Grand Unified File Index (GUFI) SQL Guide

Table of Contents

[License Info 1](#_Toc132219646)

[Structure Reference 3](#_Toc132219647)

[Tables/Schema 3](#_Toc132219648)

[Useful Views 6](#_Toc132219649)

[Extensibility 9](#_Toc132219650)

[SQL Supplemental Functions 10](#_Toc132219651)

[The GUFI query basics 10](#_Toc132219652)

[Very Basic Demonstration of Query of Tables 11](#_Toc132219653)

[Advanced SQL queries 15](#_Toc132219654)

[Query output files and databases 18](#_Toc132219655)

[querydbs 19](#_Toc132219656)

[Compound Queries 20](#_Toc132219657)

[The Tree Summary Concept 23](#_Toc132219658)

[Managing Tree Summaries 23](#_Toc132219659)

[Rollups 26](#_Toc132219660)

[Attaching gufi\_query to external databases 29](#_Toc132219661)

## License Info

* # This file is part of GUFI, which is part of MarFS, which is released
* # under the BSD license.
* #
* #
* # Copyright (c) 2017, Los Alamos National Security (LANS), LLC
* # All rights reserved.
* #
* # Redistribution and use in source and binary forms, with or without modification,
* # are permitted provided that the following conditions are met:
* #
* # 1. Redistributions of source code must retain the above copyright notice, this
* # list of conditions and the following disclaimer.
* #
* # 2. Redistributions in binary form must reproduce the above copyright notice,
* # this list of conditions and the following disclaimer in the documentation and/or
* # other materials provided with the distribution.
* #
* # 3. Neither the name of the copyright holder nor the names of its contributors
* # may be used to endorse or promote products derived from this software without
* # specific prior written permission.
* #
* # THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
* # ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* # WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
* # IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
* # INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* # BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* # DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* # LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
* # OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
* # ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
* #
* #
* # From Los Alamos National Security, LLC:
* # LA-CC-15-039
* #
* # Copyright (c) 2017, Los Alamos National Security, LLC All rights reserved.
* # Copyright 2017. Los Alamos National Security, LLC. This software was produced
* # under U.S. Government contract DE-AC52-06NA25396 for Los Alamos National
* # Laboratory (LANL), which is operated by Los Alamos National Security, LLC for
* # the U.S. Department of Energy. The U.S. Government has rights to use,
* # reproduce, and distribute this software. NEITHER THE GOVERNMENT NOR LOS
* # ALAMOS NATIONAL SECURITY, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR
* # ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE. If software is
* # modified to produce derivative works, such modified software should be
* # clearly marked, so as not to confuse it with the version available from
* # LANL.
* #
* # THIS SOFTWARE IS PROVIDED BY LOS ALAMOS NATIONAL SECURITY, LLC AND CONTRIBUTORS
* # "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
* # THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* # ARE DISCLAIMED. IN NO EVENT SHALL LOS ALAMOS NATIONAL SECURITY, LLC OR
* # CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* # EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
* # OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* # INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* # CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
* # IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
* # OF SUCH DAMAGE.

## Structure Reference

The structure of the GUFI tree looks like the diagram in Figure 2 below.

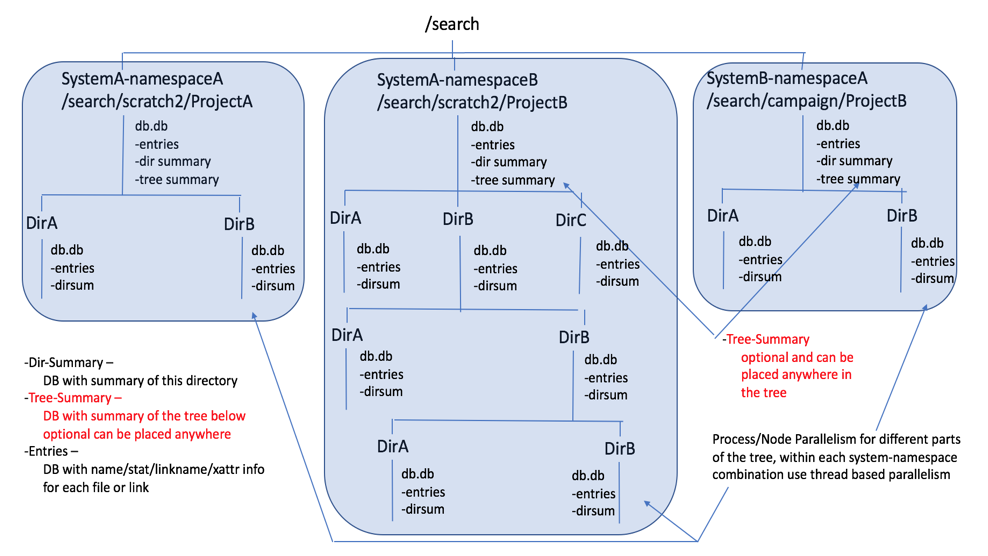


Figure 2 GUFI Overall Structure

The following diagram depicts how a source file/storage system metadata is extracted into a GUFI tree where file/link metadata is placed into a per directory GUFI database in Figure 3.

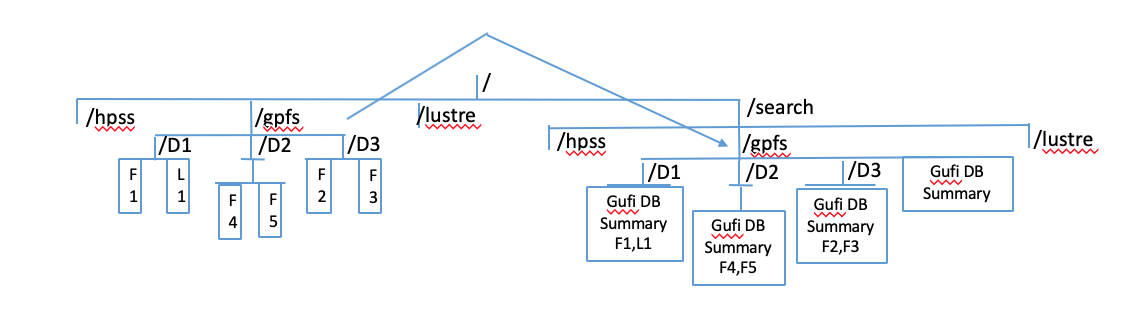


Figure 3 Mapping of source file system into GUFI tree

## Tables/Schema

The structure and function of the three types of sql record holding tables in each GUFI database are described below.

