HDF5 search tool design decisions:

Design 1

No Cache

- ***Load Data to MongoDB***
  - All attributes and attribute values are stored as strings
  - Attributes are divided into 2 groups
    - Metadata file attribute
    - Image file attribute
  - There are 3 collections in the database
    - Metadatafile: Each metadata file attributes and values are stored and represented in one record that is created per metadatafile.
    - Images: Each image file metadata attributes and values are stored and represented in one record that is created per image file
    - Attributes: Each attribute is represented with one record.
      - Attribute name
      - Attribute type (Either metadata or image file attribute)
      - This assumption is important since the user query can include attributes both from metadata file and the image file.

- ***Search on MongoDB***
  - The user query is checked to validate whether it meets the query standards of this module
  - User query can include any number of “AND” and “OR”
    - Attributes/values combined with “AND” are collected in one search statement. DB is queried with this statement for the records meeting the criterion collected in this set.
    - The query parts that are separated with “OR” is treated as different search statements,each of which is run through the MongoDB
    - After each of the statements separated by “OR”s queried all the results are printed
    - Currently there is no filter implemented to eliminate duplicates returned by separate statements of the same query.
      - This means if the query is “statement1 OR statement2” and if DB is queried for “statement1” and then for “statement2” and record A is returned from both of the searches the record A is printed twice
    - Attributes separated with “AND” are collected together.
      - These attributes are clustered in two different groups.
        - Metadata file attributes
        - Image file attributes
      - The type of the attributes are already stored in the DB in “attributes” collections
      - The DB is queried to get the type of the attribute every time it is encountered to be clustered.
      - This means the more attributes a query consists the more DB queries will be involved
        - This additional mongo queries will be reflected on search performance. And this will make the search time heavily rely on DB performance and the network.

Design 2

Cache Attribute Groups:

Attributes types(Metadata file or image file) are loaded into memory

- ***Load Data to MongoDB***
  - Loading the data is same as *Design 1.* Same assumptions and design principles apply.
- ***Search on MongoDB***
  - The search module starts with loading the attributes types to memory
  - Before user input allowed:
    - The DB is queried for “Attributes” collection to load all its records into memory
  - After this caching is complete prompt is returned to user for input queries
  - The processing of the user query is same as it explained in Design 1:
    - The user query is parsed and the attributes being asked for is clustered in two sets, “Metadata file” and “Image file” attributes
    - Unlike the first design this clustering process does not involve any DB operations
      - Instead the cached data is referred for necessary information to cluster the attributes.
      - This eliminates the additional DB queries of the first design which provides better times performing user query
      - The concerns:
        - The bigger the number of attributes registered in the DB, the bigger the size of the data to be cached.
        - For sizes after a certain threshold memory problems or slowdown is possible

Design 3

Embedded attributes:

Embedding all the metadata

- ***Load Data to MongoDB***
  - The idea is to keep embedding the all the attributes in a hierarchy
    - Say the file structure is the following:
      - test/
        - metafile.1
        - test1/
          - metafile.2
          - image.1
          - image.2
          - image.3
    - There are 2 recursive metafiles each of which specify attribute name/

value pairs for the directory level they are in.
    - The images at the bottom of hierarchy also contains individual metadata
    - Based on the directory hierarchy defined the related metadata for image one is;
        - metafile.1 + metafile.2 + image.1_metadata
        - metafile.2 may or may not have the same attribute name\value pairs defined in metafile.2
        - image.1 specifies different attribute name\value pairs than that of metafiles.
- In this design;
    - There is one record created for each metafile in each level
        - This record is composed of all the attribute name/value pairs and the hierarchy information (group/subgroup)
    - The metadata information is accumulated recursively for each level. This means:
        - The DB record for metafile.1 is only composed of attributes from metafile.1
        - The DB record for metafile.2 is composed of attributes from metafile.1 and metafile.2
        - The DB record for image metadata is composed of attributes from metafile.1 and metafile.2 and attributes from image.1
        - The information from the previous level is passed to the next one and embedded to the record of the next level
        - Attribute name/value pairs that have been defined in multiple metafiles in the hierarchy are overwritten for each embedding process. So;
            - If attribute “a” has a value both in metafile.1 and metafile.2 the value in the metafile.2 overwrites the value captured from metafile.1
            - This actually does not create a problem if the user query involves values from the metafile.1 since in that case all the files under the “test” directory are returned to the user

- The concerns:
    - The records size can get bigger and bigger for records created for files lower in the hiearchy
        - In the examples we have. There are 96 attributes in metafiles and 6 in image files.
        - So, for image files instead of creating a record of ~6 fields we will create a record of ~102 fields
        - Based on our tests with MongoDB, the record size actually affects the query performance, graph-recordsizeVSquerytime.pdf
        -
    - Larger the record size is lesser the number of records can be held in memory at a time.

- Although this DB design leads to a simpler search schema the concerns listed above favors the previous design options

- ***Search on MongoDB***

- Like “Design 1” this module gives the prompt the moment it is started.No initial caching done before.
- As explained in the previous designs the user query is checked to validate the standards of the system.
- If not an error message and example query showed to the user.
- There is a “help” option which aims to help the user creating their query
- Like in the previous designs, user query is grouped into subqueries based on the “AND” and “OR” statements involved.
- The difference with the previous designs is, the attributes specified in the user query is not clustered based on their source(metafile or image)
- Each image record also holds the metafile information on its hierarchy.
- if a user query includes specifics both an image metadata and a metafile metadata from both sources the records,
    - Say attribute “a” is an image and “b” is an metafile specific information
    - A DB query {a: 1, b: 8} will not return null since it is possible to find “a” and “b” at the same record.
    - This is possible since while the HDF5 is loaded into the DB all the related information are embedded in a single record for each image object.
- The search returns user the whole hierarchy should a record is found.