REGIONAL SDC 25

BY Developers FOR Developers

Regional SDC Denver
April 30, 2025

GUFI (Grand Unified File Indexer) What does it have to offer you?

LA-UR-25-22418

Gary Grider
Los Alamos National Laboratory

SNIA

Background: POSIX protects Exabytes of data in LANL/everywhere

File permissions (access file contents – read, modify file contents – write, execute a file – execute – that was easy (but there is more…)

- Directory permissions (dirs are different – the interpretation of rwx is different)
  - List file names and inode numbers only in a directory – dir read, file permissions irrelevant
  - CD into the directory – dir execute, file permissions irrelevant
  - Get information from the inode (size,dates,etc.) – dir execute, file permissions irrelevant
  - To search (traverse) in a tree you need dir execute and dir read (oh my but there is more
  - Everything you do traverses, like cat /u/me/file – to access the file contents you need file read permissions – but before you ever get there you need to traverse to that file (even if you know the path the file system has to traverse to it to get the inode and then determine if you have file read permissions. So you need file read and you need execute on all the directories above, so you would need dir execute permission in /u, /u/me, /u/me/files.
  - Well, isn't that special!

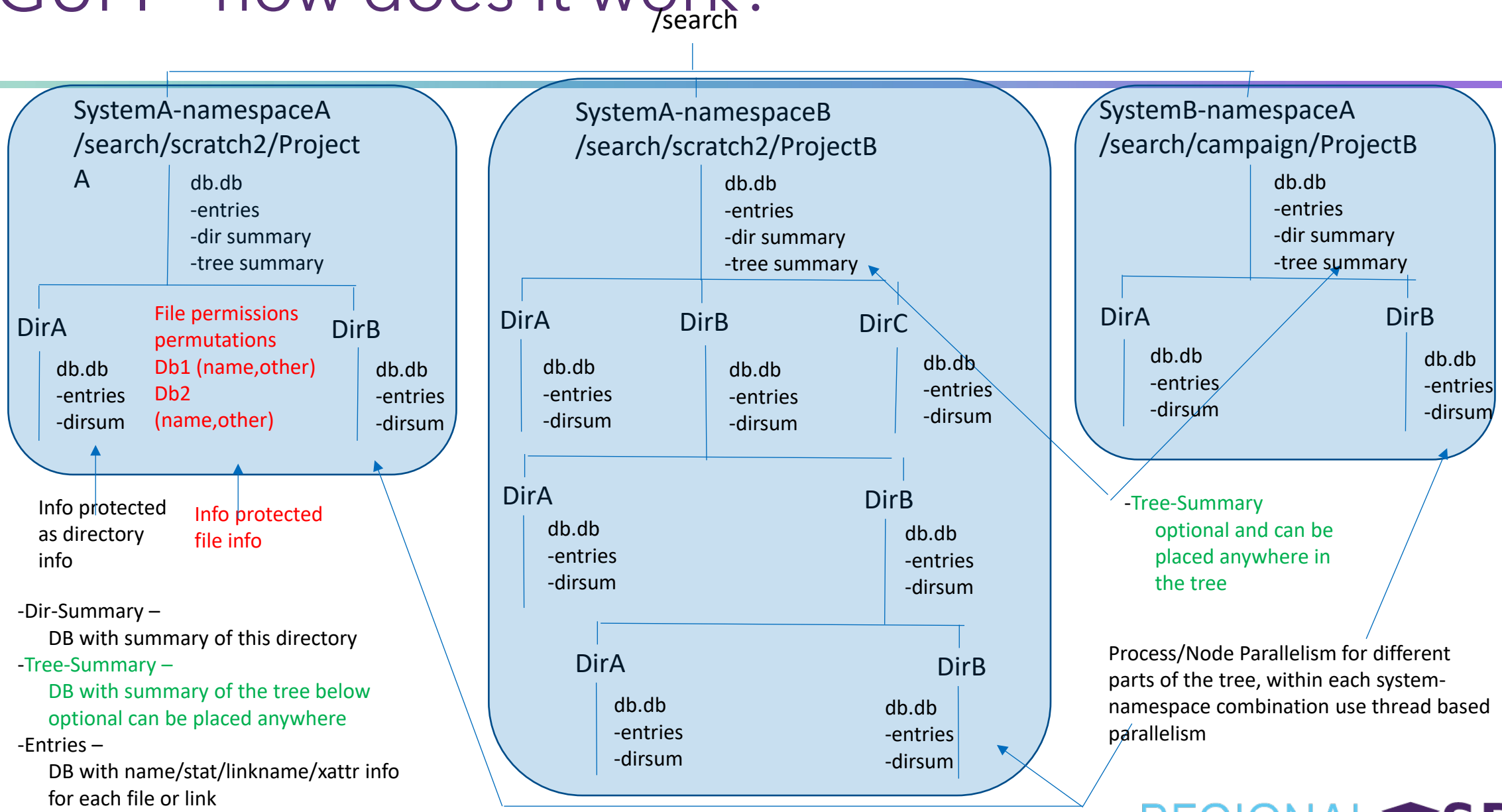
GUFI Goals

- Unified index over home, project, scratch, campaign, and archive
- Metadata only, metadata protected as the directory and as the files
- Shared index for users and admins
- Parallel search capabilities that are very fast (minutes for billions of files/dirs)
- Full/Incremental update from sources with reasonable update time/annoyance
- So fast people think its broken
- Leverage existing tech as much as possible both hdwr and software: flash, threads, clusters, sql as part of the interface, commercial db tech, …
- Simple so that an admin can easily understand/enhance
- It must make massive complex tree walk/joins/concatenations look like a table to the user!

Initial Design Thoughts

- **Why not a flat namespace?**
  - Performance is great, but…
  - Rename high in the tree is terribly costly
  - Security becomes a nightmare if users/admins can access the namespace
- **Leverage things that already work well, reduce required records to scan:**
  - POSIX permissions / tree walk (readdir+)
  - Breadth first search for parallelization
  - Our trees have inherent namespace divisions for parallelism
  - Embedded DBs are fast if not many joins and individual DB size < TB
  - Flash storage is cheap enough to hold everything with order ~10K IOPs each
  - Entries in file system reduce to essentially <dir count> * 3
  - Dense directories reduce footprint dramatically
  - SQL is easily utilized for general queries of attributes

GUFI – how does it work?



REGIONAL SDC

Draft Schemas

- Parent-Inode mapping file “directories-parent-inode directories Inode”
  - Parent inode is only kept for directories, not for files as that kills rename/move function
- "CREATE TABLE entries(
  - name TEXT PRIMARY KEY, name of file (Not path due to renames)
  - type TEXT, inode INT, f for file l for link inode
  - mode INT, posix mode bits
  - nlink INT, number of links
  - uid INT, gid INT, uid and gid
  - size INT, blksize INT, size and blocksize
  - blocks INT, blocks
  - atime INT, access time
  - mtime INT, file contents modification time
  - ctime INT, metadata change time
  - linkname TEXT, if link this is path to link
  - xattrs TEXT);"; single text string with key/value pairs with delimitors

REGIONAL SDC

Draft Schemas (continued)

- "CREATE TABLE summary(summary info for this directory
  - name TEXT PRIMARY KEY, name not path due to rename
  - type TEXT, inode INT, d for directory inode
  - mode INT, posix mode bits
  - nlink INT, number of links
  - uid INT, gid INT, uid gid
  - size INT, blksize INT, blocks INT, size, blocksize, blocks
  - atime INT, mtime INT, ctime INT, access time, dir contents mod time, md chg time
  - linkname TEXT, xattrs TEXT, if link, path to link, xattrs key/value delimited string
  - totfiles INT, totlinks INT, tot files in dir, tot links in dir
  - minuid INT, maxuid INT, mingid INT, maxgid INT, min and max uid and gid
  - minsize INT, maxsize INT, minimum file size and max file size
  - totltk INT, totmtk INT, totltm INT, total number of files lt KB mt KB, lt MB,
  - totmtm INT, totmtg INT, totmtt INT, total number of files mt MB mt GB, mt TB
  - totsize INT, total bytes in files in dir
  - minctime INT, maxctime INT, min max ctime
  - minmtime INT, maxmtime INT, min max mtime
  - minatime INT, maxatime INT, min max mtime
  - minblocks INT, maxblocks INT, min max blocks
  - totxattr INT, number of files with xattrs
  - depth INT);"; depth this directory is in the tree

REGIONAL SDC

Draft Schemas (continued)

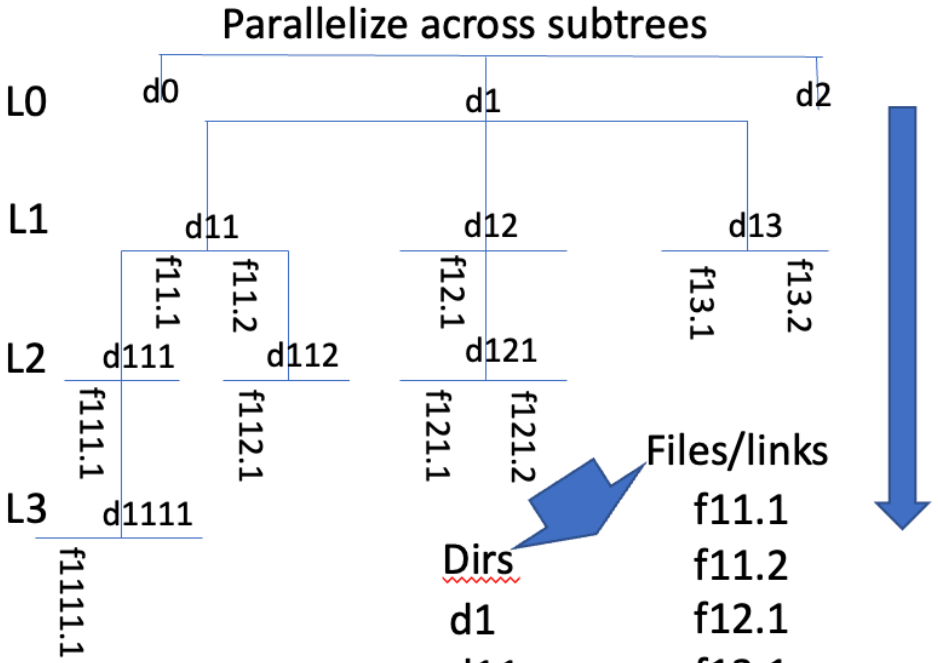
- "CREATE TABLE treesummary(summary info for this directory
  - totsubdirs INT, tot subdirs in tree
  - maxsubdirfiles INT, maxsubdirlinks INT, maxfiles in a subdir max links in a subdir
  - maxsubdirsize INT, most bytes in any subdir
  - totfiles INT, totlinks INT, tot files in tree, tot links in tree
  - minuid INT, maxuid INT, mingid INT, maxgid INT, min and max uid and gid
  - minsize INT, maxsize INT, minimum file size and max file size
  - totltk INT, totmtk INT, totltm INT, total number of files lt KB mt KB, lt MB,
  - totmtm INT, totmtg INT, totmtt INT, total number of files mt MB mt GB, mt TB
  - totsize INT, total bytes in files in tree
  - minctime INT, maxctime INT, min max ctime
  - minmtime INT, maxmtime INT, min max mtime
  - minatime INT, maxatime INT, min max mtime
  - minblocks INT, maxblocks INT, min max blocks
  - totxattr INT, number of files with xattrs
  - depth INT);"; depth this tree summary is in the tree