* The entries table houses extracted metadata for files and links within the corresponding directory.
  + "CREATE TABLE entries(name TEXT PRIMARY KEY, type TEXT, inode INT64, mode INT64, nlink INT64, uid INT64, gid INT64, size INT64, blksize INT64, blocks INT64, atime INT64, mtime INT64, ctime INT64, linkname TEXT, xattrs TEXT, crtime INT64, ossint1 INT64, ossint2 INT64, ossint3 INT64, ossint4 INT64, osstext1 TEXT, osstext2 TEXT);";
    - name TEXT character name of file or link
    - type TEXT character d for file or link
    - inode INT64 inode number from source system integer
    - mode INT64 unix mode bits integer
    - nlink INT64 unix number of links integer
    - uid INT64 unix uid integer
    - gid INT64 unix gid integer
    - size INT64 unix logical file size in bytes integer
    - blksize INT64 unix blksize for file integer
    - blocks INT64 unix number of blocks integer
    - atime INT64 unix access time epoch integer
    - mtime INT64 unix modificaton time epoch integer
    - ctime INT64 unix change time epoch integer
    - linkname TEXT unix string for link name character
    - xattrs TEXT concatenation of all extended attributes character
    - crtime INT64 create time epoch int64 (some file systems provide)
    - ossint1 INT64 storage system specific integer
    - ossint2 INT64 storage system specific integer
    - ossint3 INT64 storage system specific integer
    - ossint4 INT64 storage system specific integer
    - osstext1 TEXT storage system specific string character
    - osstext2 TEXT storage system specific string character
* The directory summary table houses extracted information about everything within the corresponding directory. It is a summary of that directory, including information about minimum file size, maximum file size, the number of files, etc.
  + "CREATE TABLE summary(name TEXT PRIMARY KEY, type TEXT, inode INT64, mode INT64, nlink INT64, uid INT64, gid INT64, size INT64, blksize INT64, blocks INT64, atime INT64, mtime INT64, ctime INT64, linkname TEXT, xattrs TEXT, totfiles INT64, totlinks INT64, minuid INT64, maxuid INT64, mingid INT64, maxgid INT64, minsize INT64, maxsize INT64, totltk INT64, totmtk INT64, totltm INT64, totmtm INT64, totmtg INT64, totmtt INT64, totsize INT64, minctime INT64, maxctime INT64, minmtime INT64, maxmtime INT64, minatime INT64, maxatime INT64, minblocks INT64, maxblocks INT64, totxattr INT64,depth INT64, mincrtime INT64, maxcrtime INT64, minossint1 INT64, maxossint1 INT64, totossint1 INT64, minossint2 INT64, maxossint2 INT64, totossint2 INT64, minossint3 INT64, maxossint3 INT64, totossint3 INT64,minossint4 INT64, maxossint4 INT64, totossint4 INT64, rectype INT64, pinode INT64), isroot INT64, rollupscore INT64);";
    - * name TEXT character name of directory
      * type TEXT character d for directory
      * inode INT64 inode number from source system integer
      * mode INT64 unix mode bits integer
      * nlink INT64 unix number of links integer
      * uid INT64 unix uid integer
      * gid INT64 unix gid integer
      * size INT64 unix logical file size in bytes integer
      * blksize INT64 unix blksize for file integer
      * blocks INT64 unix number of blocks integer
      * atime INT64 unix access time epoch integer
      * mtime INT64 unix modificaton time epoch integer
      * ctime INT64 unix change time epoch integer
      * linkname TEXT unix string for link name character
      * xattrs TEXT concatenation of all extended attributes character
      * totfiles INT64 number files
      * totlinks INT64 summed links
      * minuid INT64 min uid
      * maxuid INT64 max uid
      * mingid INT64 min gid
      * maxgid INT64 max gid
      * minsize INT64 min size
      * maxsize INT64 max size
      * totltk INT64 number <= 1024
      * totmtk INT64 number > 1024
      * totltm INT64 number <= 1048576
      * totmtm INT64 number > 1048576
      * totmtg INT64 number <= 1073741824
      * totmtt INT64 number > 1073741824
      * totsize INT64 summed size
      * minctime INT64 min ctime
      * maxctime INT64 max ctime
      * minmtime INT64 min mtime
      * maxmtime INT64 max mtime
      * minatime INT64 min atime
      * maxatime INT64 max atime
      * minblocks INT64 min blocks
      * maxblocks INT64 max blocks
      * totxattr INT64 number of files/links with xattrs present
      * depth INT64 dept (number of slashes in path)
      * mincrtime INT64 min create time
      * maxcrtime INT64 max create time
      * minossint1 INT64 min ossint1
      * maxossint1 INT64 max ossint1
      * totossint1 INT64 summed ossint1
      * minossint2 INT64 min ossint2
      * maxossint2 INT64 max ossint2
      * totossint2 INT64 summed ossint2
      * minossint3 INT64 min ossint3
      * maxossint3 INT64 max ossint3
      * totossint3 INT64 summed ossint3
      * minossint4 INT64 min ossint4
      * maxossint4 INT64 max ossint4
      * totossint4 INT64 summed ossint4
      * rectype INT64 0 for total for entire dir 1 for totals by user 2 for totals by group
      * pinode INT64 parent directory inode
      * isroot INT64 (used for rollup)
      * rollupscore INT64 (used for rollup)
* The (optional) tree summary table houses extracted summary of all summary information for all directories that live under the current directory. Like the

*directory summary* database, the *tree summary* database provides a summary, but rather than summarizing a directory, it summarizes everything in and below the directory in which the *tree summary* table is found. This is an optional table intended for further optimization of user queries, when necessary.

* + "CREATE TABLE treesummary(totsubdirs INT64, maxsubdirfiles INT64, maxsubdirlinks INT64, maxsubdirsize INT64, totfiles INT64, totlinks INT64, minuid INT64, maxuid INT64, mingid INT64, maxgid INT64, minsize INT64, maxsize INT64, totltk INT64, totmtk INT64, totltm INT64, totmtm INT64, totmtg INT64, totmtt INT64, totsize INT64, minctime INT64, maxctime INT64, minmtime INT64, maxmtime INT64, minatime INT64, maxatime INT64, minblocks INT64, maxblocks INT64, totxattr INT64,depth INT64, mincrtime INT64, maxcrtime INT64, minossint1 INT64, maxossint1 INT64, totossint1 INT64, minossint2 INT64, maxossint2 INT64, totossint2 INT64, minossint3 INT64, maxossint3 INT64, totossint3 INT64, minossint4 INT64, maxossint4 INT64, totossint4 INT64,rectype INT64, uid INT64, gid INT64);";
    - * totsubdirs INT64 number directories under this directory
      * maxsubdirfiles INT64 maximum number files in any subdir
      * maxsubdirlinks INT64 maximum number of links in any subdir
      * maxsubdirsize INT64 maximum summed size in any subdir
      * totfiles INT64 number files
      * totlinks INT64 summed links
      * minuid INT64 min uid
      * maxuid INT64 max uid
      * mingid INT64 min gid
      * maxgid INT64 max gid
      * minsize INT64 min size
      * maxsize INT64 max size
      * totltk INT64 number <= 1024
      * totmtk INT64 number > 1024
      * totltm INT64 number <= 1048576
      * totmtm INT64 number > 1048576
      * totmtg INT64 number <= 1073741824
      * totmtt INT64 number > 1073741824
      * totsize INT64 summed size
      * minctime INT64 min ctime
      * maxctime INT64 max ctime
      * minmtime INT64 min mtime
      * maxmtime INT64 max mtime
      * minatime INT64 min atime
      * maxatime INT64 max atime
      * minblocks INT64 min blocks
      * maxblocks INT64 max blocks
      * totxattr INT64 number of files/links with xattrs present
      * depth INT64 dept (number of slashes in path)
      * mincrtime INT64 min create time
      * maxcrtime INT64 max create time
      * minossint1 INT64 min ossint1
      * maxossint1 INT64 max ossint1
      * totossint1 INT64 summed ossint1
      * minossint2 INT64 min ossint2
      * maxossint2 INT64 max ossint2
      * totossint2 INT64 summed ossint2
      * minossint3 INT64 min ossint3
      * maxossint3 INT64 max ossint3
      * totossint3 INT64 summed ossint3
      * minossint4 INT64 min ossint4
      * maxossint4 INT64 max ossint4
      * totossint4 INT64 summed ossint4
      * rectype INT64 0 for total for entire tree 1 for totals by user 2 for totals by group
      * uid INT64 unix uid
      * gid INT64 unix gid

## Useful Views

* The *pentries* view provides parent inode as a query-able variable to the entries table. The reason this exists is that parent inodes are not stored because that would updating the index for moves of directories/files difficult, so parent inode is looked up in the summary so that parent inodes are never stored with child records to enable the “move” command in unix to move directories from one place to another and not having to reindex entire trees among other reasons.
  + "create view pentries as select entries.\*, summary.inode as pinode from entries, summary where rectype=0;";
* The vsummarydir view provides access to the entire directory summary and not a partial directory summary (say by user or group).
  + "create view vsummarydir as select \* from summary where rectype=0;";
* The vsummaryuser view provides access to the directory summary for each user (if this summary has been populated (not by default but easily populatable via a query)
  + "create view vsummaryuser as select \* from summary where rectype=1;";
* The vtsummarygroup view provides access to the directory summary for each group (if this summary has been populated (not by default but easily populatable via a query)
  + "create view vsummarygroup as select \* from summary where rectype=2;";
* The vtsummarydir view provides access to the entire tree directory summary and not a partial directory summary (say by user or group).
  + "create view vtsummarydir as select \* from treesummary where rectype=0;";
* The vtsummaryuser view provides access to the tree directory summary for each user (if this summary has been populated (not by default but easily populatable via a query)
  + "create view vtsummaryuser as select \* from treesummary where rectype=1;";
* The vtsummarygroup view provides access to the tree directory summary for each group (if this summary has been populated (not by default but easily populatable via a query)
  + "create view vtsummarygroup as select \* from treesummary where rectype=2;";

Probably the most important views you need to know as an sql user of GUFI are these two, vrsummary and vrpentries.

