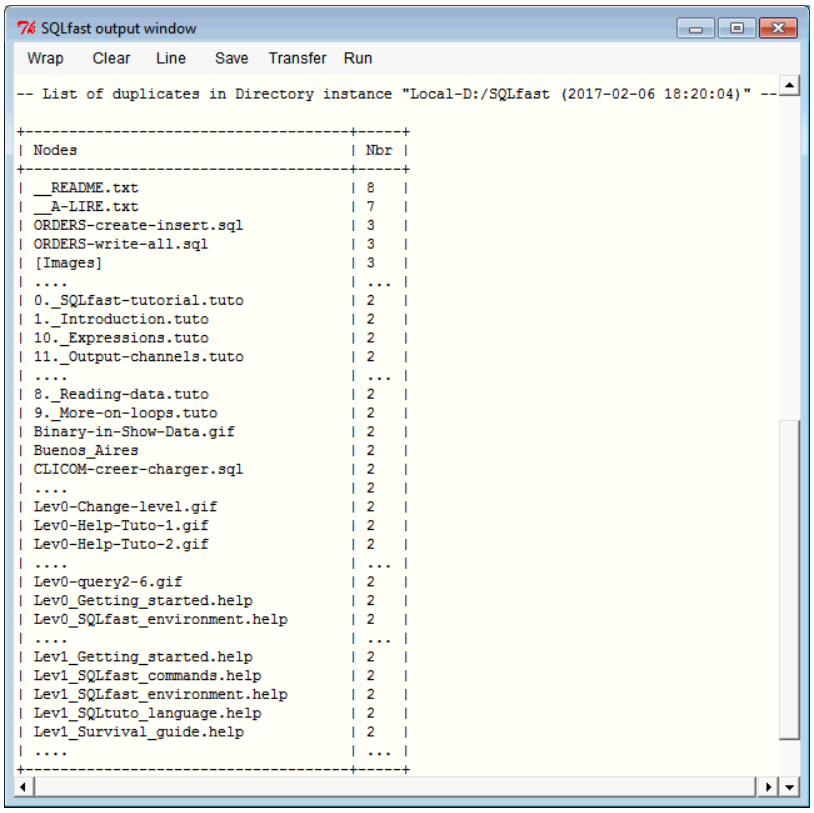
Figure 15.8 shows excerpts of the result of its execution for directory Local-D:/SQLfast for a multiplicity of 2. This report is just a short summary of the duplication phenomenon, but it is worth being examined to decide whether a more in-depth analysis may be useful. The latter is the objective of the next section

```
ask inst,ext = Instance:[!$listInstances$]
              |Type:[!$listTypes$];
if ('$ext$' = '') set ext = - all files and directories;
set extCond = ;
if ('$ext$' = 'all files and folders')
   set extCond = ;
if ('$ext$' = 'all files')
   set extCond = and NType = 'F';
if ('$ext$' = 'all directories')
   set extCond = and NType = 'D';
if (startswith('$ext$','*.'))
   set extCond = and Extension = substr('§ext§',3,99);
extract iName = $instName$;
write-ab -- Contents ($ext$) of Directory instance "$iName$";
select NodeID as Node,
       repeat('.... ',Level)
     ||case when NType = 'F'
            then LocalName
            else '['||LocalName||']'
       end as Name,
       Prefix as Source
from  NODE
where InstID = $inst$ $extCond$
order by Path;
```

Script 15.16 - Displaying the contents of a directory

```
set inst = ;
set mul  = 2;
ask-u inst,mul = Instance:[!$instName$] | Min multiplicity:;
if ('$DIALOGbutton$' = 'Cancel' or '$inst$' = '') goto EXIT;
extract iName = $instName$;
write-ab Duplicates in Directory instance "$iName$";
select case when NType = 'F'
            then LocalName
            else '['||LocalName||']'
       end as Nodes,
       count(*) as Nbr
from   NODE where InstID = $inst$
group by Nodes
having Nbr >= $mul$
order by Nbr desc, Nodes;
```

Script 15.17 - Identifying duplicate files and directories



```
-- List of duplicates in Directory instance "Local-D:/SQLfast (2017-02-06 18:20:04)" --

+-------------------------------------+-----+
| Nodes                               | Nbr |
+-------------------------------------+-----+
| __README.txt                        | 8   |
| __A-LIRE.txt                        | 7   |
| ORDERS-create-insert.sql            | 3   |
| ORDERS-write-all.sql                | 3   |
| [Images]                            | 3   |
| ....                                | ... |
| 0._SQLfast-tutorial.tuto            | 2   |
| 1._Introduction.tuto                | 2   |
| 10._Expressions.tuto                | 2   |
| 11._Output-channels.tuto            | 2   |
| ....                                | ... |
| 8._Reading-data.tuto                | 2   |
| 9._More-on-loops.tuto               | 2   |
| Binary-in-Show-Data.gif             | 2   |
| Buenos_Aires                        | 2   |
| CLICOM-creer-charger.sql            | 2   |
| ....                                | 2   |
| Lev0-Change-level.gif               | 2   |
| Lev0-Help-Tuto-1.gif                | 2   |
| Lev0-Help-Tuto-2.gif                | 2   |
| ....                                | ... |
| Lev0-query2-6.gif                   | 2   |
| Lev0_Getting_started.help           | 2   |
| Lev0_SQLfast_environment.help       | 2   |
| ....                                | ... |
| Lev1_Getting_started.help           | 2   |
| Lev1_SQLfast_commands.help          | 2   |
| Lev1_SQLfast_environment.help       | 2   |
| Lev1_SQLtuto_language.help          | 2   |
| Lev1_Survival_guide.help            | 2   |
| ....                                | ... |
+-------------------------------------+-----+
```

