Context:
Currently, the default locations for hive tables, for all users, can only be central directories, one for managed tables and one for external tables that are separately configurable. There is no further breakdown of user spaces within these central repos. Customers would like to be able to separate the spaces for internal tenants/users, to be able to apply storage quotas and other common policies. The following config properties currently determine where the table data resides for hive tables within a database.

`#1 metastore.warehouse.dir:` Root directory for Hive warehouse where all the managed (ACID) tables reside across all user databases in hive. Under here are separate directories for the databases under which are individual tables spaces. So for a managed table named "**managed_table**" in database "**testdb**", the default path would look something like `<metastore.warehouse.dir>/testdb.db/managed_table`.

`#2 metastore.warehouse.external.dir:` Default root directory for external tables across all user databases in hive. While tables of type *EXTERNAL* can reside anywhere outside of the managed warehouse root directory, the default location for such tables is controlled by the value of this property.

So a Hive DB, generally has 2 storage locations, one or both of them are implicit. But only one of these locations is persisted in the metastore database.

On Managed table creation, hive enforces that this table is within the managed warehouse tablespace.

- Managed table creation will no longer accept a location in the create statement.
- If the database has a MANAGEDLOCATION set, a location based on this value is assigned to the table.
- If the database does not have MANAGEDLOCATION set, a default location is assigned, based on #1.

External tables with location specified,

- If the location is outside the managed tablespace, it is legal and thus the location set on the database is irrelevant. The location set in the create table statement is honored.
- If the location is within the managed tablespace, the creation of the table fails as it is illegal for an external table to be residing in the managed tablespace.

External tables without a location in the create table statement, are assigned a location in the following order of precedence.

- If the database has a location set, a directory within this location.
- If the database does not have a location set, a default location based on #2 above is assigned.

Problem:
Both these locations are based on a central location which applies to all databases/tables created by all users. While the metastore is a bit lenient for external tables, it enforces the strict location constraints for transactional managed tables. There is no support for having a tenant-based central storage repository for all DBs/tables. Having such support would be useful in setting up common management policies like

- Tenant based quotas.
- Tenant based backup and replication policies.
- Tenant based encryption policies.
- Tenant based resource allocations, if applicable.

Design:
Managed Tables: Going forward, HMS will allow the location to be specified either on the database or on the table, but not both. So if the MANAGED_LOCATION is set on the database, we expect that the create table will not have a location set. If it is used, the creation of the table fails.

External tables: For external tables, Hive metastore will continue to support location on the database or be overridden for any given table.

DBS Schema table (Metastore):
Add a new varchar-type column to the DBS table of the schema. This will be the location of all *MANAGED* tables for this database. The existing DB_LOCATION_URI will now be treated as table space for *EXTERNAL* tables.

```
CREATE TABLE DBS
(
    DB_ID NUMBER NOT NULL,
    "DESC" VARCHAR2(4000) NULL,
    DB_LOCATION_URI VARCHAR2(4000) NOT NULL,
    DB_MANAGED_LOCATION_URI VARCHAR2(4000) NULL,
    "NAME" VARCHAR2(128) NULL,
    OWNER_NAME VARCHAR2(128) NULL,
    OWNER_TYPE VARCHAR2(10) NULL,
    CTLG_NAME VARCHAR2(256) DEFAULT 'hive',
    CREATE_TIME NUMBER (10)
);
```

MDatabase: Add a new location element to the model class *MDatabase* “managedLocationUri”

```
String name;
String locationUri;
String managedLocationUri;
String description;
Map<String, String> parameters;
String ownerName;
String ownerType;
String catalogName;
int createTime;
```

Thrift Changes (Metastore):
The current thrift struct for Database will be modified to return this “managed” location.

```
struct Database {
 1: string name,
 2: string description,
 3: string locationUri,
```

```
 4: map<string, string> parameters, // properties associated with the
database
 5: optional PrincipalPrivilegeSet privileges,
 6: optional string ownerName,
 7: optional PrincipalType ownerType,
 8: optional string catalogName,
 9: optional i32 createTime,               // creation time of database
in seconds since epoch
10: optional string managedLocationUri
}
```

This will add new accessors and mutators on the `org.apache.hadoop.hive.metastore.api.Database`

```
public String getManagedLocationUri() {
 return this.managedLocationUri;
}

public void setManagedLocationUri(String locationUri) {
 this.managedLocationUri = locationUri;
}
```

Hive QL Grammer changes (HiveServer):
The syntax for the Hive query language is to be enhanced to add support for being able to specify "managedLocation" for the database both in *CREATE* and *ALTER* commands.

