

Database Project

MMDA - Multi-Media Database Aggregator

Jonathan Snyder
Christopher Snyder

CMSC424: Database Design
University of Maryland
College Park, MD

Table of Contents

[Table of Contents](#)

[Phase I](#)

[Environment & Requirement Analysis](#)

[Introduction](#)

[Project Scope](#)

[Assumptions](#)

[Technical / Conceptual Problems](#)

[Systems Analysis and Specifications](#)

[Software / Resource Specifications](#)

[Top Level Flow Diagram](#)

[Tasks](#)

[Insert Media File Task](#)

[Bulk Add Task](#)

[Add Web Page Task](#)

[HTML Link Parser](#)

[Apache Tika](#)

[Query Documents Task](#)

[Time Report Task](#)

[Orphan Report Task](#)

[Reach Report Task](#)

[Phase II](#)

[Purpose of Phase II](#)

[Problems Encountered](#)

[ER Model](#)

[Relational Schema](#)

[Pseudo Code#](#)

[Insert File Task](#)

[Edit File Task](#)

[Bulk Add Task](#)

[Apache Tika](#)

[Add Web Page Task](#)

[HTML Link Parser](#)

[Deletion Task#](#)

[Query Documents Task](#)

[Time Report Task](#)

[Orphan Report Task](#)

[Reach Report Task](#)

[Phase III](#)

[Introduction to Phase III](#)

[Problems Encountered](#)

[Revisions to the Relation Schema](#)

User's Manual

[Adding a File's Metadata to the Database](#)

[Adding a Webpage's Metadata to the Database](#)

[Find DAGRs by keyword](#)

[Performing Custom Queries](#)

[Reporting](#)

Future Considerations/Performance

[Adding Large Webpages](#)

[SQL Injection Concerns](#)

Phase I

Environment & Requirement Analysis

Introduction

The purpose of this document is to provide informational and activity needs for the creation of a Multi-Media Database Aggregator. The document will provide the goals of our database system as well as a top level view of the tasks and interfaces used in our database and the interactions of these objects.

Project Scope

The scope of this project involves multiple tasks.

The first task involves researching information about the metadata stored in a variety of different files and looking for the relationships among this information.

The second includes figuring out what types of files we want to support (or support them all) and get a broad idea of how we could store this information. We then would create our database structure based off of this research.

The third task would be to create a simple interface with PHP to input single files as well as an entire folder (with the option to add only items in the root or recursively add items from all subfolders as well). This will initiate an “insert” subroutine that will parse the metadata from the data and store the data accordingly into the database.

The fourth task would be to create a Google Chrome or Mozilla Firefox extension that will allow the user to add a webpage to the database system. The “insert” subroutine will add the information about the webpage (such as URL, Title, etc) as well as add the images, downloadable files, and other webpage links found on the webpage. These documents will also be inserted into the database with information about the relationship between these documents and its parent webpage.

The fourth task would be to create a web interface with PHP that allows the user to query the metadata stored with a number of different variables. This searching feature will be generic to the type(s) of files they are searching. The purpose of this is not to provide any type of application for the database, but rather to show how the database system is working correctly.

Assumptions

- We assume that the user can read and understand English
- We assume that this database is only used by a single user and that they have the database system installed on their own web hosting platform or run on their computer.
 - I.e., we will not have user accounts to allow multiple users.
- We also assume that the user has Internet access.
- We assume that the files are accessible by the user without additional credentials.