Figure 15.8 - Duplicate nodes in a root directory instance (excerpts)

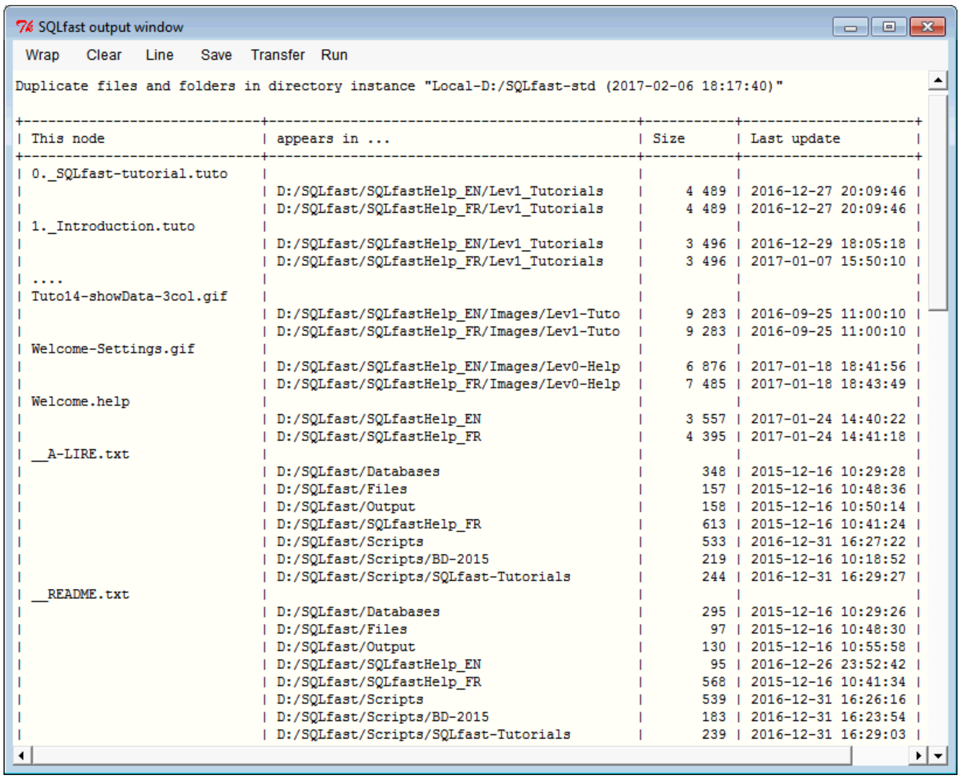
15.11.2 Description of duplicate nodes in a directory instance

This application extends the previous report by organizing duplicate nodes in a tree structure and by adding size and update time information to better compare potentially duplicate files (Figure 15.9).

We give the report a hierarchical structure in which, for each local name that appears more than once, we list the full names of the directories in which they are located, together with additional properties size and last update time.

The SQLfast code is given in Script 15.18. As usual, the hierarchy is created by the union of the rows providing the two kinds of lines in the result set: the common local names and the parent directories of these names.

