

# **MMDA: A Multi-Media Data Aggregator**

## **Database Design Project: Phase 3**

Austin Piel and Frank Tiburzi



Please view the full project at:  
<https://github.com/franktiburzi/MMDA-Project>

## **Table of Contents**

<b>1. Environment and Requirement Analysis</b>	<b>3</b>
<b>2. System Analysis and Specification</b>	<b>6</b>
<b>3. Conceptual Modeling</b>	<b>20</b>
<b>4. Description of the System</b>	<b>27</b>
<b>5. System Limitations and Technical Assumptions</b>	<b>30</b>
<b>6. User Manual</b>	<b>32</b>
<b>7. Testing and Error Handling</b>	<b>37</b>
<b>8. Conclusion</b>	<b>38</b>

# 1. Environment and Requirement Analysis

## 1.1 Purpose of this Document

This document will serve as a guide and manual to our CMSC 424 final project - a Multi-Media Data Aggregator, which we will refer to as MMDA. The document will contain implementation information of both the physical database and tables as well as the front and back-end code that allow the system to function with a web interface.

The document will also serve a record of assumptions and choices we made when designing the database system. Many task and specifications of the project could be interpreted and implemented in different ways and this document will provide a clear explanation of what occurs in our system as well as reasoning for our choices.

The document will be written with a target audience of experienced developers and database administrators and may gloss over small details that we expect readers to be familiar with.

## 1.2 Purpose of the project

The purpose of the project was to create a Multi-Media Data Aggregator. The fundamental unit of the MMDA is a Data Aggregate(DAGR). A DAGR is a grouping of an arbitrary number and type of files that we wish to store in our system. An MMDA can be compared to a file system found on a computer, where a DAGR can be compared to folders in that file system, and just like a folder a DAGR can hold one more files of any type. A DAGR in the MMDA however, does not store any physical files, but rather metadata about the DAGR, which we will discuss in more detail later.

Another primary concern of the MMDA is hierarchical relationships and groupings. The MMDA allows users to group DAGRS together in a category or to establish sub DAGRs in an already existing DAGR.

One of the most important purposes of the project was to create an easy to use web interface that allows users to perform a multitude of tasks. Through the webpage, a user is able to insert DAGRs with a chosen name and short description, search for specific DAGRs and many other task we will describe later.

### 1.3 Scope

This project is split into three different phases, each with its unique purpose. The first phase involves identifying system requirements, an overview of the system, possible technical/conceptual problems, the tasks required for each part of system operation, and assumptions about the user and how they will use the system. This phase also involves setting up a web server that will host the database and the website.

The second phase involves conceptual modeling and task emulation. In this report, we included a description of problems encountered along with justification for the solutions. In this phase we also modeled the system using entity relationship and relational diagrams and found the functional dependencies. We also set up the MySQL database in this phase.

The third and final phase of the project involved fixing any issues with the database model and fully implementing the system and all the tasks. We describe the assumptions and limitations of the system and possible future enhancements. This phase, and the project as a whole end with a presentation of the system.

### 1.4 General Assumptions

These are general assumptions we have made about the project and the system. Specific technical assumptions of the implementation will be described in detail later in the report, where as these apply to the system as a whole.

- The user has internet connection and can access the website via a web browser
- The user is able to understand English well enough to understand and operate the website
- The database software is up to date
- The database will be able to handle the number of users accessing it at one time
- The web server will be able to handle the number of users accessing it at one time
- There will be enough storage space on the server to store all of the MMDA information

### 1.5 Limitations and Solutions

Below are some of the issues we encountered when developing the MMDA. These are high level issues and the general solution we found for them rather than small technical issues such as errors in code

**Problem:** Time constraints - We are developing a large scale system in just a few months alongside other coursework.

**Solution:** Break the project up into separate phases with small task that can be accomplished one by one to stay on track, and have regular group meetings.

**Problem:** Choosing an appropriate web development stack - We need to decide on a database system and scripting language we want to write our backend in that will enable us to work effectively and efficiently.

**Solution:** We have decided on using XAMPP to host a local server with a MySQL database and PHP. PHP has many built in methods to gather metadata and for parsing HTML and hooks into MySQL very easily.

**Problem:** Hierarchical data organization - Many aspects of the MMDA require knowing parent and child DAGRs making this problem central to our design. We needed to come up with a way to easily and efficiently store and search this.

**Solution:** We will use a separate table within the MMDA just to keep track of parent-child relationships, rather than trying to store this information with the file information or metadata which could cause confusion.

**Problem:** Metadata extraction - Our MMDA supports a variety of documents from both local directories and webpages, which could complicate metadata gathering and be very difficult.