Technical / Conceptual Problems

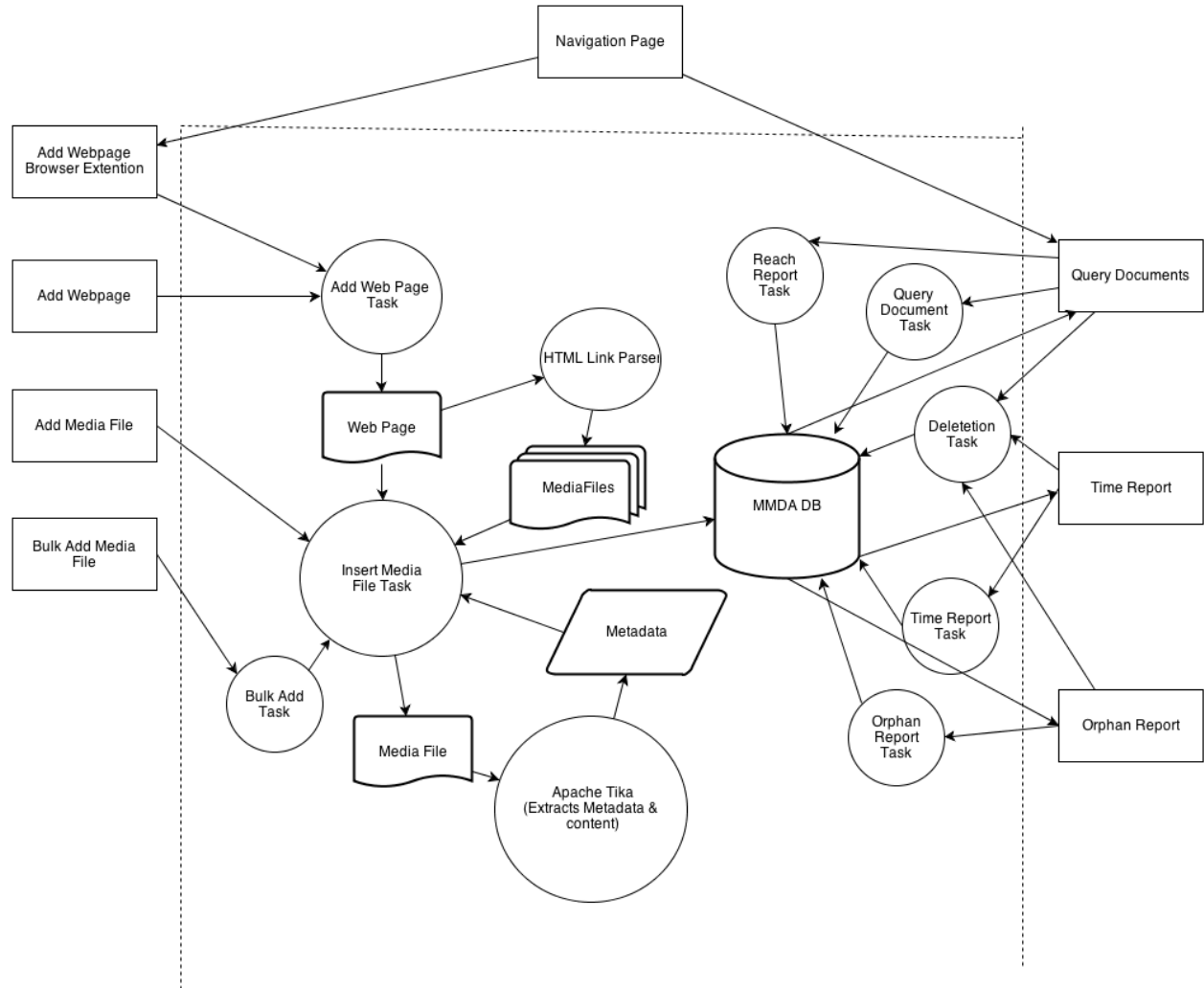
1. **Problem:** What to do if the document being added is an update of an already existing document. (ie. updating document information)
Solution: Before the addition of a document, the system will calculate a unique id for the document (such as a hash code). If this id is already existent in the database. The system will rather only update this information if the modified date (grabbed from the metadata) is newer the the document existent on the database. This functionality will be useful when the user wants to bulk add a folder that has been added before.
2. **Problem:** Parsing the websites for downloadable files, images and other webpages.
Solution: After some research, were confident we can create a PHP script using regular expressions and methods that will allow use to get the URL of all documents on a web page. This program will allow us to easily parse for file URLs that we can throw over to another task that will add the document information. For webpage links, this subroutine will only add webpages that are the first layer. Therefore, we will not recursively add the contents of the webpages that are linked to the original webpage. (Otherwise this subroutine may never end.)
3. **Problem:** Whether to create a Firefox or Google Chrome extension as a quick shortcut to parse the current web page and add it and its referencing documents.
Solution: After some research we decided to make the extension for the Chrome browser over the Firefox browser because of the ease of development and that the Chrome browser is used by more people, and would be more effective of a tool.
http://www.w3schools.com/browsers/browsers_stats.asp
4. **Problem:** How to determine how many types of documents we should support, or whether we should support them all.
Solution: We will support the most common attributes for the most common file types. For example, a word document will be supported. Some of the attributes included for this type will be title, author, and word count. For anything else that is not supported we will only store the metadata in the parent relation, that stores generic attributes common in all files types such as title and modification date. We as well will record what this file type is, as an attribute.
5. **Problem:** Identify what to do on duplicate entries.
Solution: On insert we will compare the file name and the hash of the file that is about to be inserted into the database with the filenames and hashes of the files that are in the database to find potential duplicates. If the hashes match then the user will be prompted asking if it is the same file. In almost all cases this will be the case. This will prevent duplicate entries.

Systems Analysis and Specifications

Software / Resource Specifications

This application will be hosted on a basic LAMP (Linux, Apache, MySQL, PHP) stack hosted ourselves. The application will be written almost entirely in PHP and will use a MySQL database. We will be using Apache Tika to help parse and return the metadata of the documents. To ease in inserting / updating documents from a website, a browser extension will be written in for the Chrome browser. We decided to make the extension for the Chrome browser over the Firefox browser because of the ease of development and that the Chrome browser is used by more people, and would be more effective of a tool.