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT database_comment]
  [LOCATION hdfs_path]
  [MANAGEDLOCATION hdfs_path]
  [WITH DBPROPERTIES (property_name=property_value, ...)];

ALTER (DATABASE|SCHEMA) database_name SET MANAGEDLOCATION hdfs_path;
```

Changes to Query Processing (HiveServer):

- `CREATE DATABASE` commands without *MANAGEDLOCATION* are legal. In the absence of *MANAGEDLOCATION*, HS2 or HMS will assign a default value based on "`metastore.warehouse.dir`" setting of the hive configuration. If a *MANAGEDLOCATION* is specified, there will be no restrictions on the path. HMS will not enforce that this be within the

"`metastore.warehouse.dir`". The user creating this database is required to have "write" permissions to that directory. The operation should fail otherwise.

- `CREATE DATABASE` commands without *LOCATION* will now assign a default location based on the "metastore.external.warehouse.dir" setting of the hive configuration.. Currently, if this value is not set, it defaults to using "metastore.warehouse.dir" which is wrong. This bug should be addressed either by setting a default value in HiveConf/MetastoreConf or by having a [predictable] path generated at runtime based on some criteria.

- ALTER DATABASE commands to set *MANAGEDLOCATION* will accept all valid DFS locations. Create the path, if it does not exist, as part of this execution. The users executing this command are required to have "write" permissions to this location, fail the operation otherwise.

- DESCRIBE DATABASE will now also display Managed Location for the database in addition to the current Location. Perhaps we should consider changing the display name in beeline to say "External Location" for the location element.

- DROP DATABASE does not need any changes to the language but perhaps change in execution plan to delete the *MANAGEDLOCATION* directory on the DFS as part of this task. Also EXTERNAL tables that have auto.purge set to true are to be purged from the *LOCATION*.
  `DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];`

Hive metastore side changes:

- Enforce a constraint that *LOCATION* and *MANAGEDLOCATION* cannot be the same. Create database should fail if they are the same.

- MANAGEDLOCATION is allowed to be empty/null.

- If DBS has a *MANAGEDLOCATION*, then use it as default for ACID tables that do not have a location. If *MANAGEDLOCATION* is NULL, use "`hive.warehouse.dir`" to determine the location of the ACID table. If "`create table`" specifies a location that is outside of *MANAGEDLOCATION* and "`hive.warehouse.dir`", fail to create the table. (TBD)

- If *LOCATION* is set, use it as default for all external tables that do not have a location specified in the "`create table`" operation. If a location is specified in the create table operation, then honor it for external tables ignoring the *LOCATION* for the DB.

Metatool Changes:

Hive *metatool* command line tool has support for bulk-manipulating *LOCATION* values. Such support should also be added to manipulate *MANAGEDLOCATION*.

Upgrade scenarios:

For customers migrating from an older release, we will have a new column in the *DBS* table that will be null for existing rows. Users can set/modify the *MANAGEDLOCATION* using one of the following means.

- Use "*`ALTER DATABASE SET MANAGEDLOCATION`*..."
- Using *metatool* that has support for bulk updates to managed location.

Usage (End users):

To enforce something like storage quotas for hive tenants, end users will have to follow some practices/recommendations.

Plan on having a consistent and predictable storage model. Something along like this where the tenant specific storage is forked off at the root followed by separate spaces for managed and external tables or have a common root for all tenants and fork off tenant spaces deeper in the path with then seperate spaces for managed and external tables.

```
/<tenant1>/<warehouse_dir>/managed/…
/<tenant1>/<external_warehouse_dir>/…
Or
<warehouse_dir>/<tenant1>/managed/…
<warehouse_dir>/<tenant1>/external/…
```

Admins can then set quotas on the <tenant> directories. Existing databases and tables can be moved into newly assigned tenant spaces by copying/moving files outside of hive (to preserve hive resources) and then changing the DBS/Table metadata, either metatool or beeline's alter, to point to its new location.

Assumptions (to make this work):

- Users/Admins will assign tenants spaces and limits.
- All Databases and Tables in Hive are created using a location specified in the "create .. " statement or will set the location correctly post-creation via an "alter database .." statement. This ensures that users are creating tables that reside within their root storage location.
- Tenant is a logical group and so hive cannot enforce access controls for storage directories. Hive will assume that users are well-behaved. Hive will not prevent *tenant1* from creating a database cross-referencing *tenant2's* userspace. This has to be done externally via Ranger or DFS permissions.