The correct depth-first order is defined by the sort key **Sort** built by the common local name suffixed with letters `'A'` and `'B'` respectively.



```
Duplicate files and folders in directory instance "Local-D:/SQLfast-std (2017-02-06 18:17:40)"

+-----------------------------+---------------------------------------------+-----------+---------------------+
| This node                   | appears in ...                              | Size      | Last update         |
+-----------------------------+---------------------------------------------+-----------+---------------------+
| 0._SQLfast-tutorial.tuto    |                                             |           |                     |
|                             | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials    |     4 489 | 2016-12-27 20:09:46 |
|                             | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials    |     4 489 | 2016-12-27 20:09:46 |
| 1._Introduction.tuto        |                                             |           |                     |
|                             | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials    |     3 496 | 2016-12-29 18:05:18 |
|                             | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials    |     3 496 | 2017-01-07 15:50:10 |
| ....                        |                                             |           |                     |
| Tuto14-showData-3col.gif    |                                             |           |                     |
|                             | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Tuto  |     9 283 | 2016-09-25 11:00:10 |
|                             | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Tuto  |     9 283 | 2016-09-25 11:00:10 |
| Welcome-Settings.gif        |                                             |           |                     |
|                             | D:/SQLfast/SQLfastHelp_EN/Images/Lev0-Help  |     6 876 | 2017-01-18 18:41:56 |
|                             | D:/SQLfast/SQLfastHelp_FR/Images/Lev0-Help  |     7 485 | 2017-01-18 18:43:49 |
| Welcome.help                |                                             |           |                     |
|                             | D:/SQLfast/SQLfastHelp_EN                   |     3 557 | 2017-01-24 14:40:22 |
|                             | D:/SQLfast/SQLfastHelp_FR                   |     4 395 | 2017-01-24 14:41:18 |
| __A-LIRE.txt                |                                             |           |                     |
|                             | D:/SQLfast/Databases                        |       348 | 2015-12-16 10:29:28 |
|                             | D:/SQLfast/Files                            |       157 | 2015-12-16 10:48:36 |
|                             | D:/SQLfast/Output                           |       158 | 2015-12-16 10:50:14 |
|                             | D:/SQLfast/SQLfastHelp_FR                   |       613 | 2015-12-16 10:41:24 |
|                             | D:/SQLfast/Scripts                          |       533 | 2016-12-31 16:27:22 |
|                             | D:/SQLfast/Scripts/BD-2015                  |       219 | 2015-12-16 10:18:52 |
|                             | D:/SQLfast/Scripts/SQLfast-Tutorials        |       244 | 2016-12-31 16:29:27 |
| __README.txt                |                                             |           |                     |
|                             | D:/SQLfast/Databases                        |       295 | 2015-12-16 10:29:26 |
|                             | D:/SQLfast/Files                            |        97 | 2015-12-16 10:48:30 |
|                             | D:/SQLfast/Output                           |       130 | 2015-12-16 10:55:58 |
|                             | D:/SQLfast/SQLfastHelp_EN                   |        95 | 2016-12-26 23:52:42 |
|                             | D:/SQLfast/SQLfastHelp_FR                   |       568 | 2015-12-16 10:41:34 |
|                             | D:/SQLfast/Scripts                          |       539 | 2016-12-31 16:26:16 |
|                             | D:/SQLfast/Scripts/BD-2015                  |       183 | 2016-12-31 16:23:54 |
|                             | D:/SQLfast/Scripts/SQLfast-Tutorials        |       239 | 2016-12-31 16:29:03 |
```

Figure 15.9 - Detailed report of potentially duplicate files

15.12 Application 5: Comparing two directory instances

This application compares the contents of two directory instances by identifying the components of the first one that are absent from the second one, and conversely.

This comparison particularly makes sense when the instances have similar structures, for example when we examine the evolution of two instances of the same directory or when we compare two backup external disks that are supposed to be identical.

Nodes are compared on their name and their type, but a more strict comparison can be chosen, that adds the *size* and *last update time* properties to the operands. In addition, the comparison can be limited to two subdirectories of the selected instances. This is shown in the parameter selection box of Figure 15.10.

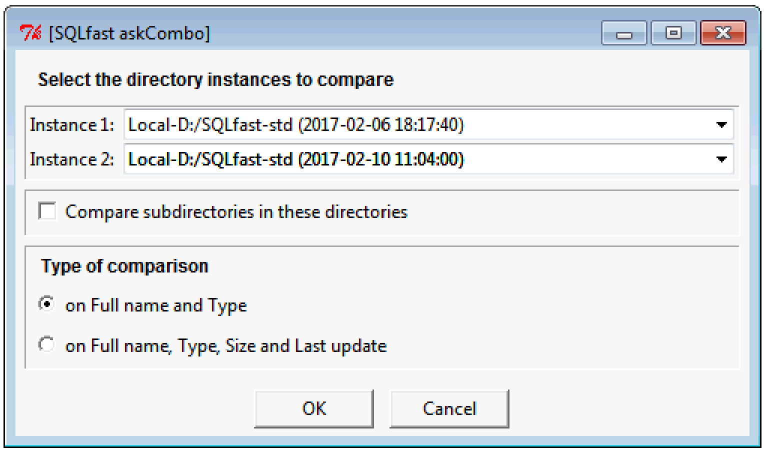
```
set inst = ;
set mul  = 2;
ask-u inst,mul = Instance:[!$instName$] | Min multiplicity:;
if ('$DIALOGbutton$' = 'Cancel' or '$inst$' = '') goto EXIT;

select
       "Node name" as "This node",
       Source as "appears in ...",
       Size,
       LastUpdate as "Last update"
from
   (select distinct
           case when NType = 'F'
                then LocalName
                else '['||LocalName||']'
           end as "Node name",
           '' as Source,
           '' as Size,
           '' as LastUpdate,
           LocalName||'A' as Sort
    from   NODE
    where  InstID = $inst$
    and    LocalName in (select LocalName from NODE
                         where  InstID = $inst$
                         group by LocalName
                         having count(*) >= $mul$)
        union
    select
           '' as "Node name",
           Prefix as Source,
           frame(thousandSep(Size,' '),13,'>',' ') as "Size",
           LastUpdate,
           LocalName||'B' as Sort
    from   NODE
    where  InstID = $inst$
    and    LocalName in (select LocalName from NODE
                         where  InstID = $inst$
                         group by LocalName
                         having count(*) >= $mul$)
   )
order by Sort;
```

Script 15.18 - Generating the hierarchical report of Figure 15.9

To clarify the algorithms, we decide to work on two temporary tables, NODE1 and NODE2, in which we store the sets of nodes to compare in both instances. The comparison can then be translated into the set theoretic difference of these tables (SQL operator `except`).