Top Level Flow Diagram



Tasks

Insert Media File Task

TASK NAME:	Insert Media File Task
PERFORMER:	PHP / MySql
PURPOSE:	To Insert a Document's metadata to the database.
ENABLING COND:	A users request to add a document or the document is linked to from a web page.
DESCRIPTION:	This task adds a document's metadata and content to the database. This task will not bother to add this document to the database if the calculated hash (using the MDF subtask) is already stored in the database.
FREQUENCY:	Every time a document's metadata needs to be added or updated.
DURATION:	Varies, Depends on Apache Tika Parsing time as well as the document retrieval time.
IMPORTANCE:	Critical
INPUT:	Course information provided by the user and a link to the document to be inputed
OUTPUT:	Data relation to be inserted into the database.
SUBTASKS:	Apache Tika, MDF (hash generation)
ERROR COND:	Required fields are not filled out / the file is not readable

Bulk Add Task

TASK NAME:	Bulk Add Task
PERFORMER:	PHP / MySql
PURPOSE:	To Insert many Document's metadata to the database.
ENABLING COND:	A users request to add multiple documents into the database at once.
DESCRIPTION:	This task iteratively adds multiple document's metadata and content to the database. This task will not bother to add this document to the database if the calculated hash (using the MDF subtask) is already stored in the database.
FREQUENCY:	Every time a document's metadata needs to be added or updated.
DURATION:	Varies, Depends on Apache Tika Parsing time as well as the amount of documents needed to be added
IMPORTANCE:	Critical
INPUT:	Folder directory to add all documents from
OUTPUT:	Data relation to be inserted into the database.
SUBTASKS:	Apache Tika, MDF (hash generation)
ERROR COND:	Required fields are not filled out / the folder is not readable

Add Web Page Task

TASK NAME:	Add Web Page Task
PERFORMER:	PHP / MySql
PURPOSE:	To Insert a webpage into the database as well as set up references to metadata of files (images and downloadable content) from the page into the database.
ENABLING COND:	A users request to add a webpage to the database. Either from the Google Chrome addon or from the interface.
DESCRIPTION:	This task uses the "Insert Media File Task" to insert the webpage as well as the content the webpage contains. After the insertion, the webpage's content UUIDs will get referenced to the webpage's UUID
FREQUENCY:	Every time a webpage's metadata is initially requested to be inserted. (This task is not used for webpages found from other webpages.)
DURATION:	Varies, Depends on Apache Tika Parsing time as well as the amount of content on the page.
IMPORTANCE:	Moderate
INPUT:	A URL from the Chrome application or interface.
OUTPUT:	None, it only calls other functions.
SUBTASKS:	HTML Link Parser
ERROR COND:	Webpage does not exist or isn't valid (404's are acceptable).

HTML Link Parser

TASK NAME:	HTML Link Parser
PERFORMER:	PHP
PURPOSE:	To find all linked to documents on a web page as candidates for insertion into the database.
ENABLING COND:	The request from the Insert/Update Document Task to find additional documents to parse when adding a web page document.
DESCRIPTION:	This task extracts links to other documents using regex.
FREQUENCY:	Every time that a document is added that is a web page.
DURATION:	Minimal, but depends on the length of the document
IMPORTANCE:	Nominal
INPUT:	A url of the document to be parsed.
OUTPUT:	A set of the documents that are linked to.
SUBTASKS:	none
ERROR COND:	The web page is not readable.

Apache Tika

TASK NAME:	Apache Tika
PERFORMER:	Apache Tika
PURPOSE:	To obtain all well defined metadata defined by the document.
ENABLING COND:	A request from Insert/Update Document Task for the metadata of a document
DESCRIPTION:	This task extracts the metadata from a file.
FREQUENCY:	Every time a document needs to be added or updated.
DURATION:	Varies, Depends on the size of the file.
IMPORTANCE:	Critical
INPUT:	Document
OUTPUT:	XML formatted metadata
SUBTASKS:	none
ERROR COND:	The file is not readable

Query Documents Task

TASK NAME:	Query Documents Task
PERFORMER:	PHP / MySql / Ajax
PURPOSE:	To report to the user the results from a search as defined by the user.
ENABLING COND:	A request from the user.
DESCRIPTION:	This task queries the database for all documents limited by the user's optionally defined restrictions. The user will then be given the opportunity to delete all or some of these documents from the database.
FREQUENCY:	Whenever the user requests
DURATION:	Varies, Depends on number of documents in the database.
IMPORTANCE:	Important
INPUT:	Optional parameters to restrict the course list by. Example: by year, semester, course.
OUTPUT:	The results of the query formatted in a table where each row is linked to another page which shows the documents which are linked to the course. The output is returned via ajax
SUBTASKS:	Deletion Task
ERROR COND:	Invalid parameters are provided by the user