* vrsummary has the same information as the summary table (information about the directory but also has dname,sname,sroll
  + dname is the directory name always, not path but the actual directory name. This is true whether directories have been rolled up or not so if you always use vrsummary for refering to directory information you should also always use dname for the directory name
  + sname is the the directory name or the rolled up directory path based on whether a directory has been rolled up (you should stay away from this and use dname), sname is used in the rpath() function the best way you get the path from GUFI. There are other ways to get path but rpath is the most consistent and always works if you use vrsummary (which you should)
  + sroll is an indicator if the directory has been rolled up, its not important for you to know what this is or how it works but sroll is needed in the rpath() function, the best way to get path
  + rpath(sname,sroll) is always the way to get path in GUFI and sname and sroll exist in both the vrsummary view and the vrpentries view
  + vrsummary(dname,sname,sroll,name,type,inode,mode,nlink,uid,gid,size,blksize,blocks,atime,mtime,ctime,linkname,xattr\_names,totfiles,totlinks,minuid,maxuid,mingid,maxgid,minsize,maxsize,totltk,totmtk,totltm,totmtm,totmtg,totmtt,totsize,minctime,maxctime,minmtime,maxmtime,minatime,maxatime,minblocks,maxblocks,totxattr,depth,mincrtime,maxcrtime,minossint1,maxossint1,totossint1,minossint2,maxossint2,totossint2,minossint3,maxossint3,totossint3,minossint4,maxossint4,totossint4,rectype,pinode,isroot,rollupscore)
* vrpentries has the same information as the entries table and pentries view (information about files and links but also has almost all the fields from the containing directory for each entry (file/link) including of course dname, sname, and sroll
  + dname is the directory name always, not path but the actual directory name. This is true whether directories have been rolled up or not so if you always use vrpentries for refering to directory information for a particular file you should also always use dname for the directory name
  + sname is the the directory name or the rolled up directory path for a particular file based on whether a directory has been rolled up (you should stay away from this and use dname), sname is used in the rpath() function the best way you get the path from GUFI. There are other ways to get path but rpath is the most consistent and always works if you use vrpentries (which you should)
  + sroll is an indicator if the directory for a particular file has been rolled up, its not important for you to know what this is or how it works but sroll is needed in the rpath() function, the best way to get path
  + rpath(sname,sroll) is always the way to get path in GUFI and sname and sroll exist in both the vrsummary view and the vrpentries
  + it also has attop which designates if this file is in a directory at the top of a rollup
  + vrpentries(dname,sname,dmode,dnlink,duid,dgid,dsize,dblksize,dblocks,datime,dmtime,dctime,dlinkname,dtotfile,dtotlinks,dminuid,dmaxuid,dmingid,dmaxgidI,dminsize,dmaxsize,dtotltk,dtotmtk,totltm,dtotmtm,dtotmtg,dtotmtt,dtotsize,dminctime,dmaxctime,dminmtime,dmaxmtime,dminatime,dmaxatime,dminblocks,dmaxblocks,dtotxattr,ddepth,dmincrtime,dmaxcrtime,sroll,atroot,name,type,inode,mode,nlink,uid,gid,size,blksize,blocks,atime,mtime,ctime,linkname,xattr\_names,crtime,ossint1,ossint2,ossint3,ossint4,osstext1,osstext2,pinode)
  + Notice the containing directory fields typically start with a d. so sname is the containing directory name, dsize is the containing directory size, etc.

Its important to note that all of the information in pentries, vrpentries, and vrsummary is just materialized through views, the stored information is actually in entries and summary tables.

Again, you should do all your queries using vrsummary and vrentries because it will be consistent for rolled up and non rolled up trees and always use rpath() function to get path passing sname and sroll to that function.

## Extensibility

As you can see from the schemas from these tables, these fields below are unused by gufi base and are reserved for site specific use in all the tables.

* + - * minossint1 INT64 min ossint1
      * maxossint1 INT64 max ossint1
      * totossint1 INT64 summed ossint1
      * minossint2 INT64 min ossint2
      * maxossint2 INT64 max ossint2
      * totossint2 INT64 summed ossint2
      * minossint3 INT64 min ossint3
      * maxossint3 INT64 max ossint3
      * totossint3 INT64 summed ossint3
      * minossint4 INT64 min ossint4
      * maxossint4 INT64 max ossint4
      * totossint4 INT64 summed ossint4
      * osstext1 TEXT storage system specific string character
      * osstext2 TEXT storage system specific string character

You can of course add entirely new tables in each database and poplulate and as long as you have inode or name in that new table, you can join with any entries and summary and pentries tables/views.

To populate these fields/tables you can always use gufi\_query as root with -w to allow writing.

For example gufi\_query -n 10 -E’create table tree.newtable …..’ myindextree

\*\*\* notice the tree.table name, this is required when you create new tables/views in GUFI

Then perhaps gufi\_query -n 10 -E’insert into newtable blah blah ..’ myindextree

Remember if you only want to traverse a few levels, gufu\_query can be told how far down to traverse. Also remember you can always use

Find -type d -maxdepth -level | xarg \_\_\_\_ gufi\_query. To run it at just one level in the tree too.

Also, it is possible to add entirely new sqlite databases at each directory. We have contemplated and likely will enable this via the attach feature of sqlite to have gufi\_query attach to your custom database and then you can join with gufi db tables if you have join fields etc. If you desire this feature feel free to let us know.

## SQL Supplemental Functions

The following functions are added to the SQLite3 funcionality by GUFI:

* The uidtouser() function converts the Unix numeric userid to a user name
* The gidtogoup() function converts the Unix numeric groupid to a group name
* The modetotxt() function converts the Unix numeric mode to the human readlable drwxrwxrwx string
* Also, a handy function for converting unix epoch timestamps to date/time in SQLite3 is datetime(date variable like mtime or other,'unixepoch')
* Because paths are not stored in the databases because of the same difficulty of making updates like moves to the index difficult the following path related functions are available. These exist path functions exist but probably should not be used
  + The path() function provides the relative path to that entry (not including the entry name which is in the name variable)
  + The fpath() function provides the full path to that entry (not including the entry name which is in the name variable)
  + The epath() function provides the immediate directory name that the entry is in (not path just name)
* The rpath() is the recommend way to get path in GUFI passing in sname and sroll (rpath(sname,sroll) if you use vrsummary and/or vrpentries as you should.
* A neat trick is to get the file extension from the filename ltrim(NAME,rtrim(NAME,replace(NAME,".",""))) this will return the text after the last dot no matter how much text there is for a text field NAME