The fragment of Script 15.19 displays the local name nodes of NODE1 that are absent from NODE2, based on their values of FullName and NType.


[SQLfast askCombo]

Select the directory instances to compare

Instance 1: Local-D:/SQLfast-std (2017-02-06 18:17:40)
Instance 2: Local-D:/SQLfast-std (2017-02-10 11:04:00)

☐ Compare subdirectories in these directories

Type of comparison

◉ on Full name and Type
○ on Full name, Type, Size and Last update

Figure 15.10 - Comparing two directory instances

Expression `case-end` performs a pretty formatting of the node names according to their type. Variables **dir1** and **dir2** contains the full names of the directories to compare.

```
select NType as "Type",
       case when NType = 'F'
            then '  '||Name
            else '['||Name||']'
       end as "In Directory (1) but not in Directory (2)"
from  (select replace(FullName,'§dir1§/','') as Name,NType
       from   NODE1
          except
       select replace(FullName,'§dir2§/','') as Name,NType
       from   NODE2);
```

Script 15.19 - Displaying the nodes in NODE1 but not in NODE2

15.13 Application 6: Searching for clone files

So far, the concept of duplicate has been based on external properties of two files: same local name, same size and same last update time. This approach is far from reliable in practice since it may lead to two kinds of errors:

- *false positive*: two files happen to have the same external properties, though their contents are quite different
- *false negative*: two files have different names and last update time while their contents are identical.

15.13.1 The problem

To go beyond the mere comparison of external properties, we have to carry out a more in-depth examination of the files to identify those of them that are real *clones*, that is, that have the same contents. Let us adopt the following reasoning:

1. clones may have different local names (ORDERS.db and ORDERS-old.db) and different extension (ORDERS.db, ORDERS.back); so, comparing names and extensions may help detecting some clones but surely not in all cases
2. clones should have the same last update time but copying or downloading a file may change the update time of the copy
3. clones have the same size
4. clones have the same contents, compared byte by byte.

If we agree on these rules, then only the last two ones should be used to detect clones. If checking whether two files have the *same size* is fairly easy, the *same contents* rule is likely to be harder to implement, specially for large files (small file can be preloaded in RAM).

Considering **S** = {f1, f2, ..., fn-1, fn} a set of **n** candidate files. A procedure that compares two files at a time by reading them in parallel, has a best case cost of **2(n-1)** sequential readings if the files are real clones. Indeed, we have to compare the following pairs: (f1, f2), (f2, f3), ..., (fn-2, fn-1), (fn-1, fn); f1 and fn are read once, all the others twice. On the contrary, if the files of **S** are not clones, each pair of files must be evaluated, that is, **n(n-1)** file comparisons. Each of them requires two sequential readings, for a total cost of **2n(n-1)** sequential readings. However, these readings will be partial, since they will stop once a pair of different bytes have been read. If, on the average, these bytes are encountered in the middle of the files, then the estimated total cost will be **n(n-1)** full sequential readings.

If a new file is added and is reported in a further instance, we will have to carry out the comparison process again. To save expensive file processing, we could think of a way to record the comparison result. This would require a new table CLONE_SET in which each row denotes a set of clones already evaluated and dependent table CLONE_MEMBER each row of which references a NODE row member of this set. When a set of clones has been identified, it is recorded in these tables. This way, introducing a new file will entail less effort, since it must be compared with one member of each clone set.

To be honest, all this seems both very expensive and complicated.

15.13.2 Saved by (computer) science

Fortunately, computer science (and SQLfast as well!) provides us with a nice technique to compare the contents of files of any size: *hash functions*, and more particularly, *secure hash functions*.

This technique has been described in Section 15.7.3 (Secure hashing). It consists in deriving from a byte string **B**, here the contents of a file, another, short, fixed length, character string (the *hash value*) that can be used as a *fingerprint* or *signature* of **B**. With good hash functions, the probability of two different byte strings producing the same signature is extremely low. Therefore, we are allowed to consider that the hash value of a file (reasonably) identifies it among all the files, past, present and future, we intend to manage. SQLfast **hashFile** function and statement return, for any file, whatever its type and its length, a *hash value* consisting of a string of 256 bits converted into a string of 64 hexadecimal characters.

Now, the problem of searching for clones can be solved quite simply. We compute the hash value of each candidate file and we store it along with the other properties of these files. Clones are files with the same *hash value*.