Time Report Task

TASK NAME:	Time Report Task
PERFORMER:	PHP / MySql
PURPOSE:	To report the user the documents in the database based off of the insertion date.
ENABLING COND:	A request from the user.
DESCRIPTION:	This task queries the database for all documents, limited by the user's defined date parameters. The user will then be given the opportunity to delete all or some of these documents from the database.
FREQUENCY:	Whenever the user requests
DURATION:	Varies, Depends on number of documents in the database.
IMPORTANCE:	Important
INPUT:	Date interval for documents
OUTPUT:	The results of the query formatted in a table. The output is returned via ajax
SUBTASKS:	Deletion Task
ERROR COND:	Invalid parameters are provided by the user

Orphan Report Task

TASK NAME:	Orphan Report Task
PERFORMER:	PHP / MySql
PURPOSE:	To report to the user all documents that are not a child of another document. (ie, all documents that are not referenced by a webpage)
ENABLING COND:	A request from the user.
DESCRIPTION:	This task queries the database for all documents where that are not part of the "references" relation. The user will then be given the opportunity to delete all or some of these documents from the database.
FREQUENCY:	Whenever the user requests
DURATION:	Varies, Depends on number of documents in the database.
IMPORTANCE:	Important
INPUT:	N/A
OUTPUT:	The results of the search query formatted in a table ordered by relevance to search query. The output is returned via ajax
SUBTASKS:	Deletion Task
ERROR COND:	Invalid parameters from the user.

Reach Report Task

TASK NAME:	Reach Report Task
PERFORMER:	PHP / MySql
PURPOSE:	To report to the user all documents that are a child of the given document. (ie, all documents that are referenced the file given by the user)
ENABLING COND:	A request from the user.
DESCRIPTION:	This task queries the database for all documents where that are a part of the “references” relation with the parent file uuid given by the user. The user will then be given the opportunity to delete all or some of these documents from the database.
FREQUENCY:	Whenever the user requests
DURATION:	Varies, Depends on number of documents in the database.
IMPORTANCE:	Important
INPUT:	UUID taken from results of Query Document Task as requested by the user.
OUTPUT:	The results of the search query formatted in a table ordered by relevance to search query. The output is returned via ajax
SUBTASKS:	Deletion Task
ERROR COND:	Invalid parameters from the user.

Phase II

Purpose of Phase II

The purpose of Phase II of this project is to take the information, planning, and research done in phase I into a more concrete development stage. In phase II, we will convert our ideas in phase I, into an ER diagram. This diagram will show the different entities that will be in our database and the relationships between these entities.

After we have established our ER diagram, we will develop the relation schema based off of this diagram and define keys. This relation scheme will be in 3rd Normal Form. From here we will define the attributes we will be using for each of the document types that the database system will support as well as the necessary attributes needed in relationship tables.

After that, we will define more concretely, the tasks that will be run and particular details about these tasks. We will define the pseudo code for these tasks, which will include pseudo sql queries (not the actual queries) when we need to insert, update, and delete to the database.

Problems Encountered

- 1. Problem:** Not all documents contain a modification date. Therefore we can't always update a file.

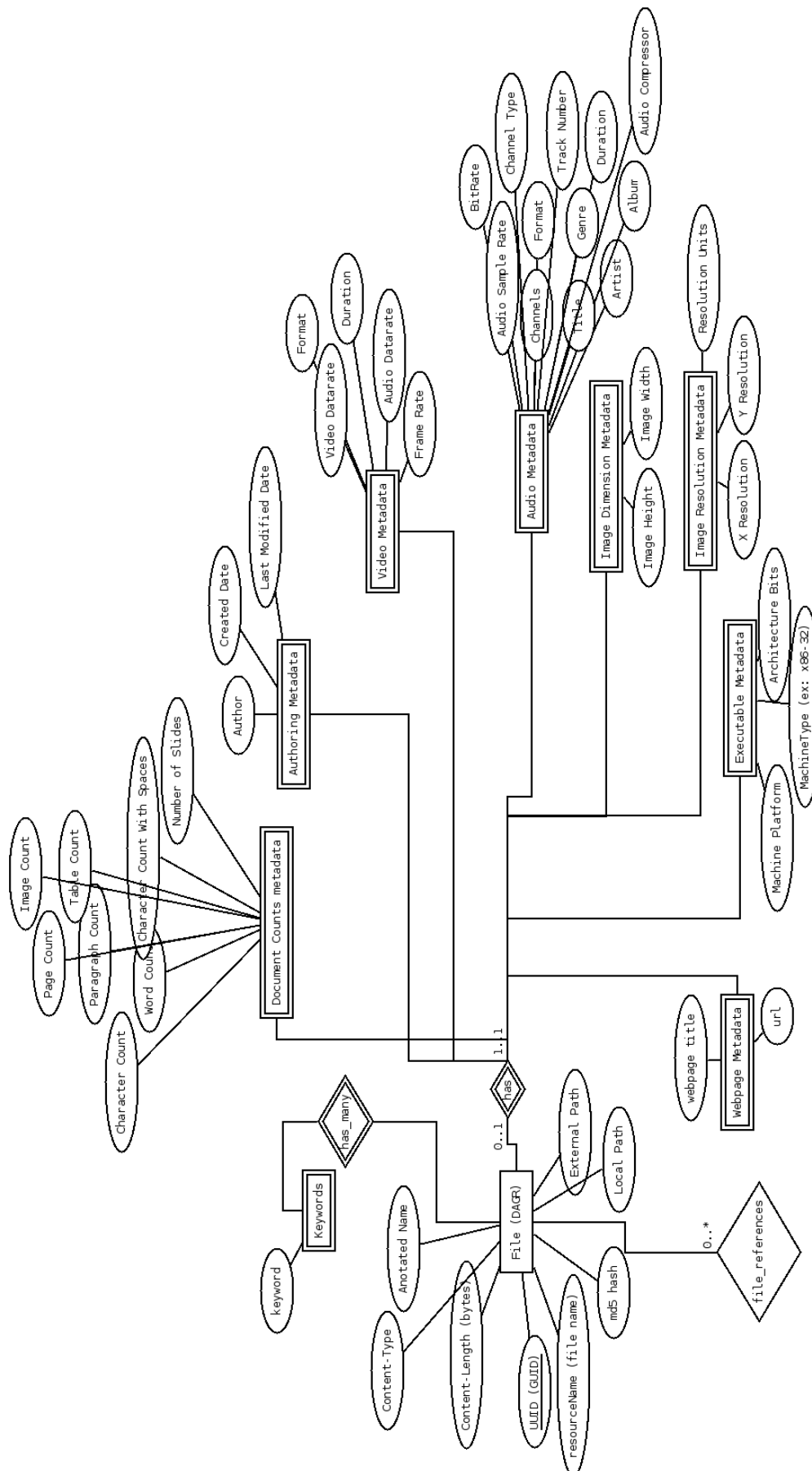
Solution: Because some files do not contain a modification date. We can always update a file. Therefore, instead we will be ignoring this functionality. When a document is being inserted into the database it will check to see if the file name already exists. If it does we will then check the hash of the file and see if it matches an existing hash. If the hash does exist, we will not bother to add the file as it has already been added. (We can create this hash with its md5).
- 2. Problem:** How to structure our file documents in the database.