## The GUFI query basics

This is a simple test tree

ggrider@pn2201328 bin % find test

test

test/l1.2

test/l1.2/l2.2

test/l1.2/l2.2/f1.xls

test/l1.2/l2.3

test/l1.2/l2.3/f1.xls

test/l1.2/f2.1.doc

test/l1.2/l2.1

test/l1.2/l2.1/f1.xls

test/l1.3

test/l1.3/l2.2

test/l1.3/l2.2/f1.bigger

test/l1.3/l2.3

test/l1.3/l2.3/f1.big

test/l1.3/f3.1.doc

test/l1.3/l2.1

test/l1.3/l2.1/f1.biggest

test/l1.1

test/l1.1/l2.2

test/l1.1/l2.2/f1.exe

test/l1.1/l2.2/f1.tar

test/l1.1/l2.3

test/l1.1/l2.3/f1.exe

test/l1.1/l2.3/f1.tar

test/l1.1/f2.1.doc

test/l1.1/l2.1

test/l1.1/l2.1/f1.exe

test/l1.1/l2.1/f1.tar

test/l1.1/f1.1.doc

make a gufi index

ggrider@pn2201328 bin % ./gufi\_dir2index test testi

ggrider@pn2201328 bin % find testi

testi

testi/test

testi/test/l1.2

testi/test/l1.2/l2.2

testi/test/l1.2/l2.2/db.db

testi/test/l1.2/l2.3

testi/test/l1.2/l2.3/db.db

testi/test/l1.2/db.db

testi/test/l1.2/l2.1

testi/test/l1.2/l2.1/db.db

testi/test/l1.3

testi/test/l1.3/l2.2

testi/test/l1.3/l2.2/db.db

testi/test/l1.3/l2.3

testi/test/l1.3/l2.3/db.db

testi/test/l1.3/db.db

testi/test/l1.3/l2.1

testi/test/l1.3/l2.1/db.db

testi/test/l1.1

testi/test/l1.1/l2.2

testi/test/l1.1/l2.2/db.db

testi/test/l1.1/l2.3

testi/test/l1.1/l2.3/db.db

testi/test/l1.1/db.db

testi/test/l1.1/l2.1

testi/test/l1.1/l2.1/db.db

testi/test/db.db

See that there is a db.db in every directory, that is the gufi database per directory in the index tree.

### Very Basic Demonstration of Query of Tables

Please do not write queries against the tables directly, use vrsummary and vrpentries.

The entries table is where the files/links are kept

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select name,type,size,uid,gid,atime from entries;' -d ' ' testi/test

f2.1.doc f 2 2078 20 1680818366

f3.1.doc f 2 2078 20 1680818374

f2.1.doc f 2 2078 20 1680818358

f1.1.doc f 2 2078 20 1680818345

f1.xls f 0 2078 20 1680817472

f1.xls f 0 2078 20 1680817476

f1.xls f 0 2078 20 1680817469

f1.bigger f 7 2078 20 1680817673

f1.big f 4 2078 20 1680817657

f1.biggest f 10 2078 20 1680817701

f1.exe f 0 2078 20 1680817412

f1.tar f 0 2078 20 1680817430

f1.exe f 0 2078 20 1680817415

f1.tar f 0 2078 20 1680817424

f1.exe f 0 2078 20 1680817390

f1.tar f 0 2078 20 1680817439

The same thing is available using vrpentries

Please use this instead if you write queries

The entries table is where the files/links are kept

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select name,type,size,uid,gid,atime from vrpentries;' -d ' ' testi/test

f2.1.doc f 2 2078 20 1680818366

f3.1.doc f 2 2078 20 1680818374

f2.1.doc f 2 2078 20 1680818358

f1.1.doc f 2 2078 20 1680818345

f1.xls f 0 2078 20 1680817472

f1.xls f 0 2078 20 1680817476

f1.xls f 0 2078 20 1680817469

f1.bigger f 7 2078 20 1680817673

f1.big f 4 2078 20 1680817657

f1.biggest f 10 2078 20 1680817701

f1.exe f 0 2078 20 1680817412

f1.tar f 0 2078 20 1680817430

f1.exe f 0 2078 20 1680817415

f1.tar f 0 2078 20 1680817424

f1.exe f 0 2078 20 1680817390

f1.tar f 0 2078 20 1680817439

This is not really that useful though, what is the directory structure for it, what are the user names and dates etc. (use the gufi functions)

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select path(),name,type,size,uidtouser(uid),gidtogroup(gid),datetime(atime,"unixepoch"),modetotxt(mode) from vrpentries;' -d ' ' testi/test

testi/test/l1.2 f2.1.doc f 2 ggrider staff 2023-04-06 21:59:26 -rw-r--r--

testi/test/l1.3 f3.1.doc f 2 ggrider staff 2023-04-06 21:59:34 -rw-r--r--

testi/test/l1.1 f2.1.doc f 2 ggrider staff 2023-04-06 21:59:18 -rw-r--r--

testi/test/l1.1 f1.1.doc f 2 ggrider staff 2023-04-06 21:59:05 -rw-r--r--

testi/test/l1.2/l2.2 f1.xls f 0 ggrider staff 2023-04-06 21:44:32 -rw-r--r--

testi/test/l1.2/l2.3 f1.xls f 0 ggrider staff 2023-04-06 21:44:36 -rw-r--r--

testi/test/l1.2/l2.1 f1.xls f 0 ggrider staff 2023-04-06 21:44:29 -rw-r--r--

testi/test/l1.3/l2.2 f1.bigger f 7 ggrider staff 2023-04-06 21:47:53 -rw-r--r--

testi/test/l1.3/l2.3 f1.big f 4 ggrider staff 2023-04-06 21:47:37 -rw-r--r--

testi/test/l1.3/l2.1 f1.biggest f 10 ggrider staff 2023-04-06 21:48:21 -rw-r--r--

testi/test/l1.1/l2.2 f1.exe f 0 ggrider staff 2023-04-06 21:43:32 -rw-r--r--

testi/test/l1.1/l2.2 f1.tar f 0 ggrider staff 2023-04-06 21:43:50 -rw-r--r--

testi/test/l1.1/l2.3 f1.exe f 0 ggrider staff 2023-04-06 21:43:35 -rw-r--r--

testi/test/l1.1/l2.3 f1.tar f 0 ggrider staff 2023-04-06 21:43:44 -rw-r--r--

testi/test/l1.1/l2.1 f1.exe f 0 ggrider staff 2023-04-06 21:43:10 -rw-r--r--

testi/test/l1.1/l2.1 f1.tar f 0 ggrider staff 2023-04-06 21:43:59 -rw-r--r--

Well, that’s nicer, path() and the other build in functions make this way more useful.

However we highly recommend using rpath(sname,sroll) instead of path() because it will work in rollup trees and well as non-rollup trees

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),name,type,size,uidtouser(uid),gidtogroup(gid),datetime(atime,"unixepoch"),modetotxt(mode) from vrpentries;' -d ' ' testi/test

test/l1.2 f2.1.doc f 2 ggrider staff 2023-04-11 01:46:40 -rw-r--r--

test/l1.3 f3.1.doc f 2 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1 f1.1.doc f 2 ggrider staff 2023-04-11 01:46:40 -rw-r--r--

test/l1.1 f2.1.doc f 2 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.2/l2.2 f1.xls f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.2/l2.3 f1.xls f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.2/l2.1 f1.xls f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.2 f1.bigger f 7 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.3 f1.big f 4 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.1 f1.biggest f 10 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.2 f1.exe f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.2 f1.tar f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.3 f1.exe f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.3 f1.tar f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.1 f1.exe f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1/l2.1 f1.tar f 0 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

Another neat feature is you can stack multiple sql statements inside -T, -S, -E clauses

See this:

ggrider@pn2201328 bin % ./gufi\_query -n1 -S'select "D-",rpath(sname,sroll),name from vrsummary;select "F-",rpath(sname,sroll),name from vrpentries;' -d " " testi/test

D- test test

D- test/l1.2 l1.2

F- test/l1.2 f2.1.doc

D- test/l1.3 l1.3

F- test/l1.3 f3.1.doc

D- test/l1.1 l1.1

F- test/l1.1 f1.1.doc

F- test/l1.1 f2.1.doc

D- test/l1.2/l2.2 l2.2

F- test/l1.2/l2.2 f1.xls

D- test/l1.2/l2.3 l2.3

F- test/l1.2/l2.3 f1.xls

D- test/l1.2/l2.1 l2.1

F- test/l1.2/l2.1 f1.xls

D- test/l1.3/l2.2 l2.2

F- test/l1.3/l2.2 f1.bigger

D- test/l1.3/l2.3 l2.3

F- test/l1.3/l2.3 f1.big

D- test/l1.3/l2.1 l2.1

F- test/l1.3/l2.1 f1.biggest

D- test/l1.1/l2.2 l2.2

F- test/l1.1/l2.2 f1.exe

F- test/l1.1/l2.2 f1.tar

D- test/l1.1/l2.3 l2.3

F- test/l1.1/l2.3 f1.exe

F- test/l1.1/l2.3 f1.tar

D- test/l1.1/l2.1 l2.1

F- test/l1.1/l2.1 f1.exe

F- test/l1.1/l2.1 f1.tar

Notice it runs the -S before running the -E query for each directory, this is technically what we call a compound query which is described in detail later.