15.13.3 Hashing files

First, we add a new column to table NODE[4]:

```
alter table NODE add column Hash char(64);
```

and we fill it for all the rows in instance **inst**:

```
update NODE set Hash = hashFile(FullName)
where  InstID = $inst$ and NType = 'F';
```

Function **hashFile** returns the hash value of the file denoted by its argument. However, it returns *null*[5] if this argument does not refer to an existing file (e.g., a directory, a link or a missing file).

An alternative iterative version, based on **hashFile** statement, is proposed in Script 15.21.

Performance

The hash functions and statement of SQLfast are based on SHA256, one of the best algorithms to generate near collision[6] free hash value distribution. What about its performance? Each call reads the sequence of bytes of a file and applies to them complex binary computation. On a standard, average, machine (laptop, i7, 7200 rpm HD), a directory instance of 2,700 files totalling 850 MB is processed in 8.5 s. So, the reading-computation speed is of 100 MB/s. Not bad, considering that this operation has to be performed only once per instance. Not surprisingly, the iterative version is a slower: 21.5 s. instead of 8.5 s.

4. Actually, it has already been defined in table NODE.
5. Constant `None` in Python, interpreted as `null` in SQLite. SQLfast statement **hashFile** returns an empty string instead of `None`.
6. *collision*: undesirable event of two different source files generating the same hash value, which could lead to false positive.

```
update NODE set Hash = hashFile(FullName)
where  InstID = $inst$ and NType = 'F';
```

Script 15.20 - Updating column Hash of table NODE: pure SQL version

```
for node,name = [select NodeID,FullName from NODE
                 where  InstID = $inst$ and NType = 'F'];
   hashFile hash =  $name$;
   update NODE set Hash = case when '$hash$' = ''
                               then null
                               else '$hash$'
                          end
   where  NodeID = '$node$';
endfor;
```

Script 15.21 - Updating column Hash of table NODE: iterative version

15.13.4 Identifying clone candidates in a directory instance

Basically, exploring directory instances to find clones is quite similar to the procedures based on local names, as we did so far, where we substitutes Hash for LocalName.

Let us first define **cloneHash**, an SQL pattern that denotes the set of Hash values that exist more than once in instance **inst** (Script 15.23, top). Condition `Hash is not null` copes with missing files and condition `Size > 0` discards empty files.

Extracting the list of clone files is now straightforward (Script 15.22).

```
select Hash,LocalName,Prefix
from   NODE where Hash in ($cloneHash$)
order by Hash;
```

Script 15.22 - List of clone files in instance **inst** (raw version)

Since the values of Hash are a bit long, we display excerpts of them:

```
substr(Hash,1,8)||'...'||substr(Hash,56,8) as "Hash"
```

We define pattern **`hashDigest`** with this expression to make queries more readable (Script 15.23, bottom).

```
set cloneHash = select Hash from NODE
                where  InstID = $inst$ and NType = 'F'
                and    Hash is not null and Size > 0
                group by Hash having count(*) > 1;

set hashDigest = substr(Hash,1,8)||'...'||substr(Hash,56,8);
```

Script 15.23 - Two patterns: set of the Hash values of clones in instance **inst** and condensed version of hash values

Following the sorting trick of Script 15.12, Script 15.24 produces the nice hierarchical report of Figure 15.11. The group name (alias Group) is meaningless, its only role being to create a two-level hierarchy.

The structure of this query is fairly simple. The first subquery of the `from` clause creates the heading of the clone groups. The second subquery creates one row per clone file.

```
select Hash as Group,LocalName as Clone,Prefix as Source
from (select $hashDigest$ as Hash,
             '' as LocalName,'' as Prefix,Hash||'C' as Sort
      from   ($cloneHash$)
         union
      select '' as Hash,
             LocalName,Prefix,Hash||'F' as Sort
      from   NODE
      where  Hash in ($cloneHash$) and InstID = $inst$
     )
order by Sort;
```

Script 15.24 - List of clone files in instance **inst** (hierarchical view)

15.13.5 Identifying clone candidates among two directory instances

The problem seems quite similar to that of finding clones in a single instance. One important distinction though: these instances should be of *different directories*. Indeed, two instances of the same directory will most of the time exhibit little differences, so that most files will be clones!