Solution: We decided to structure the files among several different types (OfficeDocument, Video, Audio, Picture, Binary, and HTML). This hierarchy structure allows us to store file types that have some similar attributes in the same relation. For example, an OfficeDocument will have a common attribute of "word count" for Word, Presentation, Spreadsheet, and PDF file types.
- 3. Problem:** Providing local computer file directory path as an attribute in the database.

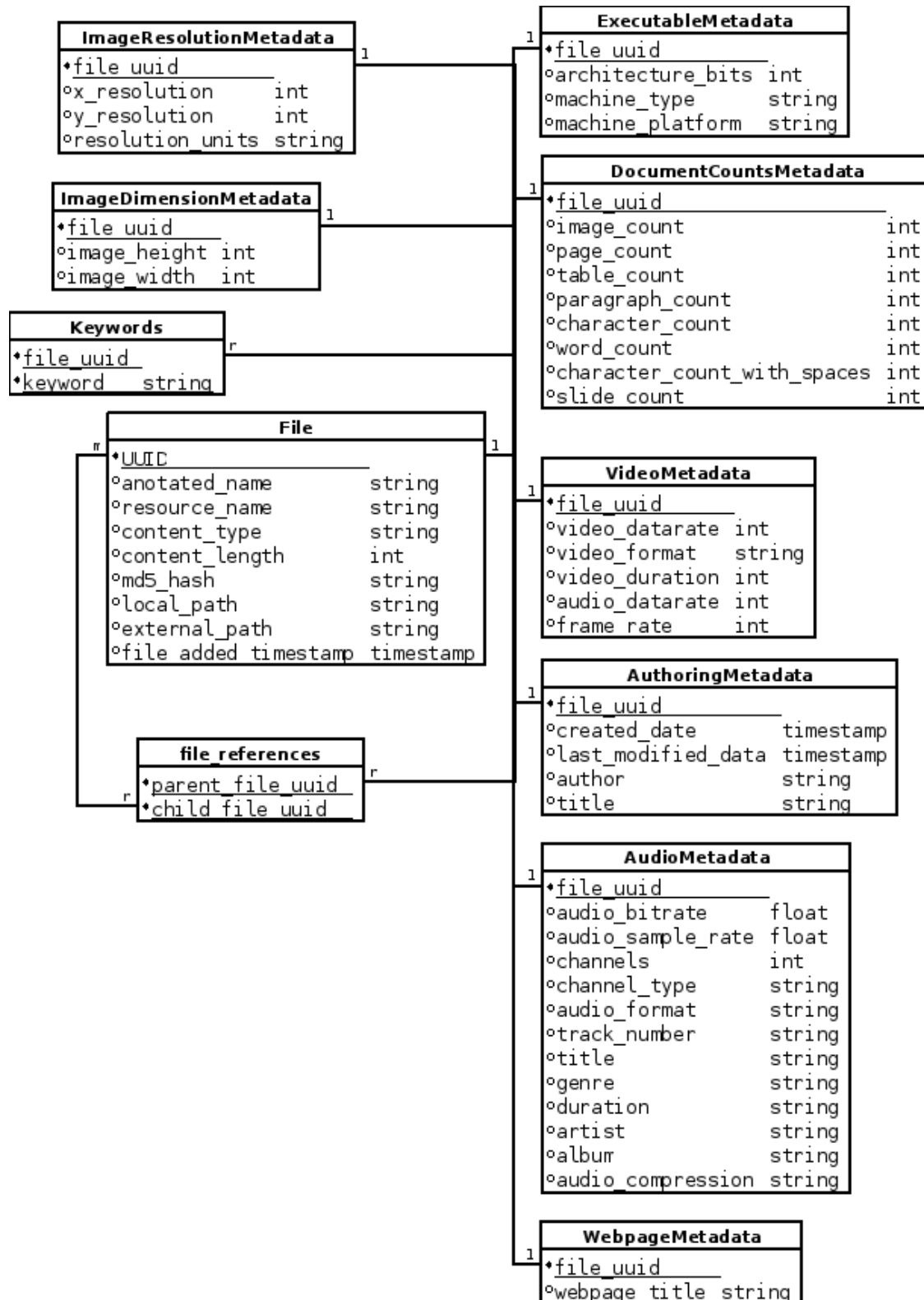
Solution: We are not able to make this feature due to security features of all web browsers. Therefore we will not be able to make files that were inserted from the users computer clickable unless we download and store the file.
- 4. Problem:** Each file type will have different metadata associated to it. And will effect the results of the queries.