REGIONAL SDC

But will it be fast you you have millions of directories which means millions of databases

- Well it was pretty descent performance but not fast enough using lots of threads and breadth first search
- So we need to shard – how do we shard when we need to keep the
- Permission Permutations Sharding – Rollups
- Make an GUFI index, then go to the bottom of the tree and walk up and combine information until you reach a place where rollup rules no longer work, or you have enough info aggregated for efficiency
- Rollups are fast to perform and turn gufi index search into a bandwidth play, so fast its almost unbelievable

Step one and rules for rollup



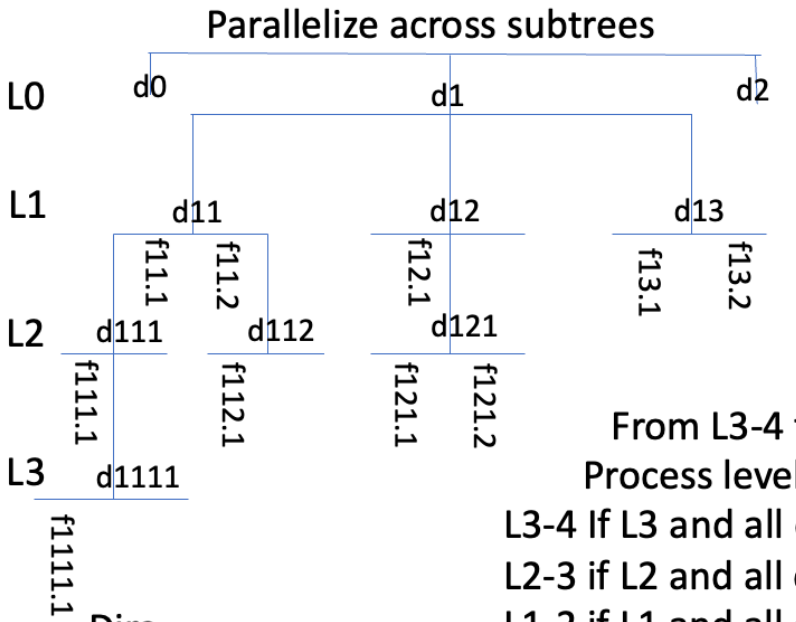
Walk down and record file/link info and directory info (name/path, uid, gid, mode, level, etc.)

Non exhaustive relatively simple rules to see if rollup is legal:

- If any of my children are not rolled up then NO
- When me and all my descendents are o+rx YES
- When me and all my descendents are ugo same and same uid, gid then YES
- When me and all my descendents are ug are same and same uid and gid and top o-rx then YES
- When me and all my descendents u same and top go-rx and same uid then YES
- Else NO

REGIONAL SDC

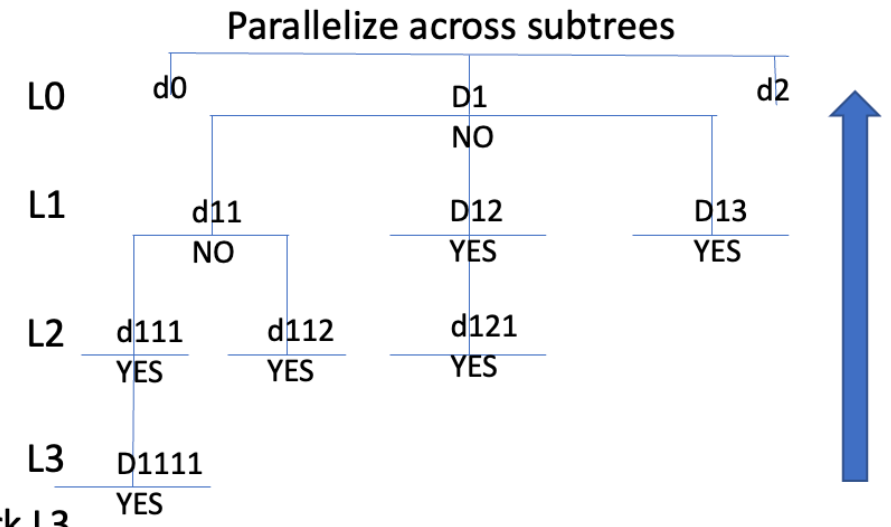
Rollup process example



Dirs
D1,uid,gid,mode,level
D11,….
D12,….
D13,…
D111,….
D112,….
D121,….
D1111,….

Process
From L3-4 to L0-1 (decr by 1)
Process level by level up the tree
L3-4 If L3 and all children (none) can roll mark L3
L2-3 if L2 and all children (L3) can roll mark L2
L1-2 if L1 and all children (L2) can roll mark L1
L0-1 if L0 and all children (L1) can roll mark L0
Insert top directory at (L-1) NO roll

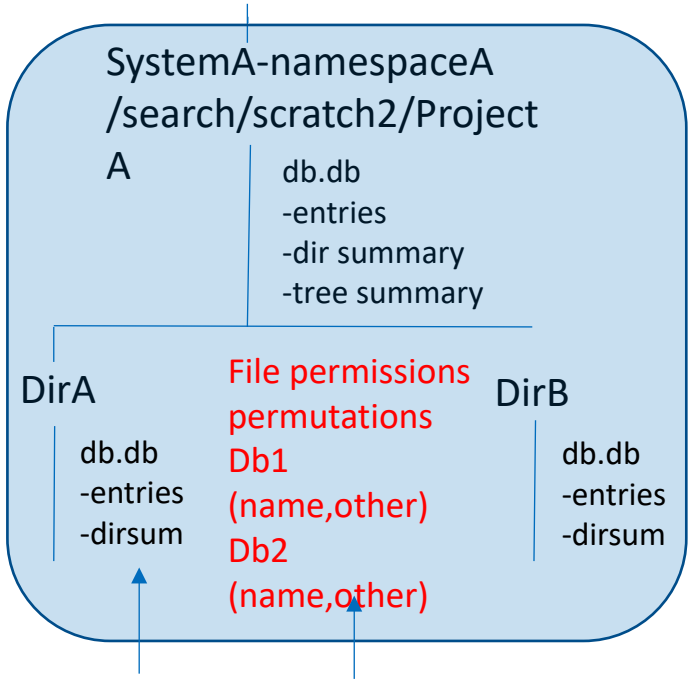
Rollup where child is YES and parent is NO
D111, D112, D12, D13
NOT D1111, D121 (as parent is YES)
Not D1, D11 (as child is NO)
Totals: total of 8 but 6 rolled into 4 so Dirs to visit went from 8 to 6



Dirs - min/max urx,grx,orx, min/max uid,gid
Top, … … NO – this is always NO
D1,…NO
D11,….NO
D12,….YES
D13,…YES
D111,….YES
D112,….YES
D121,…. YES
D1111,…. YES

REGIONAL SDC

Protecting directory level metadata and File level metadta



Db.db file is protected the same permissions as the directory, one file per directory (rolled up)

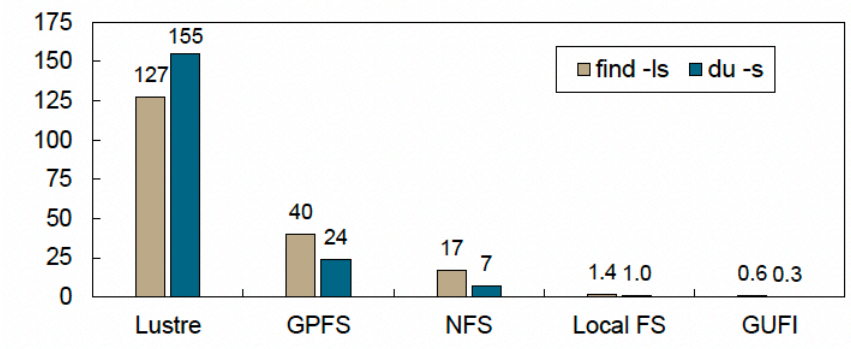
File permissions is how metadata about files (like xattr values, other metadata pulled from files) should be protected

- One DB file per permission permutation
- Roll-in processing concatenates all the db tables that the user can access

REGIONAL SDC

GUFI is State of the Art FAST

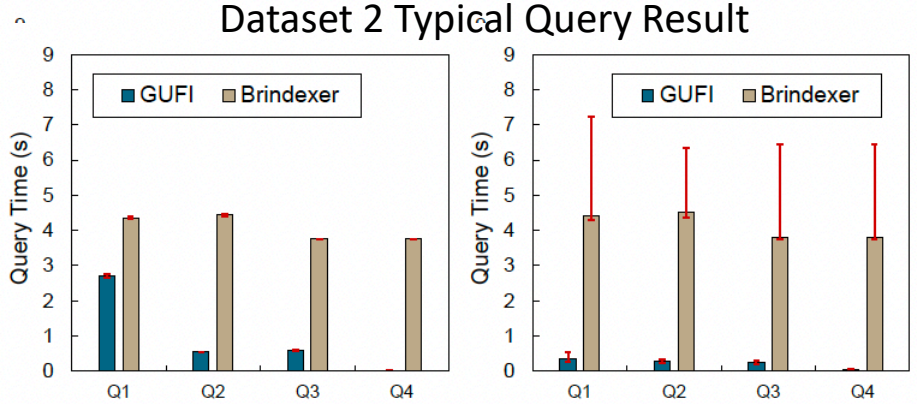
- *1) List all file names accessible by the user (top level for root).*
- *2) Print the size and name of every directory accessible by the user (top level for root)..*
- *3) Print the space used by the user's home directory (top level for root).*
- *4) Print the space used by the user's home directory (top level for root).*



List contents/attributes of Linux distro
GUFI compared to Linux utilities

Server	HPE DL380
Processor	Dual Intel Xeon 8280 2.7GHz (56 total cores, 112 threads)
Main Memory	192GiB
Storage	4x Samsung 1725A 1.6TB
Operating System	CentOS 7.7.1908
Kernel Version	Linux 5.7.9
File System	XFS
Dataset 1	1.6M directories, 13.2M files
Dataset 2	2.2M directories, 64.7M files

Dataset 2 Typical Query Result



(a) Admin Query Times (b) User Query Times

Fig. 10: Comparison of GUFI and Brindexer Query Performance. GUFI provides average speedups of 1.5x - 230x for administrator compared to the Brindexer index using a real file system namespace. User-specific queries result in even greater speedups; however, only GUFI can be accessed safely by users.

GUFI compared to state of the art (which is a variant of GUFI tech which dwarfs the other industry solutions).