Can we use the where clause?

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),name,type,size,uidtouser(uid),gidtogroup(gid),datetime(atime,"unixepoch"),modetotxt(mode) from vrpentries where size>6;' -d ' ' testi/test

test/l1.3/l2.2 f1.bigger f 7 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.1 f1.biggest f 10 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

notice only files bigger than 6 bytes

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),name,type,size,uidtouser(uid),gidtogroup(gid),datetime(atime,"unixepoch"),modetotxt(mode) from vrpentries where name like "%big%";' -d ' ' testi/test

test/l1.3/l2.2 f1.bigger f 7 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.3 f1.big f 4 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.3/l2.1 f1.biggest f 10 ggrider staff 2023-04-11 01:46:39 -rw-r--r—

Notice only files with big in the name

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),name,type,size,uidtouser(uid),gidtogroup(gid),datetime(atime,"unixepoch"),modetotxt(mode) from vrpentries where dname like "%l1%";' -d ' ' testi/test

test/l1.2 f2.1.doc f 2 ggrider staff 2023-04-11 01:46:40 -rw-r--r--

test/l1.3 f3.1.doc f 2 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

test/l1.1 f1.1.doc f 2 ggrider staff 2023-04-11 01:46:40 -rw-r--r--

test/l1.1 f2.1.doc f 2 ggrider staff 2023-04-11 01:46:39 -rw-r--r--

Notice all the files that are in a directory with l1 in the name, this is very powerful and a feature of using vrpentries, you could ask for file info about files that are in directories with any attributes from the directory without a join (we did the join for you)

Of course queries against the summary table are similarly powerful.

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),dname,totfiles,maxsize from vrsummary;' -d ' ' testi/test

test test 0 -9223372036854775808

test/l1.2 l1.2 1 2

test/l1.3 l1.3 1 2

test/l1.1 l1.1 2 2

test/l1.2/l2.2 l2.2 1 0

test/l1.2/l2.3 l2.3 1 0

test/l1.2/l2.1 l2.1 1 0

test/l1.3/l2.2 l2.2 1 7

test/l1.3/l2.3 l2.3 1 4

test/l1.3/l2.1 l2.1 1 10

test/l1.1/l2.2 l2.2 2 0

test/l1.1/l2.3 l2.3 2 0

test/l1.1/l2.1 l2.1 2 0

Notice the top directory didn’t have any files so no maxsize.

## Advanced SQL queries

So can we use the limit clause in sql?

Yes, BUT remember we are running a query for each dir, so the limit clause limits each query, not the total output

What about order by and group by? Same problem exists, it will be ordered or grouped by in every directory separately.

ggrider@pn2201328 bin % ./gufi\_query -n1 -E'select rpath(sname,sroll),name,size from vrpentries limit 1;' -d ' ' testi/test

test/l1.2 f2.1.doc 2

test/l1.3 f3.1.doc 2

test/l1.1 f1.1.doc 2

test/l1.2/l2.2 f1.xls 0

test/l1.2/l2.3 f1.xls 0

test/l1.2/l2.1 f1.xls 0

test/l1.3/l2.2 f1.bigger 7

test/l1.3/l2.3 f1.big 4

test/l1.3/l2.1 f1.biggest 10

test/l1.1/l2.2 f1.exe 0

test/l1.1/l2.3 f1.exe 0

test/l1.1/l2.1 f1.exe 0

Yes, a single file per directory (limit 1)

But we really want to limit the entire output, or sort the entire output or group by the total output, etc

Well also notice that up until now I have been using -n1. One thread to query, ouch for big queries. That is because if you use > 1 thread while the query will run, the output will get jumbled because many threads writing to stdout.

How do we deal with this?

With interim per thread output tables and aggregate table (in memory or in /tmp space temp db’s tables) you can solve this problem.

ggrider@pn2201328 bin % ./gufi\_query -n3 -I'create table out (oname TEXT);' -E'insert into out select rpath(sname,sroll) || "/" || name from vrpentries;' -K'CREATE TABLE aggregate (fname TEXT);' -J 'insert into aggregate (fname) select oname from out;' -G'select fname from aggregate;' -d ' ' testi/test

test/l1.3/f3.1.doc

test/l1.2/l2.3/f1.xls

test/l1.3/l2.2/f1.bigger

test/l1.1/l2.3/f1.exe

test/l1.1/l2.3/f1.tar

test/l1.1/f1.1.doc

test/l1.1/f2.1.doc

test/l1.2/l2.1/f1.xls

test/l1.3/l2.3/f1.big

test/l1.1/l2.1/f1.exe

test/l1.1/l2.1/f1.tar

test/l1.2/f2.1.doc

test/l1.2/l2.2/f1.xls

test/l1.3/l2.1/f1.biggest

test/l1.1/l2.2/f1.exe

test/l1.1/l2.2/f1.tar

./gufi\_query -n3 -I'create table out (oname TEXT);' -E'insert into out select rpath(sname,sroll) || "/" || name from vrpentries;' -K'CREATE TABLE aggregate (fname TEXT);' -J 'insert into aggregate (fname) select oname from out;' -G'select fname from aggregate;' -d ' ' testi/test

Notice -n3

-I creates an output table per thread for each thread to write results to, notice -E now is insert into select,

Notice the fancy sqlite || concat function

-K create aggregate table, -j insert from all the per thread out tables into the aggregate table, then -G query the aggregate table to see my output

-J does the query you want but puts the result into the output db

-G pulls the output db records and concats them into the aggregation table

Wow, this is pretty fancy.

So it enabled -n3 but what else could it do?

ggrider@pn2201328 bin % ./gufi\_query -n3 -I'create table out (oname TEXT);' -E'insert into out select rpath(sname,sroll) || "/" || name from vrpentries;' -K'CREATE TABLE aggregate (fname TEXT);' -J 'insert into aggregate (fname) select oname from out;' -G'select fname from aggregate order by fname asc;' -d ' ' testi/test

test/l1.1/f1.1.doc

test/l1.1/f2.1.doc

test/l1.1/l2.1/f1.exe

test/l1.1/l2.1/f1.tar

test/l1.1/l2.2/f1.exe

test/l1.1/l2.2/f1.tar

test/l1.1/l2.3/f1.exe

test/l1.1/l2.3/f1.tar

test/l1.2/f2.1.doc

test/l1.2/l2.1/f1.xls

test/l1.2/l2.2/f1.xls

test/l1.2/l2.3/f1.xls

test/l1.3/f3.1.doc

test/l1.3/l2.1/f1.biggest

test/l1.3/l2.2/f1.bigger

test/l1.3/l2.3/f1.big

This ordered by the entire path because we did this in the insert query. insert into out select rpath(sname,sroll) || "/" || name

We concatenated path and name into one field and then inserted that into the output databases and then aggregated into fname.

This is just a feature of SQL of course but powerful none-the-less.

How about something really interesting like this?

ggrider@pn2201328 bin % ./gufi\_query -n1 -I'create table out (oext TEXT,ocnt INT64);' -E'insert into out select ltrim(name,rtrim(name,replace(name,".",""))) as myext,count(inode) from entries group by myext;' -K'CREATE TABLE aggregate (fext TEXT,fcnt INT64);' -J 'insert into aggregate (fext,fcnt) select oext,ocnt from out;' -G'select fext,sum(fcnt) from aggregate group by fext;' -d ' ' testi/test

big 1

bigger 1

biggest 1

doc 4

exe 3

tar 3

xls 3

Ok, what is this?

