Introduction

Hive currently supports a `CREATE TEMPORARY FUNCTION` syntax that allows users to upload a function for the duration of their connection to Hive. Users would like to be able to register functions permanently in the cluster. This removes the need for users to create their functions every time they wish to use them. It also allows administrators to upload a set of standard functions that users can share.

Terminology note: throughout this document the term function is used. A function can be a traditional UDF (column transformation function), a UDAF (user defined aggregate), a UDTF (user defined table function), or a PTF (partition table function).

Approach

Functions will be registered as objects in a database (which is roughly equivalent to a SQL schema).

Jars for the function will be stored in HDFS under the directory of the database the function is loaded into.

SQL Syntax Changes

Function Syntax

To create a permanent function, the following syntax is proposed:

```
CREATE FUNCTION [<db_name>.]<function_name> AS 'java_class'
      [USING <jarset_name|fileset_name|archiveset_name> [,
<jarset_name|fileset_name|archiveset_name>] ];
```

When the function is used in a query statement, the metastore will be queried to look up the function information, and any named jar/file/archive sets (described below) referenced by the CREATE FUNCTION statement will be added to the environment to resolve the Java class of the function.

Functions will be created in the current database of the current session or, if the function name is qualified using a database name, in the database specified.

The existing CREATE TEMPORARY FUNCTION syntax will still be supported.

Functions can be modified:

```
ALTER FUNCTION [<db_name>.]<function_name> RENAME TO
[<db_name>.]<new_name>;
ALTER FUNCTION [<db_name>.]<function_name> AS 'java_class'
      [USING <jarset_name|fileset_name|archiveset_name> [,
<jarset_name|fileset_name|archiveset_name>] ];
```

Functions can also be dropped from a database.

```
DROP FUNCTION [IF EXISTS] [db_name.]function_name;
```

Functions in a particular database can be listed.

```
SHOW FUNCTIONS FROM <db_name>;
```

Additionally, it will now be valid to refer to a function in a query by including the name of the database the function is in. Thus:

```
SELECT MYDB.MYUDF(a) FROM foo;
```

will be valid syntax.

Named JAR Sets

Sets of JARs can be saved to the warehouse using CREATE JARSET.

```
CREATE JARSET [<db_name>.]<name> USING '<file>' [, <file>];
```

This will create a named set of JARs which can be referenced by the permanent UDF. The files listed in the USING clause (either local files, or in HDFS) will be copied to the database directory. A JAR set could be used to register a 3rd party JAR plus any dependent JARs it may need. If there are multiple UDFs which use this same JAR, they can all refer to the same JAR set when created using CREATE FUNCTION.

In addition to JAR sets, file and archive sets can also be created in a similar manner:

```
CREATE FILESET|ARCHIVESET [<db_name>.]<name> USING '<file>' [,
<file>];
```

The JAR/file/archive set can be modified:

```
ALTER JARSET|FILESET|ARCHIVESET [<db_name>.]<name> USING '<file>' [,
<file>];
ALTER JARSET|FILESET|ARCHIVESET [<db_name>.]<name> RENAME TO
[<db_name>.]<new_name>;
```

The JAR/file/archive set can also be dropped using DROP JARSET/FILESET/ARCHIVESET.

```
DROP JARSET|FILESET|ARCHIVESET [<db_name>.]<name>;
```

JAR/file/archive sets can be listed:

```
SHOW JARSET|FILESET|ARCHIVESET;
```

SQL Compliance

The proposed additions are based on the current CREATE TEMPORARY FUNCTION syntax, which uses non-standard syntax. While the SQL standard does have CREATE FUNCTION syntax, it differs significantly from the current Hive syntax due to the way Hive functions are implemented. For example the SQL standard specifies function parameters as well as the

return type as part of the CREATE FUNCTION statement, whereas in Hive the parameters and return type behavior is determined by the UDF implementation during query compilation time. It would not be possible to support the standard syntax without changes to the way Hive UDF parameter and return type behavior works, which are not part of the scope of this proposal.