REGIONAL SDC

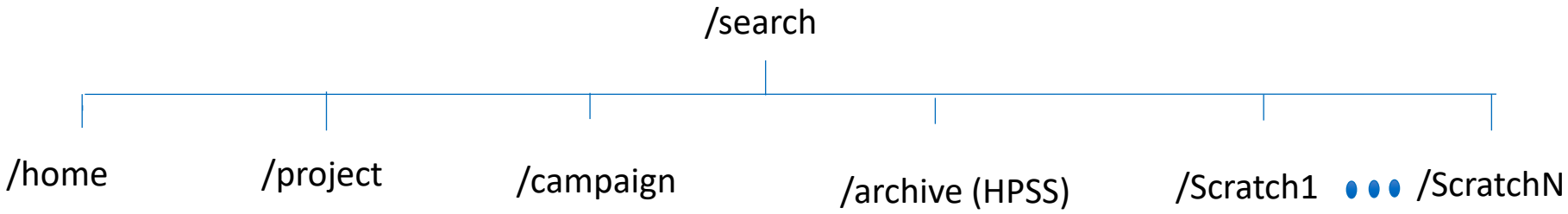
But how am I going to think about querying my hundreds of thousands of databases (including external databases)

- **Gufi makes all those databases appear as one database – really one table**
- **Gufi can be connected to workflows/data staging/movement tools (conduit/pftool/etc.)**
- **Schema can be changed/added to, just like sql database**
- **You can use POSIX XATTRS on files and GUFI will index that securely**
- **But there is an extremely powerful feature called external databases**
  - You can have a single external database for information that is for sure common need to know and that can be joined with
  - A more powerful external database feature: databases in the source file system directory that information is about (about the directory or file(s) in that directory)
  - Gufi will index the existence of these external DB's and then you can join user info with GUFI file system info in a completely secure way and if you change permissions in the source file system, GUFI reindexing will enable permissions to change and GUFI will behave like the file system permissions

How do users, admins, data scientists use it?

- **Simple for users (make these commands available on front ends and fta's) (make the syntax for these commands as close to find/start/ls as possible and the output as well). MAKE IT SO FAST USERS THINK ITS BROKEN ☺**
  - Gufi_find
  - Gufi_stat
  - Gufi_ls
- **Sysadmins and data scientists**
  - Gufi_query (extremely powerful)
  - Storage admins see all, data scientist see what they can see as a user
  - Highly threaded, breadthfirst search, can create output, outputdb's, aggregations of the threads results, connect to external databases of all kinds (user supplied, full text search, ai/ml embedded vector sim search, etc.

What does the gufi index tree look like?



- **You can search it all by starting your search at /search**
- **If you want to start your search lower you can either by min depth/maxdepth parameters or list subtrees you want to search**

There are several tables and many views

- **entries, summary, and tree summary are the main tables in directory permission database**
  - Important views
    - Vrpentries – files/links
    - Vrsummary – directory summary
    - Vrxpentries – files/links with xattrs
    - Vrxsummary – directory summary with xattrs
    - Virt tables (gufi_vt, gufi_vt_vrpentries, gufi_vt_vrsummary)
      - Makes gufi-query output look like a table to use (sqlalchemy/duckdb/sqlite itself)
  - For file permission database(s)
    - Recommend having a join field (name to join with file or dir name)
    - Can have as many other fields or relations as you want
    - Xattr tables are examples of this

Other considerations

- Can use N threads – each thread produces output, you can aggregate the output from all threads or keep it separate
- Outputs can be textual or sqlite db's
- Pick your delimiter
- Control where to start and how deep to go in the tree
- Turn on xattrs

```
ggrider@pn2201328 src % ./gufi_query
usage: ./gufi_query [options] GUFI_index ...
options:
  -h                     help
  -H                     show assigned input values (debugging)
  -v                     version
  -T <SQL_tsum>          SQL for tree-summary table
  -S <SQL_sum>           SQL for summary table
  -E <SQL_ent>           SQL for entries table
  -a                     AND/OR (SQL query combination)
  -n <threads>           number of threads
  -j                     print the information in terse form
  -o <out_fname>         output file (one-per-thread, with thread-id suffix)
  -d <delim>             delimiter (one char)  [use 'x' for 0x1E]
  -O <out_DB>            output DB
  -u                     prefix row with 1 int column count and each column with
 1 octet type and 1 size_t length
  -I <SQL_init>          SQL init
  -F <SQL_fin>           SQL cleanup
  -y <min level>         minimum level to go down
  -z <max level>         maximum level to go down
  -J <SQL_interm>        SQL for intermediate results
  -K <create aggregate>  SQL to create the final aggregation table
  -G <SQL_aggregate>     SQL for aggregated results
  -m                     Keep mtime and atime same on the database files
  -B <buffer size>       size of each thread's output buffer in bytes
  -w                     open the database files in read-write mode instead of r
ead only mode
  -x                     index/query xattrs
  -k <filename>          file containing directory names to skip
  -M <bytes>             target memory footprint
  -s <path>              File name prefix for swap files
  -e                     compress work items
  -Q <basename>
     <table>
     <template>.<table>
     <view>              External database file basename, per-attach table name,
 template + table name, and the resultant view

GUFI_index        find GUFI index here
```


Lots of helper functions

Function	Purpose
path()	Current directory relative to path passed into executable
epath()	Current directory basename
fpath()	Full path of current directory
rpath(sname, sroll)	Current directory relative to path passed into executable taking into account the rolled up name in summary table and rollup score. Should only be used with the sname and srollcolumns of the vrpentries, vrsummary, vrxpentries, and vrxsummaryviews. Usage: SELECT rpath(sname, sroll) FROM vrsummary/vrxsummary; SELECT rpath(sname, sroll) || "/" || name FROM vrpentries/vrxpentries;
level()	Depth of the current directory from the starting directory
starting point()	Path of the starting directory
subdirs(srollsubdirs, sroll)	Number of subdirectories under a directory. Only available for use with vrsummary. When the index is not rolled up, the value is retrieved from C. When the index is rolled up, the value is retrieved from vrsummary.

uidtouser(uid)
gidtogroup(gid)
modetotxt(mode)
strftime(format, timestamp)
blocksize(bytes, unit) k,m,g,t

`-T` should be used to query the `treesummary` table.

`-S` should be used to query the `summary` table and its variants.

`-E` should be used to query the `entries` table and its variants.

-a and/or (combo for above –T –S –E

-n number of threads

Function	Purpose
stdevs(numeric column)	Sample standard deviation
stdevp(numeric column)	Population standard deviation
median(numeric column)	Median of values

Example data/index.

- **Testsrc is the source tree**
- **Testidx is gufi index tree**
- **Db.db is the directory permissions database file (same permissions as directory)**
- **Fileinfodb.db is an example of an external database with permissions set same as the files it represents, with filename as the join field.**

```
ggrider@pn2201328 src % find tes
testsrc
testsrc/l1.2
testsrc/l1.2/f2.1.docx
testsrc/l1.2/l2.2
testsrc/l1.2/l2.2/f1.xls
testsrc/l1.2/l2.3
testsrc/l1.2/l2.3/f1.xls
testsrc/l1.2/f2.1.doc
testsrc/l1.2/l2.1
testsrc/l1.2/l2.1/f1.xls
testsrc/.DS_Store
testsrc/l1.3
testsrc/l1.3/l2.2
testsrc/l1.3/l2.2/f1.bigger
testsrc/l1.3/l2.3
testsrc/l1.3/l2.3/f1.big
testsrc/l1.3/f3.1.doc
testsrc/l1.3/l2.1
testsrc/l1.3/l2.1/f1.biggest
testsrc/l1.3/l2.1/garyext21.db
testsrc/l1.3/garyext21.db
testsrc/l1.1
testsrc/l1.1/l2.2
testsrc/l1.1/l2.2/f1.exe
testsrc/l1.1/l2.2/f1.tar
testsrc/l1.1/l2.3
testsrc/l1.1/l2.3/f1.exe
testsrc/l1.1/l2.3/f1.tar
testsrc/l1.1/f2.1.doc
testsrc/l1.1/l2.1
testsrc/l1.1/l2.1/f1.exe
testsrc/l1.1/l2.1/f1.tar
testsrc/l1.1/f1.1.doc
testsrc/garyext21.db
```

```
ggrider@pn2201328 src % find testidx
testidx
testidx/l1.2
testidx/l1.2/l2.2
testidx/l1.2/l2.2/db.db
testidx/l1.2/l2.3
testidx/l1.2/l2.3/db.db
testidx/l1.2/db.db
testidx/l1.2/l2.1
testidx/l1.2/l2.1/db.db
testidx/l1.3
testidx/l1.3/l2.2
testidx/l1.3/l2.2/db.db
testidx/l1.3/l2.3
testidx/l1.3/l2.3/db.db
testidx/l1.3/db.db
testidx/l1.3/l2.1
testidx/l1.3/l2.1/fileinfodb.db
testidx/l1.3/l2.1/db.db
testidx/l1.1
testidx/l1.1/l2.2
testidx/l1.1/l2.2/db.db
testidx/l1.1/l2.3
testidx/l1.1/l2.3/db.db
testidx/l1.1/db.db
testidx/l1.1/l2.1
testidx/l1.1/l2.1/db.db
testidx/db.db
```


examples

- **Explain views**
- **Vrpentries**
- **Vrpentries has dir and file info**
- **Aggregation**
- **Controlling starting point, min depth, and max depth**
- **Limiting**
- **SQL statement stacking**
- **Vrsummary**
- **Combining vrsummary and vrpentries**

- **Tree summary and combining tree summary with other**
- **Xattrs**
- **Group by**
- **Ext db full text search**
- **Ext db user supplied**
- **Ext db vector db**

Vrpentries

Even supports regex

```
ggrider@pn2201328 src % ./gufi_query -n1 -d
'|' -E "select path(),name,size from vrpent
ries where regexp('f1.big$',name);" testidx
testidx/l1.3/l2.3|f1.big|4
ggrider@pn2201328 src %
```

simple query

```
ggrider@pn2201328 src % ./gufi_query
-n1 -d'|' -E 'select name,uid,size,mo
de,mtime from vrpentries;' testidx
f2.1.doc|2078|2|33188|1680818365
f3.1.doc|2078|2|33188|1680818372
f1.1.doc|2078|2|33188|1680818343
f2.1.doc|2078|2|33188|1680818356
f1.xls|2078|0|33188|1680817472
f1.xls|2078|0|33188|1680817476
f1.xls|2078|0|33188|1680817469
f1.bigger|2078|7|33188|1680817672
f1.big|2078|4|33188|1680817656
f1.biggest|2078|10|33188|1680817700
f1.exe|2078|0|33188|1680817412
f1.tar|2078|0|33188|1680817430
f1.exe|2078|0|33188|1680817415
f1.tar|2078|0|33188|1680817424
f1.exe|2078|0|33188|1680817390
f1.tar|2078|0|33188|1680817439
ggrider@pn2201328 src %
```

Powerful functions

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||name,uidtouser(uid),s
ize,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from vrpentries;" testidx
testidx/l1.2f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:25 2023
testidx/l1.3f3.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:32 2023
testidx/l1.1f1.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:03 2023
testidx/l1.1f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:16 2023
testidx/l1.2/l2.2f1.xls|ggrider|0|-rw-r--r--|Thu Apr  6 15:44:32 2023
testidx/l1.2/l2.3f1.xls|ggrider|0|-rw-r--r--|Thu Apr  6 15:44:36 2023
testidx/l1.2/l2.1f1.xls|ggrider|0|-rw-r--r--|Thu Apr  6 15:44:29 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.1/l2.2f1.exe|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:32 2023
testidx/l1.1/l2.2f1.tar|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:50 2023
testidx/l1.1/l2.3f1.exe|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:35 2023
testidx/l1.1/l2.3f1.tar|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:44 2023
testidx/l1.1/l2.1f1.exe|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:10 2023
testidx/l1.1/l2.1f1.tar|ggrider|0|-rw-r--r--|Thu Apr  6 15:43:59 2023
```

Powerful functions with a simple where

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||name,uidtouser(uid),s
ize,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from vrpentries where size
> 0;" testidx
testidx/l1.2f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:25 2023
testidx/l1.3f3.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:32 2023
testidx/l1.1f1.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:03 2023
testidx/l1.1f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:16 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
```