First, we make an output table per thread out with oext an ocnt

Then we insert into out a query against entries that counts the number of files for each file extension (after the last dot). Yes ltrim(name,rtrim(name,replace(name,".",""))) is a fancy way to get everything after the last dot. It replaces all the dots in the name then it rtrims to get a string on the left side of the last dot then ltrim to get rid of that part – there may be more elegant things to do here (looks like there is an open src set of functions for this we don’t have maybe we should get <https://observablehq.com/@asg017/introducing-sqlite-path> )

So each thread has a table with all the extensions and counts that the thread encountered in its work.

Then we make an aggregate table with fext and fcnt

Then we load up the aggregate table from all the thread tables

Then we query the aggregate table but instead of count, we sum group by fext and presto, we have a list of the file extensions and the total number of files across the entire tree. We tried to let the threads do as much work as they could and just let the aggregate work do the last sum/group by.

Yes indeed, this is getting very powerful now, imagine how hard that would be with find and sort and merge.

## Query output files and databases

Queries can be used to produce output files as well. This query select all fields from the entries tables and writes them to output files (one per thread) and in this query there are two threads.

./gufi\_query -n 2 -o outq -E "select rpath(sname,sroll),type,size from vrpentries;" testi/test

ls -l outq\*

-rw-r--r-- 1 ggrider staff 78 Dec 10 20:19 outq.0

-rw-r--r-- 1 ggrider staff 181 Dec 10 20:19 outq.1

ggrider@pn2201328 bin % cat outq.0

testi/test/l1.2f2

testi/test/l1.1f2

testi/test/l1.1f2

testi/test/l1.3/l2.3f4

testi/test/l1.2/l2.2f0

testi/test/l1.2/l2.1f0

testi/test/l1.1/l2.3f0

testi/test/l1.1/l2.3f0

ggrider@pn2201328 bin % cat outq.1

testi/test/l1.3f2

testi/test/l1.3/l2.2f7

testi/test/l1.3/l2.1f10

testi/test/l1.2/l2.3f0

testi/test/l1.1/l2.2f0

testi/test/l1.1/l2.2f0

testi/test/l1.1/l2.1f0

testi/test/l1.1/l2.1f0

As you can see you get your output into one file per thread. Combining these results is trivial such as cat outq.\* > myoutput

Another very powerful capability is to produce an output database per thread. This allows you to do queries and produce subsets of information that you can then run further queries on the result set. This is particularly useful when you want to use order by or group by clauses across all results.

Query to output databases requires the use of the -I parameter to create the output table and you must use the insert into SQL statements to populate the output database.

This query looks at all files and counts the number of files and links and sums the bytes in all the files and links per directory. It uses the -I for creation of the output databases and -O for the path base name to the output databases

rm outdb\*

./gufi\_query -n 2 -O outdb -I "create table sument (uid int64, type text, totf**il**es int64, totsize int64);" -E "insert into sument select uid, type, count(\*), sum(size) from vrpentries group by type;" testi/test

ggrider@pn2201328 bin % ls outdb\*

outdb.0 outdb.1

As you can see it created two outdb databases. The following sqlite3 utility confirms there is a table sument in both and one database has 2 records, one for each directory processed by that thread with total number and total size of files in each directory. The other database processed one directory.

ggrider@pn2201328 bin % sqlite3 outdb.0

sqlite> .tables

sument

sqlite> select \* from sument;

2078|f|1|2

2078|f|2|4

2078|f|1|0

2078|f|1|0

2078|f|1|4

2078|f|2|0

ggrider@pn2201328 bin % sqlite3 outdb.1

sqlite> .tables

sument

sqlite> select \* from sument;

2078|f|1|2

2078|f|1|0

2078|f|1|7

2078|f|1|10

2078|f|2|0

2078|f|2|0

### 

### querydbs

To deal with the fact that you have multiple output databases, the querydbs utility is available that glues the output databases together and union’s the tables together to allow you to query the output databases as if they were one database. This is very powerful as you can then do any query you want on the result set. Also, you can see there are 4 variables, uid, type, totfiles, totsize as the bfq query instructed to be produced.

In this invocation of querydbs, we tell it to list headings and records, attach to the two databases and select \* from vsument. The union of all the databases/tables is a view named v”table” so in this case sument tables are union’ed into vsument.

You can see it put the two databases/tables together and ran the query over the entire result set.

Of course you might want to do group by if you were producing a summary per user or something. In this case we just use order by to show you can sort the result set on totfiles.

ggrider@pn2201328 bin % ./querydbs -NV -d " " outdb sument "select \* from vsument order by totsize desc" outdb.\*

uid type totfiles totsize

2078 f 1 10

2078 f 1 7

2078 f 2 4

2078 f 1 4

2078 f 1 2

2078 f 1 2

2078 f 1 0

2078 f 1 0

2078 f 2 0

2078 f 1 0

2078 f 2 0

2078 f 2 0

query returned 12 records

As instructed, it sorted by totfiles. Yes this is a contrived example but this is a powerful concept where you want to group/order results or just further query/reduce etc.

This tool can also be used to just examine one normal gufi database

ggrider@pn2201328 bin % ./querydbs -NV -d " " vdb summary "select \* from summary" testi/test/db.db

This can be helpful in debugging issues.

name type inode mode nlink uid gid size blksize blocks atime mtime ctime linkname xattr\_names totfiles totlinks minuid maxuid mingid maxgid minsize maxsize totltk totmtk totltm totmtm totmtg totmtt totsize minctime maxctime minmtime maxmtime minatime maxatime minblocks maxblocks totxattr depth mincrtime maxcrtime minossint1 maxossint1 totossint1 minossint2 maxossint2 totossint2 minossint3 maxossint3 totossint3 minossint4 maxossint4 totossint4 rectype pinode isroot rollupscore

test d 55288125 16877 5 2078 20 160 4096 0 1680817140 1680817127 1680817127 (null) 0 0 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 0 0 0 0 0 0 0 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 0 0 9223372036854775807 -9223372036854775808 9223372036854775807 -9223372036854775808 0 9223372036854775807 -9223372036854775808 0 9223372036854775807 -9223372036854775808 0 9223372036854775807 -9223372036854775808 0 0 0 1 0

query returned 1 records

## Compound Queries

Compound queries is yet another powerful concept that can be used within the bfq program. Recall that you can control the SQL related work in bfq with the following options:

-T <SQL\_tsum> SQL for tree-summary table

-S <SQL\_sum> SQL for summary table

-E <SQL\_ent> SQL for entries table

-O <out\_DB> output DB one per thread

-I <SQL\_init> SQL init

-F <SQL\_fin> SQL cleanup

It is important to understand the order in which these operations occur to understand the power of compound queries.

The order is as follows:

* Output DB’s opened if called for one per thread
* Single SQL statement run (SQL\_init) once per thread
* Loop: Walk the GUFI tree and assign directories to threads, so a thread handles one full directory at a time
  + Thread works its way through a single directory
  + Multiple SQL statement run once for the directory SQL\_tsum
  + If overall “and” flag is off (an “or” condition) or if final SQL statement in SQL\_tsum produces < 1 records
    - Multiple SQL statement run once for the directory SQL\_sum
    - If overall “and” flag is off (an “or” condition) or if final SQL statement in SQL\_sum produces < 1 records
      * Records are put into the output
      * Multiple SQL statement run once for the directory SQL\_ent
      * If overall “and” flag is off (an “or” condition) or if final SQL statement in SQL\_ent produces < 1
        + Records are put into the output
      * endif
    - endif
  + endif
* endloop
* Single SQL statement run (SQL\_fin) once per thread

As you can see that if you want to run many sql statements you can run them with branches based on outcome. These sql statements can really be anything, they don’t even have to involved GUFI databases solely for that matter. If you want to use the “and” logic the last sql statement in a group must retrieve at least one record.

Within each group (SQL\_tsum, SQL\_sum, SQL\_ent) there can be multiple SQL statements separated by semicolons, so you can be as innovative as you would like.

As you can see, this flexibility to run many SQL statements one time per thread and many times per directory is a very powerful concept.

Now we should look at compound queries in our example data, this is using the -a flag in gufi\_query

You can do a query that uses

-S’select something about summary.’

And also

-E’select something about entries.”