Solution: We will use helper function in our database abstraction layer help return applicable metadata fields.

ER Model



Relational Schema



Pseudo Code¹

Insert File Task

```
InsertFile(file) {  
    1. md5Hash = md5Sum(file); // calculate hash using md5Sum function  
    2. CommonFiles = SELECT * FROM Files WHERE FileName = 'file' ;  
    3. For each CommonFile Found // if any  
        a. If md5Hash == CommonFiles[i].md5Hash  
            i. RETURN // don't bother to insert if metadata already exists  
    4. UUID = generateUUID();  
    5. attributes = ApacheTika(file);  
    6. anotatedName = (name given by the user using HTML form)  
    7. keywords = (keywords given by the user using HTML form)  
    8. INSERT INTO Files VALUES (UUID, file,anotatedName, attributes.FileSize,  
        attributes.FileType)  
    9. for each keyword:  
        a. INSERT INTO Keywords VALUES (UUID, keyword)  
    10. switch attributes.FileType:  
        a. INSERT INTO (dependent upon file type) VALUES (dependent upon file type and  
            user annotation given)  
    11. RETURN UUID // used for other tasks  
}
```

Edit File Task

```
EditFile(UUID){  
    1. anotatedName = (current name/ updated name provided by the user using HTML form)  
    2. keywords = (current/updated keywords given by the user using HTML form)  
    3. UPDATE Files anotatedName VALUES (new/updated anotated name) where UUID =  
        file's UUID  
    4. DELETE from Keywords where UUID = file's UUID //we need to delete them all so we  
        can read them  
    5. for each keyword:  
        a. INSERT INTO Keywords VALUES (UUID, keyword)  
}
```

Bulk Add Task

```
BulkAdd(folder){  
    1. for each file in folder  
        a. InsertFile(file);  
}
```

¹ Applicable attributes will be calculated with helper functions in our program database abstraction layer.

Apache Tika

ApacheTike(file){

1. xml = Insert file location into the Apache Tika api.
2. Convert xml into a hash table of attributes that this database supports
3. Rename attributes to fit the naming schema of the database
4. RETURN this Hash table

}

Add Web Page Task

AddWebPage(url){

1. refUUIDs = HTMLParse(url);
2. pageUUID = InsertFile(url);
3. for each refUUIDs
 - a. INSERT INTO PageReference VALUES (pageUUID, refUUID[i]);

}

HTML Link Parser

HTMLParse(url){

1. toInsert = list of source URL for all <a> and tags.
2. for each toInsert
 - a. InsertFile(toInsert[i])
3. RETURN list of UUID's returned from InsertFile

}

Deletion Task²

delete(uuid) {

1. DELETE * FROM Files WHERE UUID = 'uuid';
2. // We will be creating our weak entities with "ON DELETE CASCADE" and therefore all applicable metadata will be deleted during step 1.
3. Display success message

}

Query Documents Task

queryDocs(){

1. Take all values from Query HTML Form.
2. use these values to construct filters (the where statement of the query which returns all results.
3. SELECT (applicable attributes)
FROM (join statement for all metadata)
WHERE (filter constructed from step 2);
4. Display Results in Query Documents Table

}

² The deletion task can be run via a link from any of the result pages.