Vrpentries has both file and directory info in the record

Find the files in directories that have more than 1 file

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "select path()||
name, size from vrpentries where dtotfile>1;" testidx
testidx/l1.1f1.1.doc|2
testidx/l1.1f2.1.doc|2
testidx/l1.1/l2.2f1.exe|0
testidx/l1.1/l2.2f1.tar|0
testidx/l1.1/l2.3f1.exe|0
testidx/l1.1/l2.3f1.tar|0
testidx/l1.1/l2.1f1.exe|0
testidx/l1.1/l2.1f1.tar|0
ggrider@pn2201328 src %
```

Find the files in directories that are in a directory with %1.1 in the directory name

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "select path()||
name, size from vrpentries where dname like '%1.1';" testidx
testidx/l1.1f1.1.doc|2
testidx/l1.1f2.1.doc|2
```

Note the directory summary fields that are in the vrpentries records are prefaced with a d

Aggregating

order by desc, but wait it didn't work

-E runs that query in every directory, so this order by desc ordered for every directory but not over all the directories/threads

-I makes a out db for each thread
-K makes a single aggregate db
-E inserts each thread output into out db
-J inserts each threads out into aggregate
-G selects from aggregate

So you can sort/group by over all threads/directories with aggregates

This allows all threads to run as fast as they can until the end and then aggregates so you can do last sort/sum/group/etc. over all the data from all dirs. visited for all threads

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||name,uidtouser(uid),s
ize,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from vrpentries where size
> 0 order by size desc;" testidx
testidx/l1.2f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:25 2023
testidx/l1.3f3.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:32 2023
testidx/l1.1f1.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:03 2023
testidx/l1.1f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:16 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
```

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, on
ame text, ouser text, osize int64, omode text, omtime text);" -K "create temp table agg
table(apath text, aname text, auser text, asize int64, amode text, amtime text);" -E "i
nsert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a
 %b %e %H:%M:%S %Y', mtime) from vrpentries where size > 0;" -J "insert into aggtable s
elect * from outtable;" -G "select apath||aname,auser,asize,amode,amtime from aggtable
order by asize desc;" testidx
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
testidx/l1.2f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:25 2023
testidx/l1.3f3.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:32 2023
testidx/l1.1f1.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:03 2023
testidx/l1.1f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:16 2023
```


Start/min/max depth

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||name,size from vrpentries;" testidx
testidx/l1.2f2.1.doc|2
testidx/l1.3f3.1.doc|2
testidx/l1.1f1.1.doc|2
testidx/l1.1f2.1.doc|2
testidx/l1.2/l2.2f1.xls|0
testidx/l1.2/l2.3f1.xls|0
testidx/l1.2/l2.1f1.xls|0
testidx/l1.3/l2.2f1.bigger|7
testidx/l1.3/l2.3f1.big|4
testidx/l1.3/l2.1f1.biggest|10
testidx/l1.1/l2.2f1.exe|0
testidx/l1.1/l2.2f1.tar|0
testidx/l1.1/l2.3f1.exe|0
testidx/l1.1/l2.3f1.tar|0
testidx/l1.1/l2.1f1.exe|0
testidx/l1.1/l2.1f1.tar|0
```

Max depth

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -z1 -E "select path()||name,size from vrpentries;" testidx
testidx/l1.2f2.1.doc|2
testidx/l1.3f3.1.doc|2
testidx/l1.1f1.1.doc|2
testidx/l1.1f2.1.doc|2
```

Start lower

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||name,size from vrpentries;" testidx/l1.1
testidx/l1.1f1.1.doc|2
testidx/l1.1f2.1.doc|2
testidx/l1.1/l2.2f1.exe|0
testidx/l1.1/l2.2f1.tar|0
testidx/l1.1/l2.3f1.exe|0
testidx/l1.1/l2.3f1.tar|0
testidx/l1.1/l2.1f1.exe|0
testidx/l1.1/l2.1f1.tar|0
```

Min/Max Depth

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -z2 -y2 -E "select path()||name,size from vrpentries;" testidx
testidx/l1.2/l2.2f1.xls|0
testidx/l1.2/l2.3f1.xls|0
testidx/l1.2/l2.1f1.xls|0
testidx/l1.3/l2.2f1.bigger|7
testidx/l1.3/l2.3f1.big|4
testidx/l1.3/l2.1f1.biggest|10
testidx/l1.1/l2.2f1.exe|0
```


Limiting

This orders for each thread and limits to 1 record per directory that thread encounters, the in memory out table will grow during the run

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, oname text, ouser text, osize int64, omode
 text, omtime text);" -K "create temp table aggtable(apath text, aname text, auser text, asize int64, amode text, amtime text);
" -E "insert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from
vrpentries where size > 0 order by size limit 1" -J "insert into aggtable select * from outtable;" -G "select apath||aname,ause
r,asize,amode,amtime from aggtable order by asize desc;" testidx
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
testidx/l1.2f2.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:25 2023
testidx/l1.3f3.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:32 2023
testidx/l1.1f1.1.doc|ggrider|2|-rw-r--r--|Thu Apr  6 15:59:03 2023
```

This orders for each thread and limits to 1 record per directory that thread encounters, and limits the aggregated total to 3

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, oname text, ouser text, osize int64, omode
 text, omtime text);" -K "create temp table aggtable(apath text, aname text, auser text, asize int64, amode text, amtime text);
" -E "insert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from
vrpentries where size > 0 order by size limit 1" -J "insert into aggtable select * from outtable;" -G "select apath||aname,ause
r,asize,amode,amtime from aggtable order by asize desc limit 3;" testidx
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
```


SQL Statement Stacking

This orders for each thread and limits to 3 record per directory that thread encounters, the in memory out table will grow during the run

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, oname text, ouser text, osize int64, omode
 text, omtime text);" -K "create temp table aggtable(apath text, aname text, auser text, asize int64, amode text, amtime text);
" -E "insert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from
vrpentries where size > 0 order by size limit 3" -J "insert into aggtable select * from outtable;" -G "select apath||aname,ause
r,asize,amode,amtime from aggtable order by asize desc limit 3;" testidx
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
```

This stacks another sql statement onto the –E for each directory encountered to trim the in memory out table per thread, very efficient way to find the top N without big memory use

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, oname text, ouser text, osize int64, omode
 text, omtime text);" -K "create temp table aggtable(apath text, aname text, auser text, asize int64, amode text, amtime text);
" -E "insert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a %b %e %H:%M:%S %Y', mtime) from
vrpentries where size > 0 order by size limit 3;delete from outtable where osize not in (select osize from outtable order by os
ize desc limit 3);" -J "insert into aggtable select * from outtable;" -G "select apath||aname,auser,asize,amode,amtime from agg
table order by asize desc limit 3;" testidx
testidx/l1.3/l2.1f1.biggest|ggrider|10|-rw-r--r--|Thu Apr  6 15:48:20 2023
testidx/l1.3/l2.2f1.bigger|ggrider|7|-rw-r--r--|Thu Apr  6 15:47:52 2023
testidx/l1.3/l2.3f1.big|ggrider|4|-rw-r--r--|Thu Apr  6 15:47:36 2023
```

You can stack as many sql statements on all –T –S –E and other places in gufi do do whatever you want

vrsummary

Very powerful set of aggregator fields in vrsummary record, tot files tot files with xattrs, tot files older than .. Tot files bigger than …

…

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "
select path()||name, size from vrsummary;" testidx
testidxtestsrc|160
testidx/l1.2l1.2|192
testidx/l1.3l1.3|192
testidx/l1.1l1.1|224
testidx/l1.2/l2.2l2.2|96
testidx/l1.2/l2.3l2.3|96
testidx/l1.2/l2.1l2.1|96
testidx/l1.3/l2.2l2.2|96
testidx/l1.3/l2.3l2.3|96
testidx/l1.3/l2.1l2.1|96
testidx/l1.1/l2.2l2.2|128
testidx/l1.1/l2.3l2.3|128
testidx/l1.1/l2.1l2.1|128
```

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "select path()||
name, size from vrsummary where totfiles>1;" testidx
testidx/l1.1l1.1|224
testidx/l1.1/l2.2l2.2|128
testidx/l1.1/l2.3l2.3|128
testidx/l1.1/l2.1l2.1|128
```

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "select path()||
name, size from vrsummary where totxattr>0;" testidx
testidx/l1.1l1.1|224
```


Combining Vrsummary and Vrpentries

If you query vrpentries and use directory info related where clauses, you are scanning all the files, if you want to first query the vrsummary table and only if you need to scan the vrpentries table, you can use -S and -E, there is an implied AND (if -S then do -E)

Of course you can do that in a single sql statement as well

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -S "select path()||name,
 totfiles from vrsummary where totfiles>1;" -E"select path()||name from
 vrpentries;"  testidx
testidx/l1.1l1.1|2
testidx/l1.1f1.1.doc
testidx/l1.1f2.1.doc
testidx/l1.1/l2.2l2.2|2
testidx/l1.1/l2.2f1.exe
testidx/l1.1/l2.2f1.tar
testidx/l1.1/l2.3l2.3|2
testidx/l1.1/l2.3f1.exe
testidx/l1.1/l2.3f1.tar
testidx/l1.1/l2.1l2.1|2
testidx/l1.1/l2.1f1.exe
testidx/l1.1/l2.1f1.tar
```

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select path()||
name from vrpentries where (select inode from vrsummary where totx
attr>0)>0;" testidx
testidx/l1.1f1.1.doc
testidx/l1.1f2.1.doc
```


Tree summary and combining tree summary with other

Powerful aggregate information in tree summary, summarizes everything below, at every level, which trees have this in them

DU command is now free (milliseconds)

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -z1 -T "select path(),totsubdirs,totfiles,totsize,maxsize from treesummary;" testidx
testidx|12|16|29|10
testidx/l1.2|3|4|2|2
testidx/l1.3|3|4|23|10
testidx/l1.1|3|8|4|2
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -z1 -T "select path(),totsubdirs,totfiles,totsize,maxsize from treesummary where totxattr>0;" testidx
testidx|12|16|29|10
testidx/l1.1|3|8|4|2
```

Very powerful way to limit what all directories in what subtrees you scan based on treesummary info

```
testidx/l1.1f2.1.doc
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -z1 -T "select '.' from treesummary where totxattr>0;" -E "select path()||'/'||name from vrpentries;" testidx
.
.
testidx/l1.1/f1.1.doc
testidx/l1.1/f2.1.doc
```


Xattrs

Yes you can list or search on xattrs
Notice using -x in the qufi_query cmd

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -x -E "select path(),nam
e,xattr_name xattr_value from xpentries;" testidx
testidx/l1.2|f2.1.doc|
testidx/l1.3|f3.1.doc|
testidx/l1.1|f1.1.doc|
testidx/l1.1|f2.1.doc|user.garyxattr
testidx/l1.2/l2.2|f1.xls|
testidx/l1.2/l2.3|f1.xls|
testidx/l1.2/l2.1|f1.xls|
testidx/l1.3/l2.2|f1.bigger|
testidx/l1.3/l2.3|f1.big|
testidx/l1.3/l2.1|f1.biggest|
testidx/l1.1/l2.2|f1.exe|
testidx/l1.1/l2.2|f1.tar|
testidx/l1.1/l2.3|f1.exe|
testidx/l1.1/l2.3|f1.tar|
testidx/l1.1/l2.1|f1.exe|
testidx/l1.1/l2.1|f1.tar|
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -x -E "select path(),nam
e,xattr_name xattr_value from xpentries where xattr_name is not null;"
testidx
testidx/l1.1|f2.1.doc|user.garyxattr
```

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -x -E "select path(),nam
e,xattr_name xattr_value from xpentries where xattr_name like '%gary%';
" testidx
testidx/l1.1|f2.1.doc|user.garyxattr
```

Why is this special?