If you use -a it means that it will run the -S query and then the -E query for every directory (this is a compound “or”

But if you don’t use the -a it is a compound “and” which means that it will only run the -E query if the -S query retrieves at least one row

ggrider@pn2201328 bin % ./gufi\_query -n1 -S'select "dir",rpath(sname,sroll),dname from vrsummary where dname like "%l2%";' -E'select name,size from vrpentries;' -d " " -a testi/test

f2.1.doc 2

f3.1.doc 2

f2.1.doc 2

f1.1.doc 2

dir testi/test/l1.2/l2.2 l2.2

f1.xls 0

dir testi/test/l1.2/l2.3 l2.3

f1.xls 0

dir testi/test/l1.2/l2.1 l2.1

f1.xls 0

dir testi/test/l1.3/l2.2 l2.2

f1.bigger 7

dir testi/test/l1.3/l2.3 l2.3

f1.big 4

dir testi/test/l1.3/l2.1 l2.1

f1.biggest 10

dir testi/test/l1.1/l2.2 l2.2

f1.exe 0

f1.tar 0

dir testi/test/l1.1/l2.3 l2.3

f1.exe 0

f1.tar 0

dir testi/test/l1.1/l2.1 l2.1

f1.exe 0

f1.tar 0

you see there are some directories missing (because they don’t match the like “%l2%”

but without the -a

ggrider@pn2201328 bin % ./gufi\_query -n1 -S'select "dir",rpath(sname,sroll),dname from vrsummary where dname like "%l2%";' -E'select name,size from vrpentries;' -d " " testi/test

dir testi/test/l1.2/l2.2 l2.2

f1.xls 0

dir testi/test/l1.2/l2.3 l2.3

f1.xls 0

dir testi/test/l1.2/l2.1 l2.1

f1.xls 0

dir testi/test/l1.3/l2.2 l2.2

f1.bigger 7

dir testi/test/l1.3/l2.3 l2.3

f1.big 4

dir testi/test/l1.3/l2.1 l2.1

f1.biggest 10

dir testi/test/l1.1/l2.2 l2.2

f1.exe 0

f1.tar 0

dir testi/test/l1.1/l2.3 l2.3

f1.exe 0

f1.tar 0

dir testi/test/l1.1/l2.1 l2.1

f1.exe 0

f1.tar 0

and here both directories and files are missing because the “and” cause some -E queries to not be run

## The Tree Summary Concept

The (optional) tree summary table houses extracted summary of all summary information for all directories that live under the current directory. Like the *directory summary* database, the *tree summary* database provides a summary, but rather than summarizing a directory, it summarizes everything in and below the directory in which the *tree summary* table is found. This is an optional table intended for further optimization of user queries, when necessary. See the Compound Query section for details.

gufi\_treesummary is the command used to place a tree summary record into a GUFI database. It is recommended that the GUFI administrator pick some level within the GUFI tree to keep tree summary records, perhaps at the top of the tree for each user to summarize that user or project. If updates occur to the GUFI tree in the directories below where the summary record has been written, you must delete that record and reproduce it using gufi\_treesummary.

gufi\_treesummary is typically very fast as it just walks the tree and processes the directory summary record for every subdirectory and then writes a new tree summary record.

### Managing Tree Summaries

Deployment sites that want to use the tree summary concept may want to determine where they want to use tree summaries at in their trees. It makes no sense to deploy tree summary tables very low in the tree as this would not save you any work when doing traversal based queries. Often sites have places where they know most queries will be looking/run from and that is likely going to guide where tree summaries should be placed. There might be optimal places for tree summaries for storage administrators that are higher in the trees than for users.

You may want tree summaries at multiple places in the tree. Often for large parts of the trees you will want tree summaries to be placed at the same depth in the tree structure. Currently there is no automated way to negate tree summary tables automatically when using incremental or full updates to a gufi tree because this is very likely going to depend on how Gufi is deployed. There are tools for managing tree summary tables though and this section lists some of those tools/methods.

if you want to add a tree summary at a single place in the GUFI tree you can just do a gufi\_treesummary -n threads directory where you want the summary tree put as described above.

If you want to add a tree summary at a single level, say 2 levels below the top

you can just use the find command to run gufi\_treesummary on every directory at some depth in the tree like this:

ggrider@pn2201328 bin % find testi/test -type d -maxdepth 1 -depth 1 -exec ./gufi\_treesummary -n 1 {} \;

… a lot of output …

Where did we put those treesummaries?

ggrider@pn2201328 bin % find testi/test -type d -maxdepth 1 -depth 1

testi/test/l1.2

testi/test/l1.3

testi/test/l1.1

If your find supports -P you could run multiple processes and of course you could use xargs as well

Now lets see what info is in this handy table

Just a few fields to show you that

ggrider@pn2201328 bin % ./gufi\_query -n1 -S'select path(),"mtotsubdirs",totsubdirs,"mmaxsubdirfiles",maxsubdirfiles,"mmaxsubdirsize",maxsubdirsize,"mtottreefiles",totfiles,"mdepth",depth from treesummary;' -y1 -z2 -d " " testi/test

testi/test/l1.2 mtotsubdirs 3 mmaxsubdirfiles 1 mmaxsubdirsize 2 mtottreefiles 4 mdepth 2

testi/test/l1.3 mtotsubdirs 3 mmaxsubdirfiles 1 mmaxsubdirsize 10 mtottreefiles 4 mdepth 2

testi/test/l1.1 mtotsubdirs 3 mmaxsubdirfiles 2 mmaxsubdirsize 4 mtottreefiles 8 mdepth 2

Notice we had to limit where the query ran using -y and -z so we only run the treesummary query against the level we put the treesummary at

if you want multiple processes running simultaneously

find gufitest/ -type d -maxdepth 1 -depth 1 -p 4 -exec  gufi\_treesummary -n 4 {} \;

Of course if you want to add tree summaries at multiple depths it makes the most sense to add the lowest depth first and move up the tree from there on desired levels of tree summaries. This is because gufi\_treesummary has a feature that it will use tree summaries below to create a new higher level tree summary for efficiency. This means you need to replace the lowest tree summary first and so on up the tree in order for gufi\_treesummary to produce correct tree summaries should you choose to put tree summaries at multiple levels in the tree.

if you want to find all the places you have tree summaries just use bfq and use -S to see if a treesummary table exists in that gufi db.

ggrider@pn2201328 bin % ./gufi\_query -n1 -S'select path(),name from tree.sqlite\_master where type="table" and name="treesummary"' testi/test

testi/test/l1.2treesummary

testi/test/l1.3treesummary

testi/test/l1.1treesummary

Notice we used path() here because we aren’t using vrsummary or vrpentries tables which have sname and sroll in them. Notice we used tree.sqlite\_master which is the master sqlite table in GUFI.

If you want to remove the tree summary tables from a single level simply remove the treesummary table and its views.

find gufitest/ -type f -maxdepth 1 -depth 1 -exec sqlite3 -line {} ‘drop table if exists vtsummarydir’ \;

find gufitest/ -type f -maxdepth 2 -depth 2 -exec sqlite3 -line {} ‘drop table if exists vtsummaryuser’ \;

find gufitest/ -type f -maxdepth 2 -depth 2 -exec sqlite3 -line {} ‘drop table if exists vtsummarygroup’ \;

find gufitest/ -type f -maxdepth 2 -depth 2 -exec sqlite3 -line {} ‘drop table if exists treesummary’ \;

or you could do this to get rid of them all in a tree or subtree

ggrider@pn2201328 bin % ./gufi\_query -n1 -S' drop vew if exists tree.vtsummarydir;’ -w testi/test

ggrider@pn2201328 bin % ./gufi\_query -n1 -S' drop vew if exists tree.vtsummaryuser;’ -w testi/test

ggrider@pn2201328 bin % ./gufi\_query -n1 -S' drop vew if exists tree.vtsummarygroup;’ -w testi/test

ggrider@pn2201328 bin % ./gufi\_query -n1 -S' drop table if exists tree.treesummary;’ -w testi/test

Notice the -w flag since you are writing to the database.