Time Report Task

```
TimeReport(){  
    1. Take start and end data values from HTML Form  
    2. SELECT (applicable attributes)  
        FROM (join statement for all metadata)  
        WHERE Files.date >= startdate AND Files.date <= enddate;  
    3. Display results in Time Report Page  
}
```

Orphan Report Task

```
OrphanReport(){  
    1. SELECT (applicable attributes)  
        FROM (join statement for all metadata)  
        WHERE Files.UUID IN (SELECT refUUID FROM WebPageReferences)  
    2. Display results in Result Report Page  
}
```

Reach Report Task

```
ReachReport(UUID from Query Document Task){  
    1. SELECT (applicable attributes)  
        FROM (join statement for all metadata)  
        WHERE File.uuid IN (  
            a. SELECT child_file_uuid  
                FROM file_references  
                WHERE file_reference.parent_file_uuid = (UUID from Query Document Task))  
    2. Display results in Query Documents Page  
}
```

Phase III

Introduction to Phase III

In phase III, we will finally put all of our planning into action. From our pseudo code and database diagram, we will finally develop our database, graphical interface and the underlying code that will allow the user to interact with the database.

The database will first be created using a MySQL database. Afterwards, a basic user interface will be designed and implemented. This interface will be very basic and simple enough to show the basic functionalities of our database system. Finally, we will enable our interface to work with the database by using php scripts to take the form data and run query processes to the database, returning results of success or failure.

Completed source code can be viewed and downloaded at
<https://github.com/chrisnyder2337/mmda>

Problems Encountered

1. **Problem:** Creating an interface for a user to query the database can be difficult to create. The amount of control we should give the user in querying is unclear from the project description.
Solution: We decided to provide the user with a single textbox that will be the “where” statement of the query. This solution is simple enough to implement and allows the user full control of what they would like to query but still allows the user not to worry about how to join the tables together.

Revisions to the Relation Schema

There was not much of a change to the relational schema. The only change was the addition of “title” and “author” attributes to the AuthoringMetadata relation. This was added due to the realization of how many different types of files have these attributes in their metadata.

We also changed the “file_references” relationship entity in the ER diagram. This change allows us to develop parent and child relationships with any type of DAGR and is not limited to web page parse results.

User's Manual

Adding a File's Metadata to the Database

1. Browse for the file(s) on your local machine that you would like to add to the database.



The screenshot shows the 'Mulimedia Data Aggregator' application with a dark navigation bar containing links: 'Add File', 'Add Webpage', 'Keywords', 'Query Metadata', and 'Reports'. The main content area is titled 'Insert New File'. It features a 'Select File(s)' section with a 'Choose Files' button and the filename 'hedgieBirthday.png'. Below this, a note states: 'To bulk add use Ctrl/Shift to select many files.' A blue 'Insert File Metadata' button is positioned below the note. At the bottom of the page, a footer line reads: 'Project for CMSC424: Database Design at the University of Maryland. Created by Christopher Snyder and Jon Snyder. © 2013'.

2. After insertion, a confirmation message will show.

This screenshot shows the same application after a successful insertion. A green confirmation banner at the top of the main content area displays the message: 'hedgieBirthday.png's DAGR was added.' To the right of the message is a blue 'EDIT' button. The navigation bar remains the same.

3. We can then view and edit the DAGR. At this point we can add keywords, an annotated name, and assign a parent DAGR.

The screenshot displays the 'Edit DAGR' form for a specific entry (UUID: fb3c6ca3-6149-11e3-af39-b499baf5b6ef). The form includes three input fields: 'Annotated Name' with the value 'Hedgehog Birthday', 'Parent DAGR' with a dropdown menu currently showing '--- NONE ---', and 'Keywords' with the value 'birthday, hedgehog, mypics'. Below the keywords field, a note indicates 'Comma Seperated' (sic). At the bottom of the form are two buttons: 'Delete DAGR' and 'Save Changes'.

 hedgieBirthday.png (fb3c6ca3-6149-11e3-af39-b499baf5b6ef)	 Keywords
uuid: fb3c6ca3-6149-11e3-af39-b499baf5b6ef	Parent DAGR
Filename: hedgieBirthday.png	- NO PARENT -
Content Type: image/png	Children DAGRs
File Size: 82102	- NO CHILDREN -
md5 Hash: 14c0bd6c3e1c89cb161c2b1b09516ab8	Descendant DAGRs (Reach Query)
Local Path: ./dagfiles/hedgieBirthday.png	- NO DESCENDANTS -
file_added_timestamp: 2013-12-09 22:20:12	
X Axis Resolution: 315	
Y Axis Resolution: 315	
Resolution Units: PixelInterleaved	

Adding a Webpage's Metadata to the Database

1. Either enter a website url manually or use the chrome extension to add your current web page URL to the textbox shown in the "Insert New Web page" page.