Remember xattr values are protected like the file and not like the directory so they are kept in separate db files (potentially – with different permissions)

Group by

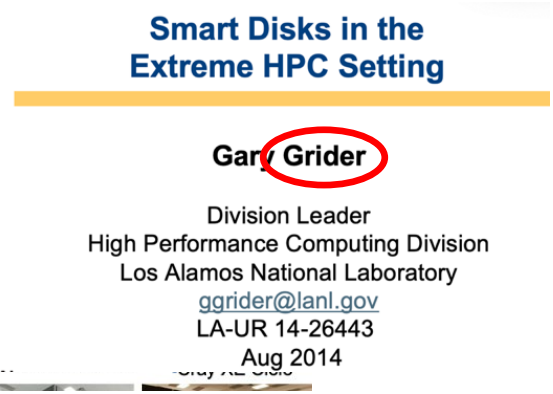
Nice feature, group by, notice we use an aggregate to get all the files and sizes and we group by size on the aggregated select. Of course you could do histograms on about anything using this feature, especially since the summary record has histograms for size and date as does the treesummary record as well

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -I "create table outtable(opath text, oname text, ouser text, osize i
nt64, omode text, omtime text);" -K "create temp table aggtable(apath text, aname text, auser text, asize int64, amo
de text, amtime text);" -E "insert into outtable select path(),name,uidtouser(uid),size,modetotxt(mode),strftime('%a
 %b %e %H:%M:%S %Y', mtime) from vrpentries;" -J "insert into aggtable select * from outtable;" -G "select asize,cou
nt() from aggtable group by asize;" testidx
0|9
2|4
4|1
7|1
10|1
```


Full text ext db example (history db)

From hpc history gufi db (has full text search words table joined on inode with entries (wordf MATCH 'grider and cdc'

Notice twords is total word count for the document




```
ggrider@pn2201328 ~ % ./sqlite/sqlite-src-3180000/gufi-111618/bfq -n1 -Pp -d'|' -E "select path()||'/'||name from entries
 where path()||name like '%Seagate%';" /Volumes/hpchist/idx
/Volumes/hpchist/idx/Z-GaryGriderHistory/recent-presentations/Seagate-Kinetic-LANL-info-v1.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/recent-presentations/Seagate-potential-CRADA-Info-v1.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/trinity/Trinity-Seagate-Storge.jpg
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-documents/106273doc/Seagate-Collab-08182014.docx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-documents/106273doc/HPC-DO/Seagate-PTS4-gg-kl-draft.docx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/LANL-Seagate-CRADA.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Xyratex-visit.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Kinetic-LANL-info-v1.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-potential-CRADA-Info.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Kinetic-LANL-info.pptx
/Volumes/hpchist/idx/Z-GaryGriderHistory/pictures/morepics/machinepics/Trinity-Seagate-Storge.jpg
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-documents/106273doc/HPC-DO/LANL-Sandia-Capability/LANL--Sandia-Seagate.doc
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/CCN-9/HEC-IWG-FS-IO-Workshop-FY05/Seagate-presenta
tion.ppt
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/asci/Sgpfs-gfs-dfs/sgpfs-workshop/04_10min_Talks/C
hris_Malakapalli_Seagate/Seagatesgpfs.ppt
ggrider@pn2201328 ~ % ./sqlite/sqlite-src-3180000/gufi-111618/bfq -n1 -Pp -d'|' -E "select path()||'/'||name,twords from
entries inner join words on inode=tinode where wordf MATCH 'grider AND cdc' and path()||name like '%Seagate%';" /Volumes
/hpchist/idx
/Volumes/hpchist/idx/Z-GaryGriderHistory/recent-presentations/Seagate-Kinetic-LANL-info-v1.pptx618
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Xyratex-visit.pptx1475
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Kinetic-LANL-info-v1.pptx618
/Volumes/hpchist/idx/Z-GaryGriderHistory/older-presentations/106273ppt/HPC-DO/Seagate-Kinetic-LANL-info.pptx432
```


The swiss army knives of data curation/science Running commands in queries.

- **Intop Run a command as a function passing in anything you want and getting an integer back**

```
ggrider@pn2201328 src % ./gufi_query -n1 -d"|" -E
 "select path()||'/'||name,intop('cat testsrc'||s
ubstr(path(),instr(path(),'/'))||'/'||name||' | w
c -c') from vrpentries where size>0;" testidx
testidx/l1.2/f2.1.doc|2
testidx/l1.3/f3.1.doc|2
testidx/l1.1/f1.1.doc|2
testidx/l1.1/f2.1.doc|2
testidx/l1.3/l2.2/f1.bigger|7
testidx/l1.3/l2.3/f1.big|4
testidx/l1.3/l2.1/f1.biggest|10
```

- **Strop Run a command as a function passing in anything you want and getting a string back**

- **Yes run any command on any thing driven by any query you want across your holdings**

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -E "select
 strop('wc -c testsrc'||substr(path(),instr(path(),'/'))|
|'/'||name) from vrpentries where size>0;" testidx
       2 testsrc/l1.2/f2.1.doc
       2 testsrc/l1.3/f3.1.doc
       2 testsrc/l1.1/f1.1.doc
       2 testsrc/l1.1/f2.1.doc
       7 testsrc/l1.3/l2.2/f1.bigger
       4 testsrc/l1.3/l2.3/f1.big
      10 testsrc/l1.3/l2.1/f1.biggest
```

A data scientist/curators dream tool for record oriented ops

The swiss army knives of data curation/science
Running commands to populate virtual tables

- **Create a custom virtual table in gufi_query (this one runs per directory (notice %s (%s src path %n name %i index path) (notice -p (path to src)**

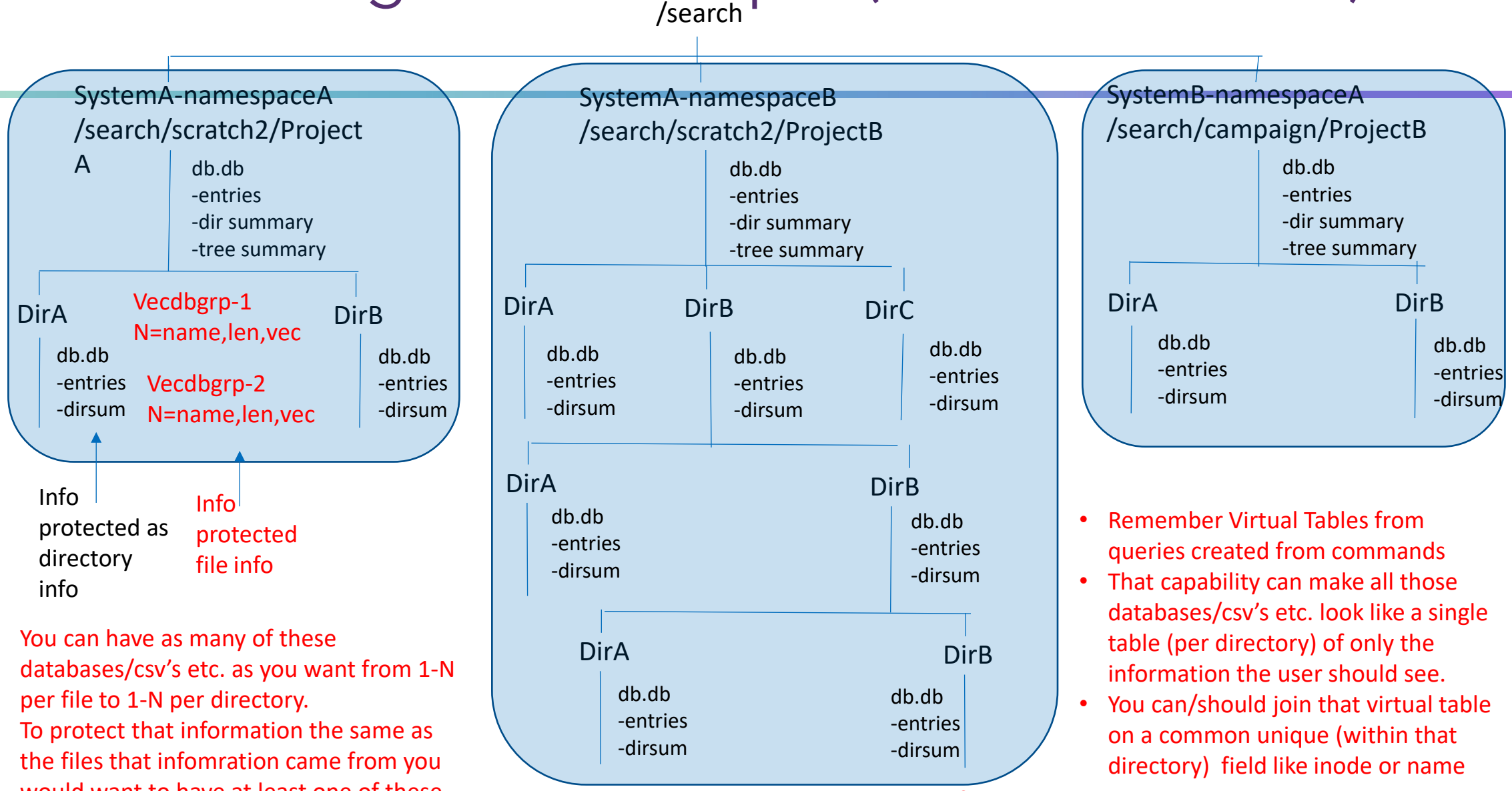
- **Create a custom virtual table without gufi_query (find all the csv's, turn them into a single table and sort). Notice casting to make it into an int64**

```
ggrider@pn2201328 src % ./gufi_query -n1 -d'|' -p
 "testsrc" -S "create virtual table temp.csv usin
g run_vt(cmd='find \"%s\" -maxdepth 1 -name \"f1.
big*\" -exec cat \"{}\" \\;',cols='mycat,mycat2',
d=','); select * from temp.csv; drop table temp.c
sv" testidx
22|222
1|1
333|33333
```

```
ggrider@pn2201328 src % echo -n "create virtual t
able temp.csv using run_vt(cmd='find "testsrc"  -
name \"f1.big*\" -exec cat \"{}\" \\;',cols='myca
t,mycat2',d=','); select mycat||','||mycat2 from
temp.csv where cast(mycat2 as int64)>2 order by m
ycat2 desc; drop table temp.csv" | ./gufi_sqlite3
333,33333
22,222
```

A data scientist/curators dream tool for table oriented ops

Nearest Neighbor Example (GUFI reminder)