The notion of JAR sets (and their use by functions) is not part of the SQL standard. The standard does not appear to have a mechanism to register library files to the database. As for the usage of libraries/files by UDFs, there does not seem to be a designated place for these in the SQL standard syntax. Some databases specify the library name as a non-standard extension to the syntax, which is similar to the approach taken here. Other databases specify the library name as part of the string representing the function's external name (equivalent to the class name of the function), which must then be parsed to get the various parts.

Security

In addition to making functions permanent, functions need to be secured. There are several points that need securing:

1. Control of who can add jars to Hive.
2. Control of who can create and drop functions.
3. Control of who can execute a function.

These will be added as new privileges to the Hive authorization system.
Since functions are tied to databases, a database owner needs to be able to add jars, create functions, and drop functions in his database. However, currently Hive does not have the ability in its execution framework to protect data in one database from functions uploaded in another database. For this reason, initially, only site administrators should be able to add jars and create or drop functions. However, the system should still be built with the ability for these permissions to be given to database owners so that once Hive can properly secure data across databases this feature can be enabled.

Granting execution of a function to user will follow the standard SQL semantics. Users will also require proper permissions to alter/drop.

```
GRANT EXECUTE|ALTER|DROP ON function_name TO grantee [WITH GRANT
OPTION];
REVOKE [GRANT OPTION FOR] EXECUTE|ALTER|DROP ON function_name FROM
grantee;
```

Use of JAR/file/archive sets will require USAGE privileges. Users will also require proper permissions to alter/drop.

```
GRANT USAGE|ALTER|DROP ON jarset_name TO grantee [WITH GRANT OPTION];
REVOKE [GRANT OPTION FOR] USAGE|ALTER|DROP ON jarset_name FROM
grantee;
```

Database(schema) for Functions

This proposal includes allowing functions to be defined with an owning database/schema. Created functions will be created under the user's current database.

The built-in Hive functions will be registered with the some system database name, such as "sys". An alternative would be to for built-ins to be registered without a database name.

To preserve existing behavior with temporary UDFs, any temporary UDFs created without a database name will use the same database name as that of built-in functions.

When resolving function names which are not qualified with a database name, the following database names will be used, in the following order:
1) Database name of the built-in functions
2) Current database name

Metastore Type Changes

Function-related type changes:

```
// User-defined function
struct Function {
  1: string           dbName,
  2: string           name,
  3: string           className,
  4: string           owner,
  5: i32              createTime,
  6: list<FileSet>    fileSets,
}
```

JAR/file/archive set related type changes:

```
// Represents a single file in a FileSet
struct StoredFile {
  1: i32 createTime,

  // When creating files, user should specify the source HDFS path to
the file.
  2: optional string sourcePath,

  // When retrieving the file info from the metastore,
  // storedPath (containing path to the warehouse copy of the file)
  //  will be filled out rather than sourcePath
  3: optional string storedPath,
}
```

```
enum FileSetType {
  JAR     = 1,
  FILE    = 2,
  ARCHIVE = 3,
}

// A set of JARs/files/archives
struct FileSet {
  1: FileSetType      type,
  2: string           dbName,
  3: string           name,
  4: string             owner,
  5: i32              createTime,
  6: list<StoredFile> files,
}
```

Metastore API Changes

void createFunction(Function newFunc)
Function getFunction(String dbName, String funcName)
boolean dropFunction(String dbName, String funcName)
void alterFunction(String dbName, String funcName, Function newFunc)
void createFileSet(FileSet fs)
FileSet getFileSet(String dbName, String name)
boolean dropFileSet(String dbName, String name)
void alterFileSet(String dbName, String name, FileSet newFs)

Function Registry Changes:

If function entry does not exist for the function name, then perform metastore lookup for that function name. If found, load any applicable JARs/files/archives, and add the new UDF entry to the function registry. Loaded JARs are added to the Map-Reduce tasks using the -libjars option, which will allow them to resolve the registered UDF classes during plan deserialization.