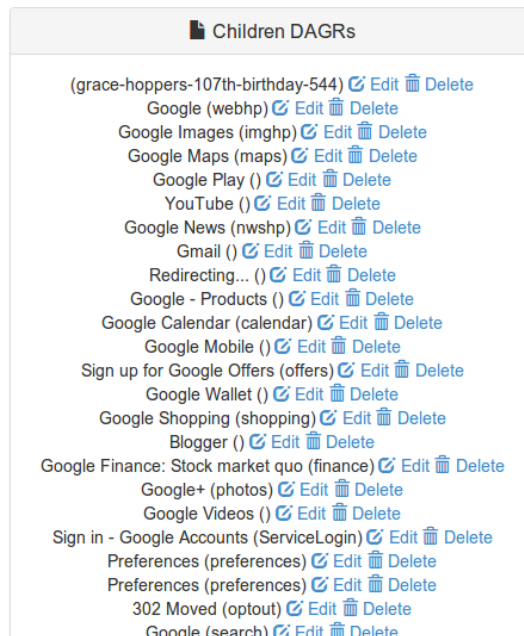
Mulimedia Data Aggregator
Add File
Add Webpage
Keywords
Query Metadata
Reports

Insert New Webpage

Submit

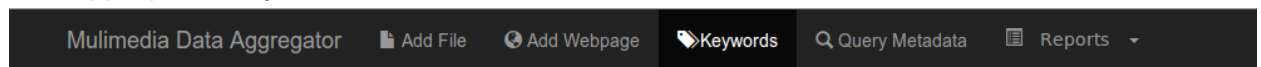
2. You will get the same type of screen as adding a regular file. However, your DAGR will automatically be populated with children which are linked files parsed from the website.

Such as images and linked hypertext links.



Find DAGRs by keyword

1. The user can easily find DAGRs by keyword by going to the keywords page and selected their appropriate keyword.



Keywords

exams
funny
school
text

Performing Custom Queries

1. The user can perform custom queries using the open ended query form. The user is supplied with the ability to perform any type of where clause. Below the form, the user is

provided with available attributes that they may use as they please.

Mulimedia Data Aggregator

Add File

Add Webpage

Keywords

Query Metadata

Reports

Query Metadata

Where Clause

File.annotated_name = 'hat'

Provide SQL Conditions. Example: File.annotated_name = "hat"

Query

Query Help

Show / Hide

File
Main file metadata.
<code>File.annotated_name</code> User annotated Name
<code>File.resource_name</code> File name of document
<code>File.content_type</code> ex: html, pdf, etc Recommend use <code>LIKE</code> for these.

Authoring
Generic file metadata involving author and date.
<code>AuthoringMetadata.created_date</code>
<code>AuthoringMetadata.last_modified_date</code>
<code>AuthoringMetadata.author</code>
<code>AuthoringMetadata.title</code>

Image
Demension metadata of image files.
<code>ImageResolutionMetadata.x_resolution</code>
<code>ImageResolutionMetadata.y_resolution</code>
<code>ImageResolutionMetadata.resolution_unit</code>

Audio

Reporting

- There are three main types of reports the user may run.
 - The Orphan Report displays all of the DAGRs which have now parent DAGR
 - The Serile Report displays all DAGRs which have no children
 - The Time Report shows all DAGRs which were added within the user's specified time frame.

Mulimedia Data Aggregator
Add File
Add Webpage
Keywords
Query Metadata
Reports
Orphan Report
Sterile Report
Time Report

Orphan Report

These are all of the DAGRs which have no Parent DAGRs.

uuid	Annotated Name	Filename	Content Type	File Size	md5 Hash	Local Path	External UF
31dc33e8-6154-11e3-af39-b499baf5b6ef Edit Delete	Google		text/html; charset=ISO-8859-1		0efaeeb02c5b55560a0667731da93dfd		https://www.
5d35d4c8-6150-11e3-af39-b499baf5b6ef Edit Delete		Lec 21 - Triple Integrals.xoj	application/x-gzip	314934	5f9d76d4ce919474d6190d483a2416d6	./dagrfiles/Lec 21 - Triple Integrals.xoj	
bbea3d2d-614c-11e3-af39-b499baf5b6ef Edit Delete		Add Gift Card App to facebook.pdf	application/pdf	256217	7076ff8aa5ddf22246578fcb75f59483	./dagrfiles/Add Gift Card App to facebook.pdf	
df90ba9d-	15 7/ID219) nnt	annlication/vnd ms-	121856	42237683057cf7320d9651981aa0d233	./danrfiles/15 7/ID219) nnt		

Future Considerations/Performance

Adding Large Webpages

When adding webpages that have a lot of large file references it takes a long time to scan the metadata of each file. If we were to create a program that wanted to parse such large webpages, we may have to create limits on the amount of metadata it can parse.

SQL Injection Concerns

Providing a user with a open ended query field is not advantages because it leaves the possibility of SQL injection attacks. Because, this is just a demonstration of the operation of the MMDA database system and not a practical application, forgoing this precaution is alright. In a more practical application, the serperation between the database and the end user would be more abstract and secure.