- You can have as many of these databases/csv's etc. as you want from 1-N per file to 1-N per directory.
- To protect that information the same as the files that infomration came from you would want to have at least one of these files per "READ permission permutation uid/gid/mode bits
- You can put these databases in the source file system or the index file system (they can also be the same file system of course)
- Remember Virtual Tables from queries created from commands
- That capability can make all those databases/csv's etc. look like a single table (per directory) of only the information the user should see.
- You can/should join that virtual table on a common unique (within that directory) field like inode or name

REGIONAL SDC

Nearest Neighbor example

Sprinkle these vec db's in the tree

Notice in l1.2 directory there are two representing two different permission permutations

find testsrc -name 'vec*'

testsrc/l1.2/vecgroupa.db

testsrc/l1.2/vecgroupb.db

testsrc/l1.3/l2.2/vec.db

testsrc/l1.3/l2.3/vec.db

testsrc/l1.3/vec.db

testsrc/l1.3/l2.1/vec.db

testsrc/l1.1/vec.db

The index with db per dir

find testidx -name 'db.db'

testidx/l1.2/l2.2/db.db

testidx/l1.2/l2.3/db.db

testidx/l1.2/db.db

testidx/l1.2/l2.1/db.db

testidx/l1.3/l2.2/db.db

testidx/l1.3/l2.3/db.db

testidx/l1.3/db.db

testidx/l1.3/l2.1/db.db

testidx/l1.1/l2.2/db.db

testidx/l1.1/l2.3/db.db

testidx/l1.1/db.db

testidx/l1.1/l2.1/db.db

testidx/db.db

The vec.db vec table

CREATE TABLE vec (vname text,vlen int64, vvec text);

Nearest Neighbor example1

Naïve (collect all order/limit at the end)

Collect all records and order by and limit 3

./gufi_query -n1 -d'|' -a -p "testsrc"

** create an output table per thread

-I "create table vvecog(vogname text, volen int64,vodist int64);"

** create a single temp aggregate table for all threads

-K "create temp table vvecag(vagname text, vaglen int64,vagdist int64);"

** create temp virttable - find vec* -exec sqlite3 query (include vec* files once per dir)

-S "create virtual table temp.vvec using run_vt(cmd='find \"%s\" -maxdepth 1 -name \"vec*.db\" -exec sqlite3 -separator \",\" \"{}\" \"select vname,vlen,abs(4-vlen) as vdist from vec;\" \\;',cols='myvname,myvlen,myvdist',d=',');

** insert query of virtual table into output table per thread

** notice join of virtual table and vrpentries on name (this allows easy selection of which files you want to include (only files < 1000 bytes).

insert into vvecog select '%s'||'/'||myvname,myvlen,myvdist from temp.vvec inner join vrpentries on name=myvname where size < 10000;

** drop virtual table (once per directory because you run find per directory)

drop table temp.vvec;"

** after all threads complete serialize insert into aggregate table once per run

-J "insert into vvecag select * from vvecog;"

** finally select on the aggregate table once order by distance and limit 3

-G "select vagname,vaglen,vagdist from vvecag order by vagdist asc limit 3;" testidx

testsrc/l1.2/f2.1.doc|4|0

testsrc/l1.3/f3.1.doc|4|0

testsrc/l1.2/f1.1.doc|3|1

Nearest 3 vector length files of size < 1000 of all the files I can see

Nearest Neighbor example2

Tiny better (reduce top 3 when creating aggr)

Collect all records and order by and limit 3

./gufi_query -n1 -d'|' -a -p "testsrc"

-I "create table vvecog(vogname text, volen int64,vodist int64);"

-K "create temp table vvecag(vagname text, vaglen int64,vagdist int64);"

-S "create virtual table temp.vvec using run_vt(cmd='find \"%s\" -maxdepth 1 -name \"vec*.db\" -exec sqlite3 -separator \",\" \"{}\" \"select vname,vlen,abs(4-vlen) as vdist from vec;\" \\;',cols='myvname,myvlen,myvdist',d=',');

insert into vvecog select '%s'||'/'||myvname,myvlen,myvdist from temp.vvec inner join vrpentries on name=myvname where size < 10000;

** drop virtual table (once per directory because you run find per directory)

drop table temp.vvec;”

** after all threads complete serialize insert into aggregate table once per run order/limit

-J "insert into vvecag select * from vvecog order by vodist asc limit 3;"

** finally select on the aggregate table once order by distance and limit 3

-G "select vagname,vaglen,vagdist from vvecag order by vagdist asc limit 3;" testidx

testsrc/l1.2/f2.1.doc|4|0

testsrc/l1.3/f3.1.doc|4|0

testsrc/l1.2/f1.1.doc|3|1

Nearest Neighbor example3

Some better (reduce top 3 per thread per dir)

Collect all records and order by and limit 3

./gufi_query -n1 -d'|' -a -p "testsrc"

-I "create table vvecog(vogname text, volen int64,vodist int64);"

-K "create temp table vvecag(vagname text, vaglen int64,vagdist int64);"

-S "create virtual table temp.vvec using run_vt(cmd='find \"%s\" -maxdepth 1 -name \"vec*.db\" -exec sqlite3 -separator \",\" \"{}\" \"select vname,vlen,abs(4-vlen) as vdist from vec;\" \\;',cols='myvname,myvlen,myvdist',d=',');

insert into vvecog select '%s'||'/'||myvname,myvlen,myvdist from temp.vvec inner join vrpentries on name=myvname where size < 10000 order by myvdist asc limit 3;

** drop virtual table (once per directory because you run find per directory)

drop table temp.vvec;"

** after all threads complete serialize insert into aggregate table once per run order/limit

-J "insert into vvecag select * from vvecog order by vodist asc limit 3;"

** finally select on the aggregate table once order by distance and limit 3

-G "select vagname,vaglen,vagdist from vvecag order by vagdist asc limit 3;" testidx

testsrc/l1.2/f2.1.doc|4|0

testsrc/l1.3/f3.1.doc|4|0

testsrc/l1.2/f1.1.doc|3|1

Nearest Neighbor example3

much better (trim output per thread per dir)

Collect all records and order by and limit 3

./gufi_query -n1 -d'|' -a -p "testsrc"

-I "create table vvecog(vogname text, volen int64,vodist int64);"

-K "create temp table vvecag(vagname text, vaglen int64,vagdist int64);"

-S "create virtual table temp.vvec using run_vt(cmd='find \"%s\" -maxdepth 1 -name \"vec*.db\" -exec sqlite3 -separator \",\" \"{}\" \"select vname,vlen,abs(4-vlen) as vdist from vec;\" \\;',cols='myvname,myvlen,myvdist',d=',');

insert into vvecog select '%s'||'/'||myvname,myvlen,myvdist from temp.vvec inner join vrpentries on name=myvname where size < 10000 order by myvdist asc limit 3;

delete from vvecog where vogname not in (select vogname from vvecog order by vodist asc limit 3);

** drop virtual table (once per directory because you run find per directory)

drop table temp.vvec;”

** after all threads complete serialize insert into aggregate table once per run order/limit

-J "insert into vvecag select * from vvecog order by vodist asc limit 3;"

** finally select on the aggregate table once order by distance and limit 3

-G "select vagname,vaglen,vagdist from vvecag order by vagdist asc limit 3;" testidx

testsrc/l1.2/f2.1.doc|4|0

testsrc/l1.3/f3.1.doc|4|0

testsrc/l1.2/f1.1.doc|3|1

Nearest Neighbor example4

crazy better (adjust query as it runs)

Collect all records and order by and limit 3

./gufi_query -n1 -d'|' -a -p "testsrc"

-I "create table vvecog(vogname text, volen int64,vodist int64);"

-K "create temp table vvecag(vagname text, vaglen int64,vagdist int64);"

-S "create virtual table temp.vvec using run_vt(cmd='find \"%s\" -maxdepth 1 -name \"vec*.db\" -exec sqlite3 -separator \",\" \"{}\" \"select vname,vlen,abs(4-vlen) as vdist from vec;\" \\;',cols='myvname,myvlen,myvdist',d=',');

insert into vvecog select '%s'||'/'||myvname,myvlen,myvdist from temp.vvec inner join vrpentries on name=myvname where size < 10000 order by myvdist asc limit 3;

delete from vvecog where vogname not in (select vogname from vvecog order by vodist asc limit 3);

** drop virtual table (once per directory because you run find per directory)

drop table temp.vvec;”

Working on it

** after all threads complete serialize insert into aggregate table once per run order/limit

-J "insert into vvecag select * from vvecog order by vodist asc limit 3;"

** finally select on the aggregate table once order by distance and limit 3

-G "select vagname,vaglen,vagdist from vvecag order by vagdist asc limit 3;" testidx

testsrc/l1.2/f2.1.doc|4|0

testsrc/l1.3/f3.1.doc|4|0

testsrc/l1.2/f1.1.doc|3|1

External DB Vector, gufi can be or gen dynamically a RAG from Exabyte holdings/billions objects / use vector sim to answer a question you have!

```
Query Index with sqlite-lembed
Matching on "icn"
INSERT INTO temp.lembed_models(name, model)
    SELECT 'model', lembed_model_from_file('snowflake-arctic-embed-m-long-q8_0.gguf');

WITH matches AS (
    SELECT
        tinode,
        distance
    FROM embeddings
    WHERE embedding MATCH lembed('model', 'icn')
    ORDER BY distance ASC
    LIMIT 10
)
INSERT INTO intermediate
SELECT
    distance,
    rpath(vrpentries.sname, vrpentries.sroll) || '/' || vrpentries.name
FROM matches
LEFT JOIN vrpentries ON vrpentries.inode == matches.tinode;

0.574531555175781|chistory.snowflake/idx/Bulletins/1991-02-txt-bulletins-included-in-cnews.pdf
0.636207818984985|chistory.snowflake/idx/cnews/1976-12-img-ocr.pdf
0.654783666133881|chistory.snowflake/idx/bits/1994/1994-09.pdf
0.667844891548157|chistory.snowflake/idx/cnews/1977-08-img-ocr.pdf
0.668464779853821|chistory.snowflake/idx/cnews/1977-05-img-ocr.pdf
0.669988930225372|chistory.snowflake/idx/cnews/1986-01-txt.pdf
0.671966791152954|chistory.snowflake/idx/Bulletins/1983-1986-index-txt.pdf
0.672669172286987|chistory.snowflake/idx/cnews/1978-11-img-ocr.pdf
0.673577129840851|chistory.snowflake/idx/cnews/1976-11-img-ocr.pdf
0.676900923252106|chistory.snowflake/idx/cnews/1977-11-img-ocr.pdf
```

AI

The lowest distance document

360000000 24dante 920103162403 0 9201031624dante stext 112/18/85 3 01
Starting in February 1991, the ICN Change Bulletin was incorporated in the Computing and Communications Division News. Issues from February 1991 to the present are available on CFS in /icndoc/news

Gufi virtual tables of particular interest

- **Gufi_vt makes gufi_query look like a table**
- **You can use gufi_sqlite3 (an interactive sqlite3 interface that has all the gufi extensions - or use these from sqlalchemy, duckdb, other)**

- **Gufi_vt_vrpentries (automatically generated fixed all rows all cols+path)**
- **Gufi_vt_vrsummary (automatically generated fixed all rows all cols+path)**
- **Gufi_vt_treesummary (automatically generated fixed all rows all cols+path)**
- **Gufi_vt (must generate using create virt table temp.yourname ….**
  - Can do any query with any rows and cols you choose

- **Because the results from the gufi_query in gufi_vt* tables returns back to your query you can get aggregation (sort/group..) inside your query**

Gufi_vt examples

And if it wasn't powerful enough, making gufi_query look like a table so you can do internal and external db's all appear like a table, like a query that uses gufi native, xattrs, user external db's, full text, and vector similarity lookups all from only the part of the namespace you can see

Above is the fixed field all records table (notice threads 2). Refering to the table forks qufi_query with N threads

Below is non fixed, you provide literally everything gufi_query can do and make it look like a table to consume it with as many hidden threads as you want.

