SingleFileSystem

This is a basic filesystem wrapper to hide away other parts of the filesystem - and enable usage of single-file tables without much changes.

Suppose we have the following files:
hdfs:///baz/foo.orc
hdfs:///baz/bar.orc

The goal would be to create a table in which only contents of the file foo.orc is visible.

Right now for tables we need to specify directories; so specifying a single file will not work right out of the box - and specifying hdfs:///baz will also implicitly ignore bar.orc as well.

Idea is to add a wrapper filesystem which could hide-away other things; by adding a prefix:

sfs+hdfs:///baz/foo.orc

The above URI could communicate that we are talking about a single file thing - but it will not work correctly since we are converting the path to string and back here and there - so a more sophisticated one will be needed; the URI should

- self-contain if it may show the file
- Should be able to list the target file at the last directory portion; by adding some fixed string; like:

sfs+hdfs:///baz/foo.orc/#SINGLEFILE#

Behaviour for the following paths

- sfs+hdfs:///baz/foo.orc/#SINGLEFILE#/foo.orc
    - File - gives access to the backing file
- sfs+hdfs:///baz/foo.orc/#SINGLEFILE#
    - Directory - which shows foo.orc as a single entry in the directory
- sfs+hdfs:///baz/foo.orc
    - Existing files on the backing filesystem will be shown as a directory Containing a #SINGLEFILE# directory
- sfs+hdfs:///baz/ ; sfs+hdfs:///
    - Existing files and directories are shown as directories

Single file tables could be created by specifying a location

- Which uses an “sfs+” prefix for the filesystem
- And also uses an “imaginary” #SINGLEFILE# directory at the end of the url

sfs+hdfs:///baz/foo.orc/#SINGLEFILE#

Implemented filesystem api methods:

- open() for the targetfile
- status/listing for browsing the original filesystem

Prototype implementation notes

- although the handler will be the same separate scheme -s must be registered for every wrappable fs: sfs+hdfs,sfs+s3, …

Current implementation lives at 2 places

- Hive's codebase contains the filesystem drivers/registrations/etc
- There was some Impala changes necessary to enable the sfs+ prefix to be used

The benefit of this approach is that this will enable to use the same tables/feature from other systems like Hive or even spark (not tested - but theoretically it should work).

```
seq 1 3 > f1 ; seq 1 10 > f2
hdfs dfs -mkdir /tmp/d1 ; hdfs dfs -copyFromLocal f{1,2} /tmp/d1/
export
HADOOP_CLASSPATH=$HOME/hive/ql/target/hive-exec-3.1.3000.7.2.12.0-223
.jar
dev@i10a:~/impala$ hdfs dfs -ls /tmp/d1
Found 2 items
-rw-r--r--   3 dev supergroup          6 2021-09-22 17:59 /tmp/d1/f1
-rw-r--r--   3 dev supergroup         21 2021-09-22 17:59 /tmp/d1/f2
dev@i10a:~/impala$ hdfs dfs -find sfs+hdfs:///tmp/d1
sfs+hdfs:///tmp/d1
sfs+hdfs:///tmp/d1/f1
sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#
sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#/f1
sfs+hdfs:///tmp/d1/f2
sfs+hdfs:///tmp/d1/f2/#SINGLEFILE#
sfs+hdfs:///tmp/d1/f2/#SINGLEFILE#/f2
dev@i10a:~/impala$ hdfs dfs -cat
sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#/f1
1
2
3
dev@i10a:~/impala$ hdfs dfs -ls -r sfs+hdfs:///tmp/d1
Found 2 items
```

```
drwxr-xr-x   - dev supergroup          21 2021-09-22 17:59 sfs+hdfs:///tmp/d1/f2
drwxr-xr-x   - dev supergroup          6 2021-09-22 17:59 sfs+hdfs:///tmp/d1/f1
dev@i10a:~/impala$ hdfs dfs -ls -r sfs+hdfs:///tmp/d1/f1
Found 1 items
drwxr-xr-x   - dev supergroup          6 2021-09-22 17:59 sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#
dev@i10a:~/impala$ hdfs dfs -ls -r sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#
Found 1 items
-rw-r--r--   3 dev supergroup          6 2021-09-22 17:59
sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#/f1
dev@i10a:~/impala$ hdfs dfs -ls -r sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#/f1
-rw-r--r--   3 dev supergroup          6 2021-09-22 17:59
sfs+hdfs:///tmp/d1/f1/#SINGLEFILE#/f1
```

Changes are accessible here:
Hive: https://github.infra.cloudera.com/zhaindrich/hive/tree/dev-sfs3
Impala: https://github.infra.cloudera.com/zhaindrich/impala/tree/dev-sfs3

Changes are also uploaded to gerrit:
https://gerrit.sjc.cloudera.com/c/cdh/impala/+/144114
https://gerrit.sjc.cloudera.com/c/cdh/hive/+/144119

Testing

The following steps are manually tested in a local environment for Impala development. I am using the Impala patch at https://github.infra.cloudera.com/fangyurao/Impala/commit/daa77994ef2864e2d6ac39092bd5c63d01cbed0b revised after a discussion with Zoltan on Sep 23, 2021.