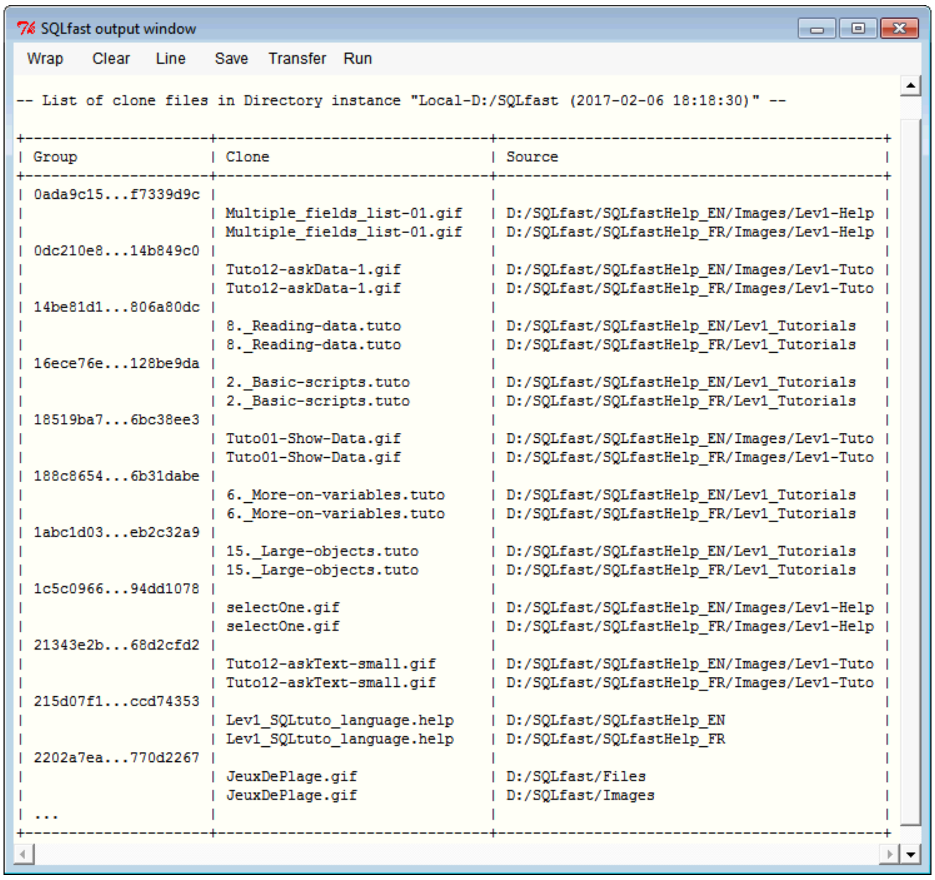
The approach we suggest consists in finding clones in the **union** of two source instances. This will work nicely provided (1) each instance has been cleaned from its own clones, or at least if these clones have been reduced as far as possible and (2) the instances have not too many common files. Otherwise a too large number of files will be declared clones, making the report of little interest.

To make the main query clearer, we first define pattern **cloneHash2** that computes the set of hash values common to both instances, denoted by variables inst1 and inst2 (Script 15.25). These values are those of candidate clones.

The main query is shown in Script 15.26. It creates a hierarchical report in which each set of clones coming from both instance are presented as a group. The main difference with the report of Figure 15.11 is that the source directory of each clone (its Prefix) is indexed with symbol <1> or <2> according to this source.

The query sorts three kinds of rows so that they appear in this order for each group of clones:

1. the group id (the concise Hash value (sort key = `Hash||'C0'`)
2. the clones from instance <1> (sort key = `Hash||'F1'`)
3. the clones from instance <2> (sort key = `Hash||'F2'`)



```
-- List of clone files in Directory instance "Local-D:/SQLfast (2017-02-06 18:18:30)" --

+---------------------+-------------------------------+--------------------------------------------+
| Group               | Clone                         | Source                                     |
+---------------------+-------------------------------+--------------------------------------------+
| 0ada9c15...f7339d9c |                               |                                            |
|                     | Multiple_fields_list-01.gif   | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Help |
|                     | Multiple_fields_list-01.gif   | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Help |
| 0dc210e8...14b849c0 |                               |                                            |
|                     | Tuto12-askData-1.gif          | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Tuto |
|                     | Tuto12-askData-1.gif          | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Tuto |
| 14be81d1...806a80dc |                               |                                            |
|                     | 8._Reading-data.tuto          | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials   |
|                     | 8._Reading-data.tuto          | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials   |
| 16ece76e...128be9da |                               |                                            |
|                     | 2._Basic-scripts.tuto         | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials   |
|                     | 2._Basic-scripts.tuto         | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials   |
| 18519ba7...6bc38ee3 |                               |                                            |
|                     | Tuto01-Show-Data.gif          | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Tuto |
|                     | Tuto01-Show-Data.gif          | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Tuto |
| 188c8654...6b31dabe |                               |                                            |
|                     | 6._More-on-variables.tuto     | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials   |
|                     | 6._More-on-variables.tuto     | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials   |
| 1abc1d03...eb2c32a9 |                               |                                            |
|                     | 15._Large-objects.tuto        | D:/SQLfast/SQLfastHelp_EN/Lev1_Tutorials   |
|                     | 15._Large-objects.tuto        | D:/SQLfast/SQLfastHelp_FR/Lev1_Tutorials   |
| 1c5c0966...94dd1078 |                               |                                            |
|                     | selectOne.gif                 | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Help |
|                     | selectOne.gif                 | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Help |
| 21343e2b...68d2cfd2 |                               |                                            |
|                     | Tuto12-askText-small.gif      | D:/SQLfast/SQLfastHelp_EN/Images/Lev1-Tuto |
|                     | Tuto12-askText-small.gif      | D:/SQLfast/SQLfastHelp_FR/Images/Lev1-Tuto |
| 215d07f1...ccd74353 |                               |                                            |
|                     | Lev1_SQLtuto_language.help    | D:/SQLfast/SQLfastHelp_EN                  |
|                     | Lev1_SQLtuto_language.help    | D:/SQLfast/SQLfastHelp_FR                  |
| 2202a7ea...770d2267 |                               |                                            |
|                     | JeuxDePlage.gif               | D:/SQLfast/Files                           |
|                     | JeuxDePlage.gif               | D:/SQLfast/Images                          |
| ...                 |                               |                                            |
+---------------------+-------------------------------+--------------------------------------------+
```