This makes connecting it to other ecosystems easy too, see below

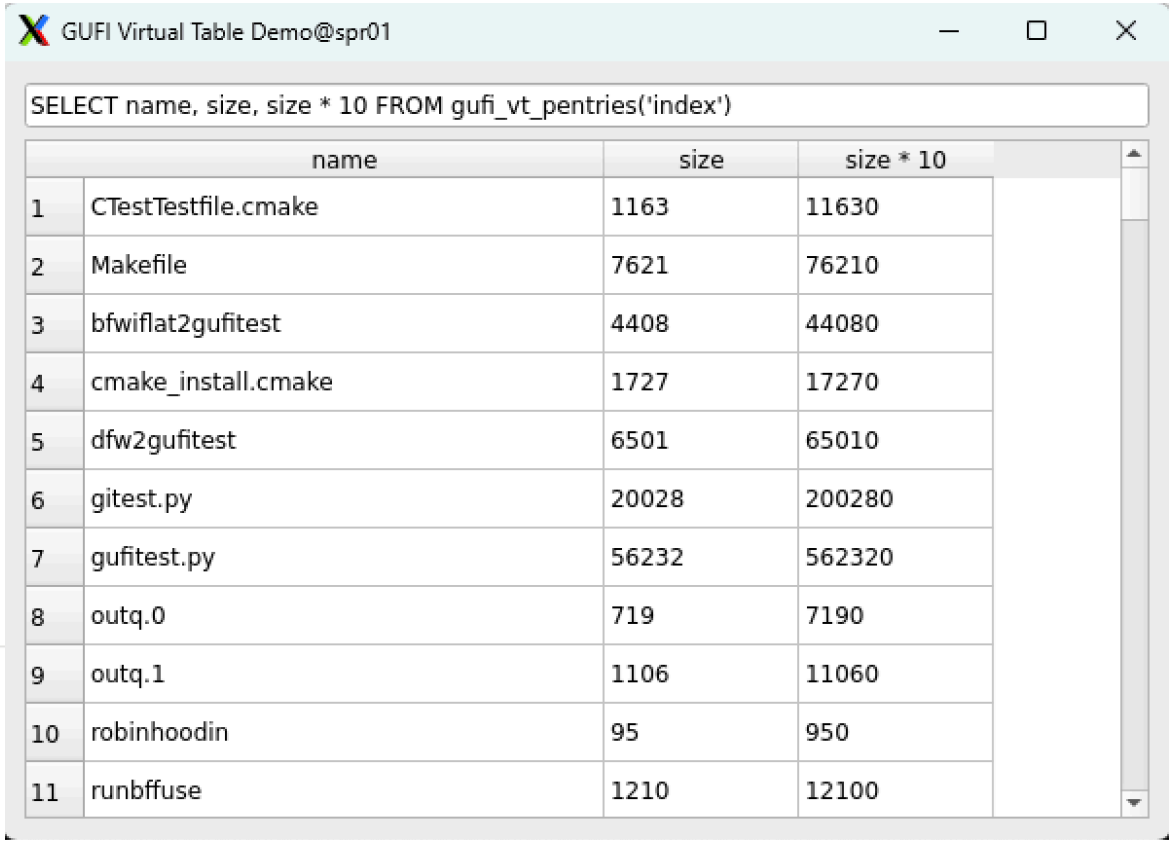
```
[ggrider@pn2201328 src % echo -n "SELECT path||'/'
 '||name,'|',size  FROM gufi_vt_vrpentries('testi
 dx',2) where size>0 order by size desc;" | ./guf
 i_sqlite3
 testidx/l1.3/l2.1/f1.biggest|10
 testidx/l1.3/l2.2/f1.bigger|7
 testidx/l1.3/l2.3/f1.big|4
 testidx/l1.2/f2.1.doc|2
 testidx/l1.1/f1.1.doc|2
 testidx/l1.1/f2.1.doc|2
 testidx/l1.3/f3.1.doc|2
```

```
[ggrider@pn2201328 src % echo -n "CREATE VIRTUAL
 TABLE temp.gufi USING gufi_vt('testidx', E='SE
 LECT path() || '/' || name AS fullpath, xattr_n
 ame, xattr_value,size FROM vrxpentries where si
 ze>0;'); SELECT fullpath,'|', xattr_name,'|' xa
 ttr_value,'|',size FROM gufi ORDER BY size;" |
 ./gufi_sqlite3
 testidx/l1.2/f2.1.doc|||2
 testidx/l1.3/f3.1.doc|||2
 testidx/l1.1/f1.1.doc|||2
 testidx/l1.1/f2.1.doc|user.garyxattr||2
 testidx/l1.3/l2.3/f1.big|||4
 testidx/l1.3/l2.2/f1.bigger|||7
 testidx/l1.3/l2.1/f1.biggest|||10
```


Are there ways to interface to GUFI (Sqlalchemy)

A few lines of python and you have a gui. Gufi can fit into sqlalchemy which makes it fit into that ecosystem

(this is python QTKmpackage)



	name	size	size * 10
1	CTestTestfile.cmake	1163	11630
2	Makefile	7621	76210
3	bfwiflat2gufitest	4408	44080
4	cmake_install.cmake	1727	17270
5	dfw2gufitest	6501	65010
6	gitest.py	20028	200280
7	gufitest.py	56232	562320
8	outq.0	719	7190
9	outq.1	1106	11060
10	robinhoodin	95	950
11	runbffuse	1210	12100

```python
#!/usr/bin/env python3

import argparse
import sys

import sqlalchemy

from PyQt5.QtWidgets import QApplication, QLineEdit, QTableWidget, QTableWidgetItem,
QVBoxLayout, QWidget

class GUFI_VT_Widget(QWidget):
    def __init__(self, engine):
        super(GUFI_VT_Widget, self).__init__()
        self.conn = engine.connect()

        self.setWindowTitle('GUFI Virtual Table Demo')
        self.setGeometry(0, 0, 600, 400)

        self.layout = QVBoxLayout()

        self.sql_box = QLineEdit()
        self.sql_box.setPlaceholderText('SQL')
        self.sql_box.returnPressed.connect(self.run_query)
        self.layout.addWidget(self.sql_box)

        self.table = QTableWidget()
        self.create_table(None)
        self.layout.addWidget(self.table)

        self.setLayout(self.layout)
```

```python
#!/usr/bin/env python3

import argparse
import sys

import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk as gtk

import sqlalchemy

class GUFI_VT_Window(gtk.Window):
    def __init__(self, engine):
        super(GUFI_VT_Window, self).__init__()
        self.conn = engine.connect()

        self.set_title('GUFI Virtual Table Demo')
        self.set_size_request(600, 400)
        self.connect('destroy', self.cleanup)

        # input text box
        self.query_entry = gtk.Entry()

        # button to run SQL
        run_button = gtk.Button.new_with_label('Run')
        run_button.connect('clicked', self.run_query)
```

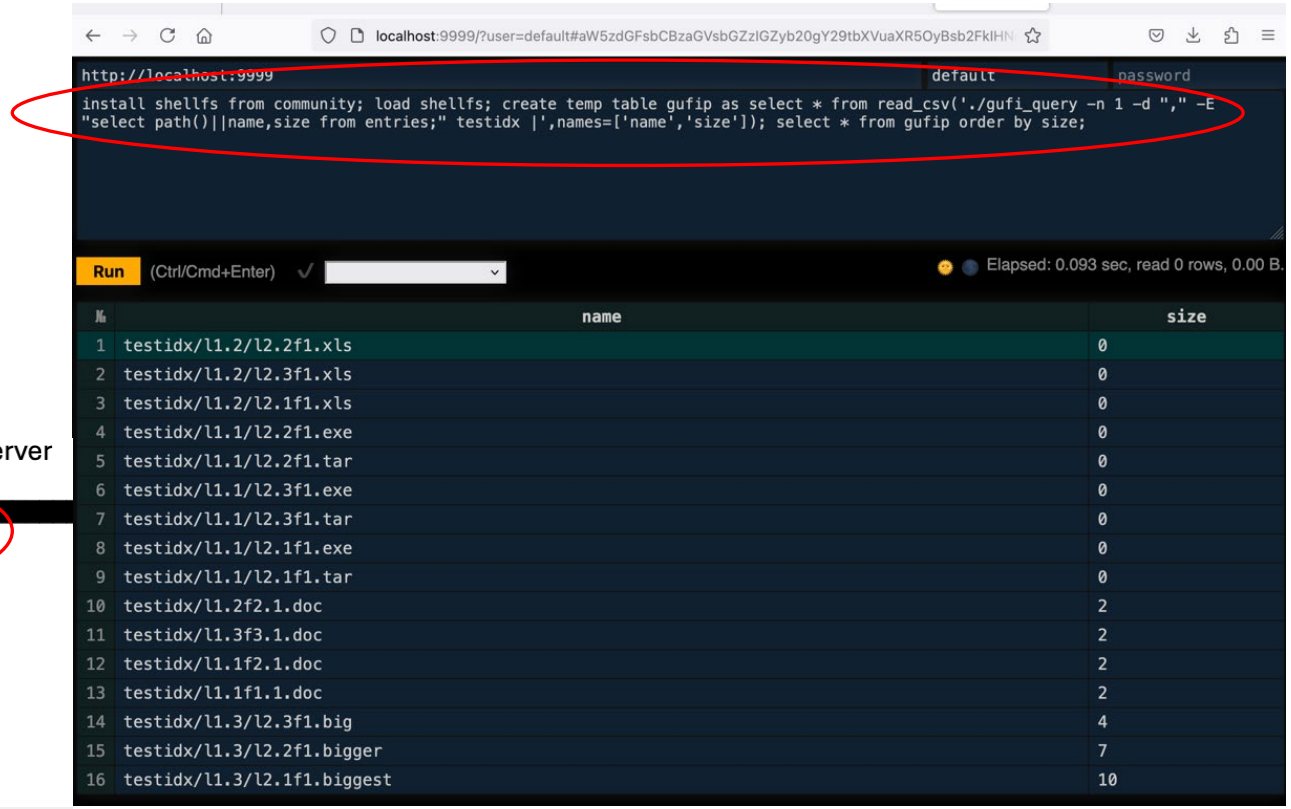

Are their ways to interface to GUFI (DuckDB)

```
D select * from read_csv($$ /gufi_query -n1 -d "," -E "select name,size from vrpentries where name like '%f1%%';" testidx |$$, names=['name','size']) order
by size desc;
```

name varchar	size int64
f1.biggest	10
f1.bigger	7
f1.big	4
f1.1.doc	2
f1.xls	0
f1.xls	0
f1.xls	0
f1.exe	0
f1.tar	0
f1.exe	0
f1.tar	0
f1.exe	0
f1.tar	0