Test data preparation

On the command line, we create a text file containing the contents of the external table to be created and then upload the file to HDFS.

```
source $IMPALA_HOME/bin/impala-config.sh
seq 1 3 > d1.txt
hadoop fs -mkdir hdfs://localhost:20500/test-warehouse/sfs
hadoop fs -copyFromLocal d1.txt
hdfs://localhost:20500/test-warehouse/sfs
```

Basic test

In order for the user non_owner to create an external table associated with the text file uploaded above, we have to grant the following privileges to the user non_owner as the user admin in the Impala shell.

- GRANT ALL ON URI 'sfs+hdfs://localhost:20500/test-warehouse/sfs/d1.txt/#SINGLEFILE#' TO user non_owner;
- GRANT ALL ON URI 'hdfs://localhost:20500/test-warehouse/sfs/d1.txt' TO user non_owner;

The user non_owner will then be able to create an external table associated with the text file we just uploaded to HDFS in the Impala shell.

Note that in Hive, the current implementation only requires the requesting user to possess the privilege on "sfs+hdfs://localhost:20500/test-warehouse/sfs/d1.txt/#SINGLEFILE#". Granting the requesting user the privilege on "hdfs://localhost:20500/test-warehouse/sfs/d1.txt" does not enable the requesting user to create such an external table.

```
[localhost:21050] default> CREATE EXTERNAL TABLE t2s_01 (a STRING)
LOCATION
'sfs+hdfs://localhost:20500/test-warehouse/sfs/d1.txt/#SINGLEFILE#';
Query: CREATE EXTERNAL TABLE t2s_01 (a STRING) LOCATION
'sfs+hdfs://localhost:20500/test-warehouse/sfs/d1.txt/#SINGLEFILE#'
+-------------------------+
| summary                 |
+-------------------------+
| Table has been created. |
+-------------------------+
Fetched 1 row(s) in 0.16s
```

The user non_owner is also able to retrieve the contents in the external table.

```
[localhost:21050] default> SELECT * FROM t2s_01;
Query: SELECT * FROM t2s_01
Query submitted at: 2021-09-23 14:28:17 (Coordinator:
http://fangyu-downstream-dev-7.gce.cloudera.com:25000)
```

```
Query progress can be monitored at:
http://fangyu-downstream-dev-7.gce.cloudera.com:25000/query_plan?query
_id=cb4903c5b2c85d3a:ef3df10e00000000
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
+---+
Fetched 3 row(s) in 1.11s
```

Wildcard URL

It is possible to use wildcard URL's. For instance, we could grant the following privileges to the user `non_owner` as the user admin in the Impala shell so that if the user `non_owner` wants to create other external tables associated with different text files, we do not have to explicitly grant the privilege on each text file.

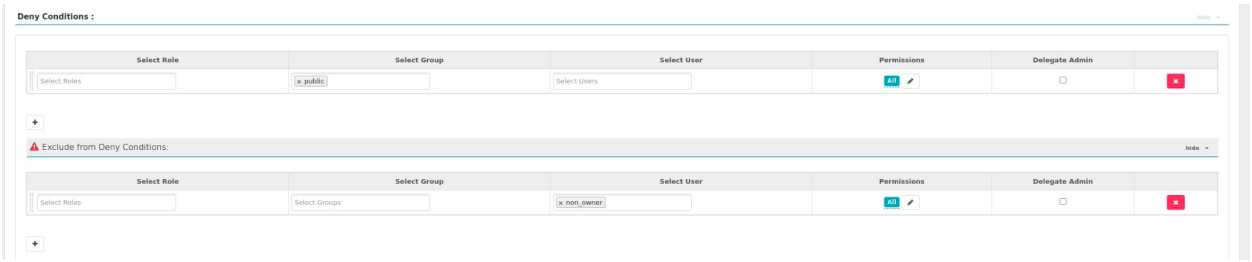
- `GRANT ALL ON URI 'sfs+hdfs://localhost:20500/test-warehouse/sfs/*/#SINGLEFILE#' TO user non_owner;`
- `GRANT ALL ON URI 'hdfs://localhost:20500/test-warehouse/sfs/*' TO user non_owner;`

Deny conditions

It is possible to deny all other users (including the user `admin`) except for the non_owner user the privilege to create such an external table via Ranger's web UI. For each URL that will be checked, we add the following rule to exclude the user `non_owner` from the deny condition.

In the following screenshot, there is a deny policy for the group `public`, which denotes all users. We also have a rule to exclude the user `non_owner` from the deny condition added for the group `public`.

Note that deny conditions could not be managed via the Impala shell now so we have to edit them via Ranger's web UI.


Deny Conditions :
hide
Select Role
Select Group
Select User
Permissions
Delegate Admin
Select Roles
public
Select Users
All
Exclude from Deny Conditions.
hide
Select Role
Select Group
Select User
Permissions
Delegate Admin
Select Roles
Select Groups
non_owner
All