Since tree summaries are the only place where the number of subdirs exists (that does not appear in directory summaries due to rename and other events), if you combine that with the number of files and number of links in tree summaries, examining the tree summaries is a great place to see where horribly malformed trees live (way too many files for the dirs. or way too many dirs. for the files). Looking at tree summary information in the large likely how you learn how far to place tree summaries down in the tree over time and a good way to show users of such poorly formed trees that they have a mess that we could help them with, etc.

Tree summaries are also a good surrogate for how much work is in a portion of a tree and what kind of work which could be used for optimization later on.

## Rollups

The concept of a rollup is used where a tree you are indexing has a lot of directories, where the number of directories is like > 1% of the number of files (or average of 100 files/directory or potentially even a lower percent that 1%. The speed of queries is directly proportional to the number of directories. Every directory has an sqlite database. The time spent in query is highly proportional to the open time for a database. The databases are quite small so the I/O time is typically not significant, so you want to use a file system that can do a lot of open’s per second using multiple threads and you likely want to use a relatively high IOP underlying file storage. Typically, the file system open performance will be lower than the IOP’s of the devices if you are using modern flash. You can of course use multiple threads but still the open time dominates typically. You can of course spread the gufi index over multiple machines for VM’s to improve performance but that complicates queries with interim results, but can be done of course.

Rollups are an innovative solution to this performance improvement need. The way rollups work is if you have an intact gufi index tree, you can run the rollup process. It starts at the bottom of the tree and as it traverses up the tree, it will for a particular directory, copy all the information from all the child directory databases into the parent, but if and only if the access to all those child directories is 100% compatible with the parent. In other words it will roll the children up into the parent if every user that could access the parent could also access all the children (user, group and other are considered). This process continues to move up the tree until and for every “rolled up” directory, it has all the information from all the tree below if the permissions permit it. Typically you want to stop the rolled up db’s at something like 250,000-1M entries, because once you reach that level, you are now bandwidth dominated and no longer open latency dominated. Different trees will roll up differently based on permissions, but we have seen greater than 1000 way reduction in the number of databases one needs to consult to walk the entire tree. Speedups are typically proportional to the number of directories you saved and therefor the speed is somewhat astounding and you have now turned your query into a bandwidth problem and not an IOP’s problem and so you go as fast as the bandwidth you have, which can be impressive using modern flash devices.

Rollup’s should be considered to be temporary though because if anything changes in the src tree and you reindex part of the tree, you should drop all the rollups, replace the part of the tree that changed and then rerun the rollup. The way the rollup is implemented both building a rollup and deleting it quite easy.

When rollups are in place it modifies queries you need to be sure to be using rpath(sname,sroll) and vrsummary and vrpentries.

To rollup a tree

ggrider@pn2201328 bin % ./rollup -n 2 -L 100000 testr/test

Roots:

testr/test

Thread Pool Size: 2

Files/Links Limit: 100000

Not Processed: 0

Cannot Roll Up: 0

Can Roll Up: 13

Subdirectories:

min: 0

max: 3

median: 0.00

sum: 12

average: 0.92

Files/Links:

min: 0

max: 2

median: 1.00

sum: 16

average: 1.23

Level:

min: 0

max: 2

median: 2.00

sum: 21

average: 1.62

Successful: 13

Failed: 0

Files/Links: 16

Directories: 13 (1 empty)

Total: 29

Remaining Dirs: 1 (7.69%)

Took 0.02 seconds

ggrider@pn2201328 bin % ./gufi\_query -n1 -S 'select path(),rpath(sname,sroll), name, size from vrpentries;' -d " " -x testr/test

testr/test test/l1.2 f2.1.doc 2

testr/test test/l1.2/l2.2 f1.xls 0

testr/test test/l1.2/l2.3 f1.xls 0

testr/test test/l1.2/l2.1 f1.xls 0

testr/test test/l1.3 f3.1.doc 2

testr/test test/l1.3/l2.2 f1.bigger 7

testr/test test/l1.3/l2.3 f1.big 4

testr/test test/l1.3/l2.1 f1.biggest 10

testr/test test/l1.1 f1.1.doc 2

testr/test test/l1.1 f2.1.doc 2

testr/test test/l1.1/l2.2 f1.exe 0

testr/test test/l1.1/l2.2 f1.tar 0

testr/test test/l1.1/l2.3 f1.exe 0

testr/test test/l1.1/l2.3 f1.tar 0

testr/test test/l1.1/l2.1 f1.exe 0

testr/test test/l1.1/l2.1 f1.tar 0

This is rolled up, notice path() stays the same, we only consulted 1 database at the top of the tree, all info rolled up. I know we said not to use path() but this is just for illustration that shows that only one db.db gufi db file was opened and used. This means that the entire tree rolled up, which is no surprise given this all was done by the same user as the user with same umask everywhere.

ggrider@pn2201328 bin % ./gufi\_query -n1 -S 'select path(),rpath(sname,sroll),name,size from vrpentries;' -d " " -x testi/test

testi/test/l1.2 test/l1.2 f2.1.doc 2

testi/test/l1.3 test/l1.3 f3.1.doc 2

testi/test/l1.1 test/l1.1 f1.1.doc 2

testi/test/l1.1 test/l1.1 f2.1.doc 2

testi/test/l1.2/l2.2 test/l1.2/l2.2 f1.xls 0

testi/test/l1.2/l2.3 test/l1.2/l2.3 f1.xls 0

testi/test/l1.2/l2.1 test/l1.2/l2.1 f1.xls 0

testi/test/l1.3/l2.2 test/l1.3/l2.2 f1.bigger 7

testi/test/l1.3/l2.3 test/l1.3/l2.3 f1.big 4

testi/test/l1.3/l2.1 test/l1.3/l2.1 f1.biggest 10

testi/test/l1.1/l2.2 test/l1.1/l2.2 f1.exe 0

testi/test/l1.1/l2.2 test/l1.1/l2.2 f1.tar 0

testi/test/l1.1/l2.3 test/l1.1/l2.3 f1.exe 0

testi/test/l1.1/l2.3 test/l1.1/l2.3 f1.tar 0

testi/test/l1.1/l2.1 test/l1.1/l2.1 f1.exe 0

testi/test/l1.1/l2.1 test/l1.1/l2.1 f1.tar 0

Notice this is not rolled up, path() changes each time a new db is consulted. I know we said not to use path() but this is just for illustration that shows that only one db.db gufi db file was opened and used.

In a large tree, the rollups will be where ever the -L (max number of entries to roll up) is hit or the permissions change such that rollups are not legal in posix rules. Rollups again are temporary and the rollups should be unrolluped before doing maintenance on a tree/subtree (you know you don’t have to walk entire trees and update everything if you know nothing has changed in parts of the tree).

To unrollup a tree/subtree you use

./unrollup -n number-of-threads testr/test

We know that in the example the rollup was in testr/test

sqlite> select count(name) from summary;

sqlite> select count from summary;

Error: in prepare, no such column: count (1)

sqlite> select count(name) from summary;

1

As you can see, there would have been an entry for all rolled up directories in the summary table if it were rolled up, but there is only one record for that testr/test directory.

Lets test that by rolling it up again

ggrider@pn2201328 bin % ./rollup -n1 -L10000 testr/test

Roots:

testr/test

Thread Pool Size: 1

Files/Links Limit: 10000

Not Processed: 0

Cannot Roll Up: 0

Can Roll Up: 13

Subdirectories:

min: 0

max: 3

median: 0.00

sum: 12

average: 0.92

Files/Links:

min: 0

max: 2

median: 1.00

sum: 16

average: 1.23

Level:

min: 0

max: 2

median: 2.00

sum: 21

average: 1.62

Successful: 13

Failed: 0

Files/Links: 16

Directories: 13 (1 empty)

Total: 29

Remaining Dirs: 1 (7.69%)

Took 0.03 seconds

ggrider@pn2201328 bin % sqlite3 testr/test/db.db

SQLite version 3.37.0 2021-12-09 01:34:53

Enter ".help" for usage hints.

sqlite> select count(name) from summary;

13

sqlite> .quit

## Attaching gufi\_query to external databases

To be done