13 rows 2 columns

Gufi can appear as a table to DuckDB and fit into that ecosystem
And duckdb can “finish” query (final sort/group)



Here is a quick example, fire up the http duckdb server

```
D install httpserver from community;
100% ▕████████████████████████████████████████▏
D load httpserver;
D select httpserve_start('localhost',9999,'');
```

httpserve_start('localhost', 9999, '') varchar
HTTP server started on localhost:9999

And on my browser pointed at localhost:9999

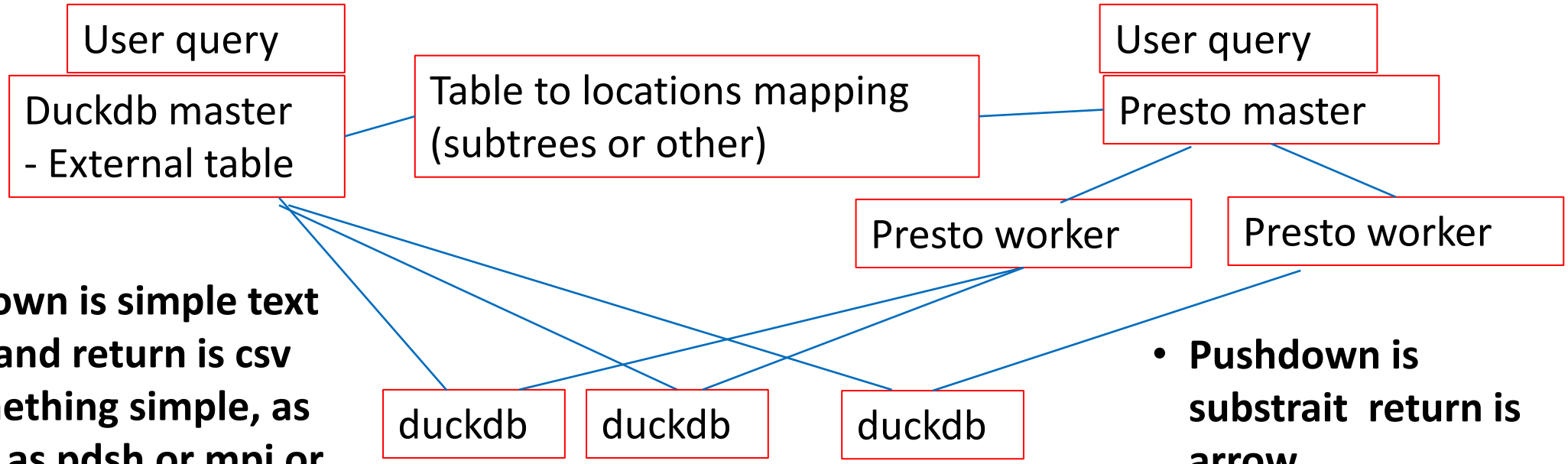
Notice DuckDB can use $$ for a quote

Other existing or relatively easy to provide gufi query related tools/interfaces/concepts

- **Use gufi to make lists of filed/directories to act upon using pftool/condiuit**
  - Conduit can orchestrate action and use pftool to move files in parallel
- **bffuse.c could be updated to work with current gufi indexes, it allows you to start a fuse daemon mounted somewhere and provide it a query and you will only see the files/directories that match the query.**
- **bfresultsfuse.c could be updated to work with current gufi indexes, it allows you to run a gufi_query and create an output database with the results, then you can start a fuse daemon pointed at that database that materializes the results as a mounted tree. (an example would be to create a custom db for a user daily or something)**
- **Given gufi index is just a posix tree and given, once you drop tree summaries and rollups, you can remove/move a subtree anywhere you want it to be, you can even tar up the subtree index and move it to another machine and run gufi on that machine or you can mount a gufi index tree over nfs/smb, or it could be placed in a**
- **Work is underway to use gufi to take a snapshot of statistics about file systems on periodic basis to have longitudinal statistics about your file/storage system contents.**

Can gufi scale to more than one machine (beyond threads). (query pushdown – work to be done)

- **In both cases the master can complete the query (final sort/group by)**



- **Pushdown is simple text query and return is csv or something simple, as simple as pdsh or mpi or other**
- **Shard the tree, either full subtree (easy) and/or follow gufi junctions (to be explored)**
- **Pushdown is substrait return is arrow**

Hopefully you can see the power gufi brings to exabytes of holdings, billions of objects, in any number of storage systems maintaining full need to know security.
Asking questions of our holdings using file/storage, extended attributes, user supplied metadata, extracts from the data itself (text, other file metadata, features, etc.) with added value like full text search and embedded vector similarity. Both thread and cross machine parallelism with connectivity to all your favorite analytics/data science tools including query engines, data frames, inference engines and pick your language (English, html, sql, R, python, C, C++, …).

Administration

- **Creating indexes**
- **Rollup**
- **Treesummary**
- **Incremental updates (in different states of completeness)**
  - Posix
  - Gpfs/campaign
  - Lustre
  - HPSS
  - Other
    - Robinhood extraction (prototyped)
    - Starfish extraction (requested but not started)

Creating index 1

8.1 Directly Indexing a Filesystem

8.1.1 gufi_dir2index

gufi_dir2index is used to directly create an index based off of the contents of a provided directory.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-x	pull xattrs from source file-sys into GUFI
-z <max_level>	maximum level to go down to
-k <filename>	file containing directory names to skip
-M <bytes>	target memory footprint
-C <count>	Number of subdirectories allowed to be enqueued for parallel processing. Any remainders will be processed in-situ
-e	compress work items

8.

gu

Th

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-d <delim>	delimiter (one char) [use 'x' for 0x1E]
-M <bytes>	target memory footprint

Table 3: gufi_trace2index Flags and Arguments

Each source filesystem found in the trace files will be converted to an index placed underneath index_root.

8.2.2 gufi_dir2trace

gufi_dir2trace generates trace files to allow for indexes to be easily transfered to different locations rather than requiring entire trees to be copied around.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-x	pull xattrs from source file-sys into GUFI
-d <delim>	delimiter (one char) [use 'x' for 0x1E]
-k <filename>	file containing directory names to skip
-M <bytes>	target memory footprint
-C <count>	Number of subdirectories allowed to be enqueued for parallel processing. Any remainders will be processed in-situ
-e	compress work items
-q <basename>	Basename of file to keep track of during indexing

Table 2: gufi_dir2trace Flags and Arguments

.2.1 Usage

i_dir2trace [flags] input_dir... output_prefix

ce files with the name output_prefix.${i}, where i ∈ [0, number of threads), be created.

REGIONAL SDC

Creating index 2

8.2.3 gufi_trace2index

gufi trace2index is used to convert trace files into indexes. It is essentially the same as gufi dir2index except it obtains data from trace files instead of the filesystem being indexed.

Per thread trace files may be passed into gufi trace2index directly. Note that passing in too many trace files at once might result in running out of file descriptors. Alternatively, the per thread trace files may be concatenated in any order into a smaller number of larger files for processing.

Extended attributes will be processed if they are found in the traces. There is no need to tell gufi trace2index to process them with -x.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-d <delim>	delimiter (one char) [use 'x' for 0x1E]
-M <bytes>	target memory footprint

Table 3: gufi trace2index Flags and Arguments

Each source filesystem found in the trace files will be converted to an index placed underneath index_root.

8.2.3.1 Usage

```
gufi_trace2index [flags] trace_file...  index_root
```

8.3.3 Usage

Extended attributes are not pulled from the filesystem by default. In order to pull them, pass -x to gufi dir2index or gufi dir2trace.

Note that only xattr pairs in the user namespace (user.*) are extracted.

Rollups

9.1.5 gufi_rollup executable

In order to apply rollup to an index, run the gufi_rollup executable:

```
gufi_rollup index_root
```

Figure 4 shows the overall structure of the gufi_rollup executable. gufi_rollup recursively descends the index and performs the rollup operation on a directory once all of the directory's children have been processed as shown in Figure 5.

9.1.6 Undoing a rollup

To remove rollup data from an index, run the gufi_unrollup executable:

```
gufi_unrollup index_root
```


Treesummary 1

9.2 Generate treesummary Tables

The treesummary table is an optional table that is placed into db.db. It contains a summary of the entire subtree starting at current directory using minimums, maximums, and totals of numerical values.

When a query is provided to gufi query -T, the treesummary table is queried first. Because treesummary tables do not necessarily exist, gufi query first checks for the existence of the treesummary table in the directory being processed before performing the -T query. If the -T query returns no results, the entire subtree will be skipped.

9.2.1 gufi_treesummary

Starting from a directory provided in the command line, gufi treesummary recursively traverses to the bottom of the tree, collecting data from the summary table of each child directory database. If a treesummary table is discovered in a subdirectory, descent down the tree is stopped as the treesummary table contains all of the information about that directory as well as all subdirectories underneath it. Once all of the data has been collected, it is summarized and placed into the treesummary table of the starting directory.

Generating treesummary tables for all directories using this top-down approach will take a long time due to the repeated traversals across the same directories. Because of this, gufi treesummary generates the treesummary table for the provided directory only.

If generating treesummary tables using gufi_treesummary, the tables should be generated at optimal points within the index. For example, if the index is on a home directory, it may be useful to generate treesummary tables at each user's home directory.

Example Call:

```
gufi_treesummary index_root
```

9.2.1 gufi_treesummary

Starting from a directory provided in the command line, gufi treesummary recursively traverses to the bottom of the tree, collecting data from the summary table of each child directory database. If a treesummary table is discovered in a subdirectory, descent down the tree is stopped as the treesummary table contains all of the information about that directory as well as all subdirectories underneath it. Once all of the data has been collected, it is summarized and placed into the treesummary table of the starting directory.

Generating treesummary tables for all directories using this top-down approach will take a long time due to the repeated traversals across the same directories. Because of this, gufi treesummary generates the treesummary table for the provided directory only.

If generating treesummary tables using gufi_treesummary, the tables should be generated at optimal points within the index. For example, if the index is on a home directory, it may be useful to generate treesummary tables at each user's home directory.

Example Call:

```
gufi_treesummary index_root
```


Treesummary 2

9.2.2 gufi_treesummary_all

gufi treesummary all generates treesummary tables for all directories. This is done by walking to the bottom of the tree and generating treesummary tables while walking back up, which only occurs after all subdirectories have had their treesummary tables generated. Leaf directories, by definition, do not have subdirectories, and further traversal down is unnecessary and impossible. Their treesummary tables are thus duplicates of their summary tables, providing the base case for the walk back up the tree. Directories above the leaves can then use the treesummary data found in their immediate subdirectories, which are 1) guaranteed to exist and 2) guaranteed to summarize the entire subdirectory's subtree, to generate their own treesummary tables.

Example Call:

```
gufi_treesummary_all index_root
```

9.2.3 gufi_rollup

Just as with gufi treesummary all, rolling up a tree involves walking to the bottom of the tree and working upwards. This allows for treesummary generation to be performed automatically during the roll up operation, resulting in treesummary tables being generated for all directories whether or not they were rolled up.

Incremental and special loading (work in progress)

- **Incremental approaches**
  - Posix
  - GPFS/Campaign
  - Lustre
  - Hpss db2 query/HPSS log
- **Others**
  - Dump from robinhood (prototypes)
  - Other