Figure 15.11 - Hierarchical view of the clone files of a directory instance

```
set cloneHash2 = select Hash
                 from (select Hash from NODE
                       where  InstID = $inst1$ and NType = 'F'
                       and    Hash is not null and Size > 0
                          intersect
                       select Hash from NODE
                       where  InstID = $inst2$ and NType = 'F'
                       and    Hash is not null and Size > 0);
```

Script 15.25 - Pattern computing the set of the hash values common to instances inst1 and inst2

```
select Hash as "Group",LocalName as Clone,Prefix as Source
from (select $hashDigest$ as "Hash",
             '' as LocalName,'' as Prefix,
             Hash||'C0' as Sort
      from   NODE
      where  Hash in ($cloneHash2$)
         union
      select '' as "Hash",
             LocalName,'<1>'||Prefix as Prefix,
             Hash||'F1' as Sort
      from   NODE
      where  Hash in ($cloneHash2$) and InstID = $inst1$
         union
      select '' as "Hash",
             LocalName,'<2>'||Prefix as Prefix,
             Hash||'F2' as Sort
      from   NODE
      where  Hash in ($cloneHash2$) and InstID = $inst2$
     )
order by Sort;
```

Script 15.26 - Identifying clones between two directory instances

15.14 A touch of optimization

No indexes have been created yet, apart from those automatically associated with primary keys (and unique keys, if any) by most RDBMSs.

Creating non unique indexes may accelerate some queries. However, deciding which ones will really be useful is not an easy task. Indeed, the benefit of an index depends on the size of the table, its selectivity (which measures the proportion of the table that can be accessed through it), the frequency of the queries that will use it and the cost of its updating. In addition, a missing index can be compensated by the access strategy of the DBMS, which can decide to dynamically create an index on the fly for the execution of a query.

Due to their small size, tables DIRECTORY and INSTANCE can be left without additional indexes.

Concerning table NODE, additional indexes will generally be useless for a population of a few thousands rows. Moreover, reducing execution time from 1 second to 100 milliseconds is of little interest when we want to produce, once in a while, a report that will require one minute to analyze.

A common sense rule suggests to index foreign keys, as these indexes will support such operations as joins with the parent table and delete/update in the parent table. This is quite justified in this case since most queries operate on the nodes of a definite instance. So, we create an index on InstID (Script 15.27). As to ParentID, we observe that only Script 15.8, that computes Path, includes a condition `is null` on this column. We suggest to discard this foreign key index.

An index on NType, though this column is mentioned in many conditions (Scripts 15.10, 15.12, 15.14, 15.16, 15.20, 15.21, 15.23 and 15.25), will be useless due to its low selectivity (two values only). Same argument and conclusion for an index on Level.

Several queries include `where`, `order by` or `group by` clauses on columns Hash, Path, FullName, LocalName and Extension, all in combination with InstID. Indexes on these columns may decrease the execution time of report generation queries but not by far.[7] Script 15.7, that computes ParentID values could make use on an index on (InstID, FullName). However, this operation already is very fast without such index.

```
create index ndxNodeInstID on NODE(InstID);
```

Script 15.27 - We just need one additional index

A last argument to be conservative about index creation is that each index on NODE entails a loading time penalty. So, the lesser, the best for this operation.

Of course, reporting on *very large* directory instances may change our conclusion.

7. On the example shown in this chapter, most reporting queries require less than 1 second with an index on InstID only.

15.15 Building the DIRECTORY application

The functions developed above have been integrated into a global application. The main program, called **DIR-MAIN.sql**, in directory **Scripts/Case_Studies/Cases_Directory_Manegement**. Its structure is that of a simple loop that displays the control panel of Figure 15.12 and executes the function selected.

Beside the analysis functions (right side column), some basic management functions have been integrated (left side column):

- creating a new DIRECTORY.db database
- loading a new instance of an existing or a new root directory
- inserting the hash values of an instance
- deleting a root directory and all its instances
- deleting an instance of a root directory and all its nodes
- opening the help window

The latter function display a small tutorial document (DIRECTORY.tuto) that describes the main functions of the application.