**Solution:** When possible we used the many built in PHP functions that can get certain types of metadata, and when writing our own functions we made them general enough to support many file types.

## 2. System Analysis and Specification

### 2.1 Description of Procedure

The MMDA database is hosted on a remote web server and is operated via web browser by the user. Users are able to perform various queries on the database, as well as insert, delete, and categorize documents from the database.

### 2.2 From the User's Perspective

First, users navigate to the domain from which the server can be accessed. The website acts as a front-end to the database and allows users to interact with it without accessing the data explicitly. By using queries, the user is able to send requests to the server and receive the requested information via web browser.

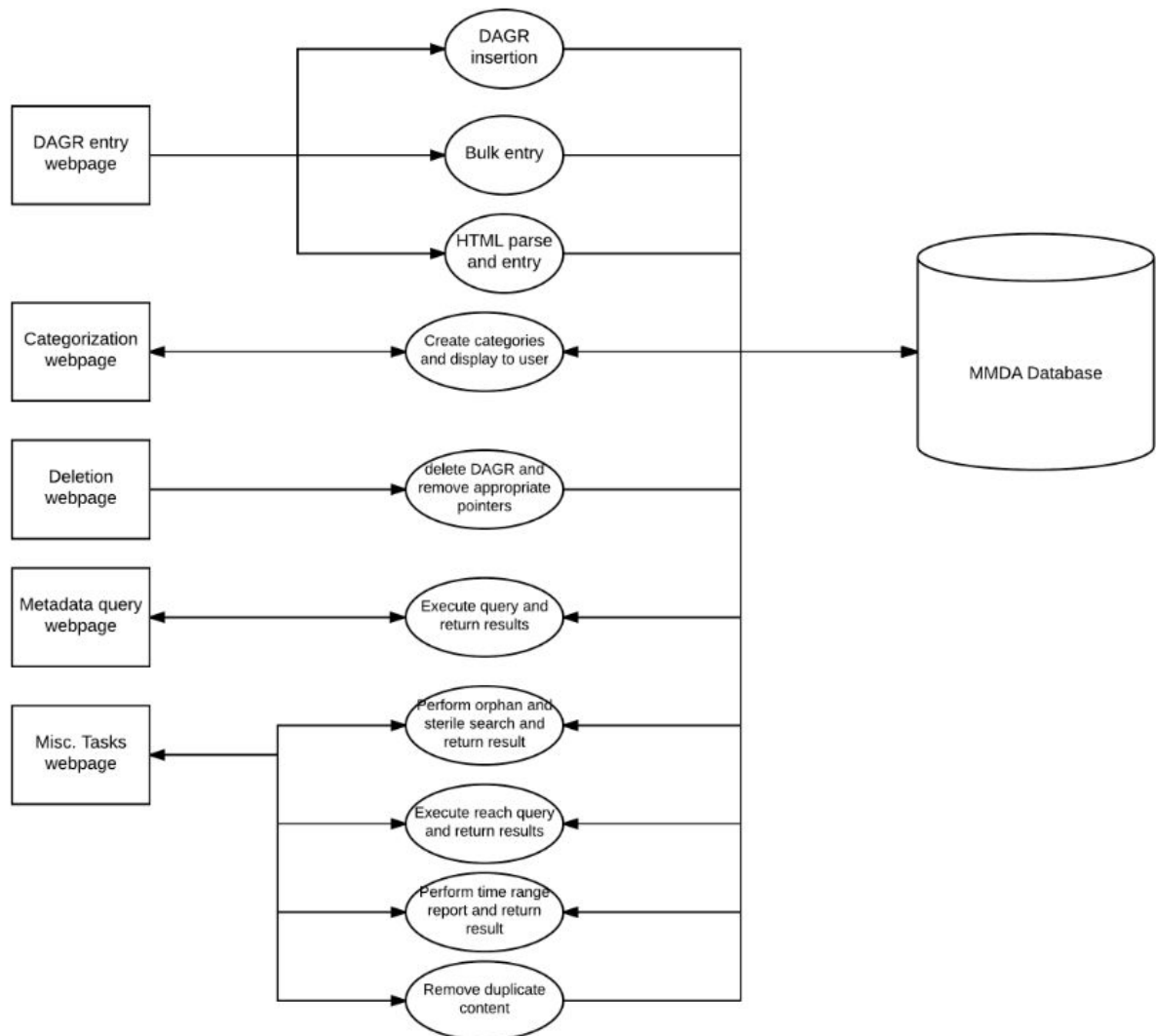
### 2.3 From the Developer's Perspective

The developers will use several components in order to complete the system:

- The XAMPP web server stack contains most of the services used by our project. This includes the Apache HTTP server and the MySQL database management system
- Some of the software we will be using to develop and test the system:
  - Facebook Messenger: This application allows us to communicate easily and efficiently about the project
  - GitHub: This will allow us to set up a repository allowing us to work independently and then merge our code. This also provides version control in the case of a system failure
  - Atom: This editor allows us to easily edit and develop code for the project
  - Google Chrome: The web browser that we will be using to test, view, and demonstrate our web application. We will also be developing a chrome browser extension

## 2.4 Information Flow Diagram

The information flow diagram shows the flow of information between the user and the MMDA database



## 2.5 Task Forms

Task Number	DIT
Task Name	DAGR Insertion Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Insert a new document into our database, and automatically assign an identification number and have its metadata recorded
Enabling Condition	Database allows new entries and associates its metadata correctly and assigns identification number
Description	The user will be able to view a heirarchy of DAGRs and choose where to insert the new document. The system will handle organizing the documents and recording metadata of the document. The user can rename the entry without losing identifying information.
Frequency	The user can enter a new document whenever they choose to. This could be done very often or rarely done.
Duration	Short
Importance	Critical
Maximum Delay	short
Input	New document
Output	Updated database containing new document
Document Use	Any file type can be added
Operations Performed	Database updated
Subtasks	Store metadata, assign GUID
Error Condition	N/A

Task Number	BDET
-------------	------



Task Name	Bulk Data Entry Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Add multiple files to the database at once
Enabling Condition	Database allows new entries and associates its metadata correctly and assigns identification number
Description	Same as regular insertion but will take in many documents at once. The user will be able to view a hierarchy of DAGRs and choose where to insert the new documents. The system will handle organizing the documents and recording metadata of the document. The user can rename the entry without losing identifying information.
Frequency	Whenever a user bulk enters documents
Duration	Short
Importance	High
Maximum Delay	short
Input	New document
Output	Updated database containing new document
Document Use	Any file type can be added
Operations Performed	Database updated
Subtasks	Store metadata, assign GUID
Error Condition	N/A

Task Number	PADET
Task Name	HTML Parser and Automatic Data Entry Task
Performer	Austin Piel, Frank Tiburzi, HTML parser, MySQL

Purpose	Parse HTML documents and submit the contents into the database with appropriate metadata and identifying information
Enabling Condition	Functioning HTML parse and database properly accepts and stores new files
Description	Scan through an HTML file and extract different embedded files and external links. Obtain metadata about all files/links and automatically add them into the database while storing their relation to the HTML document they came from
Frequency	Whenever a user enters HTML documents
Duration	Short
Importance	High
Maximum Delay	short
Input	New document
Output	Updated database containing new document
Document Use	Any file type can be added
Operations Performed	Database updated
Subtasks	Store metadata, assign GUID, build HTML parser
Error Condition	N/A

Task Number	WBIT
Task Name	Web Browser Interface Task
Performer	Austin Piel, Frank Tiburzi, Text editor
Purpose	Create a simple but robust web interface for interacting with the database and allow users

	to upload an HTML page shown in browser
Enabling Condition	The database that the website will interact with is completed and functional
Description	This is a website that will allow users to visualize the database structure and search for files or metadata. It will also connect with a browser plugin allowing instant upload of HTML pages when browsing the web
Frequency	The user should be able to visit the website and it needs to be ready to display information
Duration	Long
Importance	Critical
Maximum Delay	Very short.
Input	1) The database contents 2) HTML pages displayed in browser
Output	1) A webpage displaying the information in a user friendly way 2) A updated database
Document Use	Database, HTML pages
Operations Performed	Generating webpage and database updates
Subtasks	Document entry, HTML parsing, webpage design
Error Condition	N/A

Task Number	SCT
Task Name	Support of Categorization Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Allow users to organize files into named groupings
Enabling Condition	Functioning database and web application
Description	Through the web interface, a user can view

	files and create their own categories to make viewing easier for them. This can include groupings within groupings and they can insert and delete freely in these
Frequency	Any time a user creates a grouping or inserts/deletes from one
Duration	short
Importance	high
Maximum Delay	Short. We do not want lagging applications on the web interface
Input	Database contents
Output	User defined grouping
Document Use	Web interface, database
Operations Performed	Update database relationships, recording that documents have a connection
Subtasks	Document insertion/deletion, web interface
Error Condition	N/A

Task Number	DDT
Task Name	Deletion of a DAGR Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Remove a file record from the database, and delete subfiles if desired by user
Enabling Condition	Functional database and relationships defining parent and children files
Description	Deleting a DAGR will not only delete a DAGR and its metadata from the database, but will remove references to it from other DAGRs. There will be an option to delete or keep children files aswell