The mapping between the button checked and the script to execute is implemented through item list **modList**, which is made up of the names of the scripts separated by semi-colons. The number of the button checked by the user is returned in variable **op**. This value is used to get the module name through expression

```
itemS('$modList$',$op$)
```

that returns the **op**th item in list **modList**.[8]

The main script is shown in Script 15.28. Some comments on its main parts:

– [1]: sets the current script directory **scriptPath**

– [2]: loads the common patterns, collected in script DIRECTORY-Common-Patterns.sql

– [3]: builds the module name list in **modList**

– [4]: sets the default operation

– [5]: main loop

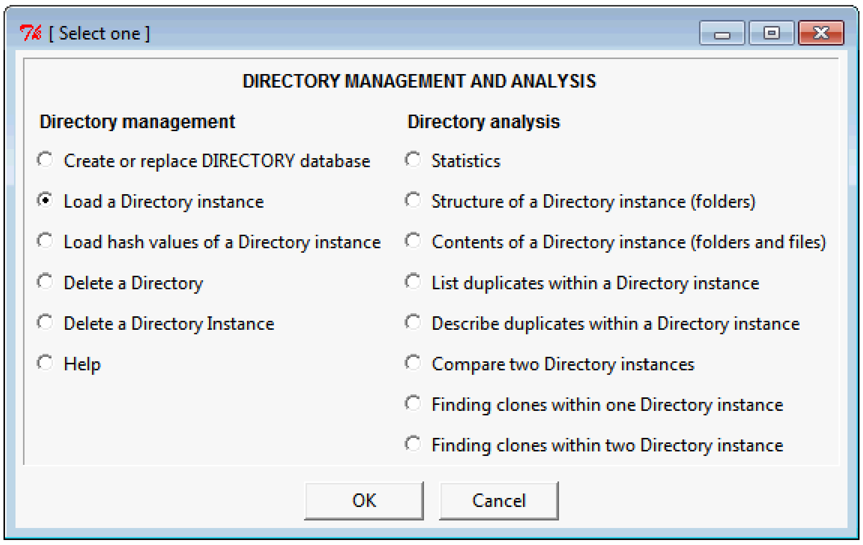
– [6]: creates and displays the control panel; returns operation number **op**

– [7]: extracts the selected module name in variable **module**

– [8]: executes the selected module.

The main script also comprises ancillary operations that are not shown here.

8. Suffix **S** refers to the standard separator, which is the semi-colon.


[Select one]

DIRECTORY MANAGEMENT AND ANALYSIS

Directory management	Directory analysis
○ Create or replace DIRECTORY database	○ Statistics
[✓ ● Load a Directory instance]	○ Structure of a Directory instance (folders)
○ Load hash values of a Directory instance	○ Contents of a Directory instance (folders and files)
○ Delete a Directory	○ List duplicates within a Directory instance
○ Delete a Directory Instance	○ Describe duplicates within a Directory instance
○ Help	○ Compare two Directory instances
	○ Finding clones within one Directory instance
	○ Finding clones within two Directory instance

Figure 15.12 - Control panel of the DIRECTORY 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.

15.16 Suggestions

This chapter has shown that simple tree structures can provide much interesting information on the contents of storage media. On the analysis side, a lot of other reporting queries can be developed, such as the following:

- comparing two media, to check whether they have the same contents, despite different structures
- comparing the structure of two instances
- identifying the files that have been renamed (comparing two successive instances)
- computing the size of each directory
- synthesizing the activity of some files across successive instances (based on varying update and access time); identifying dead (or archived) files.

On the other hand, directory structure modification can be developed, such as copying, deleting and renaming files.

```
set scriptPath = SQLfast-Tutorials/Chapter-48;                    [1]
execSQL $scriptPath$/DIRECTORY-Common-Patterns.sql;               [2]
set modList = Create-or-Replace;Load-Instance;                    [3]
set modList = $modList$;Load-Hash;Delete-Directory;
...
set modList = $modList$;Select-Clone-Candidates-2;                [3]
set op = 2;                                                       [4]
while (True);                                                     [5]
   selectOne-u op =                                               [6]
      [/b @t@tDIRECTORY MANAGEMENT AND ANALYSIS]
      {Directory management}
    | Create or replace DIRECTORY database
    | Load a Directory instance
    | Load hash values of a Directory instance
    | Delete a Directory
    | Delete a Directory Instance
    | Help
   || {Directory analysis}
    | Statistics
    | Structure of a Directory instance (folders)
    | Contents of a Directory instance (folders and files)
    | List duplicates within a Directory instance
    | Describe duplicates within a Directory instance
    | Compare two Directory instances
    | Finding clones within one Directory instance
    | Finding clones within two Directory instance;
   if ('$DIALOGbutton$' = 'Cancel') exit;
   compute module = itemS('$modList$',$op$);                      [7]
   execSQL $scriptPath$/DIRECTORY-$module$.sql;                   [8]
endwhile;                                                         [5]
```

Script 15.28 - Main script of DIRECTORY application