Frequency	Whenever a user deletes a DAGR
Duration	short
Importance	high
Maximum Delay	short
Input	A DAGR identifying name to be deleted
Output	The updated database with removed DAGR and references
Document Use	Web interface, database
Operations Performed	Update database
Subtasks	N/A
Error Condition	N/A

Task Number	DMQT
Task Name	DAGR metadata query Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Allow users to search for a DAGR by its metadata
Enabling Condition	Appropriate storage and relationship of metadata to DAGR
Description	The user will be able to look up a document by searching for its metadata attributes
Frequency	Whenever a user performs a metadata search
Duration	medium

Importance	high
Maximum Delay	short
Input	User entered metadata information
Output	DAGR associated with provided metadata
Document Use	Web interface search tool, database
Operations Performed	Return some content from database
Subtasks	DAGR metadata storage
Error Condition	N/A

Task Number	OSRT
Task Name	Orphan and Sterile Reports Task
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Return DAGRs without any references to or from them
Enabling Condition	Functional database with hierarchical relationships
Description	Presents the user with a list of DAGRs that have no references to them, and are a parent to no other DAGRs.
Frequency	Whenever a user requests the report
Duration	short
Importance	medium

Maximum Delay	short
Input	User input specifying a request for orphan and sterile report
Output	List of DAGRs
Document Use	Web interface, database
Operations Performed	Search relationships in database and return contents
Subtasks	Reach Queries task
Error Condition	N/A

Task Number	RQT
Task Name	Reach Queries Task
Performer	Austin Piel, Frank Tiburzi, MySQL, Search algorithm
Purpose	Return DAGRs that are referenced by or referenced from a user specified DAGR
Enabling Condition	Functional database with hierarchical relationships
Description	When the user enters a DAGR, it will return a list of DAGRs that reference it, and all the DAGRs that it points to. The user can specify the depth/height they wish to search, but the default is one level
Frequency	Whenever a user submits a query

Duration	medium
Importance	high
Maximum Delay	short
Input	User specified DAGR and optional depth
Output	List of referencing and referenced DAGRs
Document Use	Web interface page to select DAGR, database
Operations Performed	Search relationships in database and return contents
Subtasks	n/a
Error Condition	N/A

Task Number	TRDR
Task Name	Time Range DAGR report
Performer	Austin Piel, Frank Tiburzi, MySQL
Purpose	Return DAGRs associated with a date given by the user
Enabling Condition	Database with metadata including document create date and DAGR entry date
Description	The user will enter either a date, or a range of dates on the web application
Frequency	Whenever a user requests a time range report
Duration	short
Importance	medium
Maximum Delay	short



Input	User specified date or date range
Output	List of DAGRs submitted on that date/range
Document Use	Web interface page to enter dates, database
Operations Performed	Search database for corresponding dates
Subtasks	N/A
Error Condition	N/A

Task Number	IDCT
Task Name	Identify Duplicate Content Task
Performer	Austin Piel, Frank Tiburzi, MySQL, Web Server
Purpose	Remove duplicate documents with different GUIDs
Enabling Condition	Identification of database content in alternate ways from GUID to know if it is replicated
Description	In some cases, the same document could be in the database more than once, but with a different GUID, giving the illusion it is two different documents. This will remove the duplicates
Frequency	After any data entry this will be performed automatically
Duration	short
Importance	medium
Maximum Delay	medium

Input	A new document entry. This will happen indirectly in the background so no direct input
Output	An updated database with duplicates removed
Document Use	database
Operations Performed	Search database and delete duplicate records
Subtasks	Deletion of DAGRs
Error Condition	N/A

Task Number	MDT
Task Name	Modify a DAGR Task (optional)
Performer	Austin Piel, Frank Tiburzi, MySQL, HTML editor
Purpose	Update contents of a DAGR
Enabling Condition	Ability to insert and delete for local files, HTML editor for HTML documents
Description	If the user wants to change details of a DAGR through the web interface they can perform this function on both local files and HTML documents
Frequency	Whenever a user modifies a DAGR
Duration	medium
Importance	medium
Maximum Delay	short
Input	DAGR the user wishes to modify and their specified modifications
Output	Database containing updated DAGR
Document Use	Database, HTML editor

Operations Performed	Update any document type, and update the database with changes
Subtasks	HTML editing
Error Condition	N/A

## 3. Conceptual Modeling

### 3.1 Entity Relationship Diagram

Below is our E.R. diagram of our full MMDA database. One thing to note is the inheritance from the generalized entity “file”. This entity is not a table in our database but used to simplify the diagram, and all of its attributes are present on the entities below it.

Some attributes could be represented in various data forms. For these, carefully chose the best form to best be stored in the database and are described below.

**Date Entered:** Unix timestamp(INT)

**Date Created:** Unix timestamp(INT)

**Filesize:** Bytes(INT)

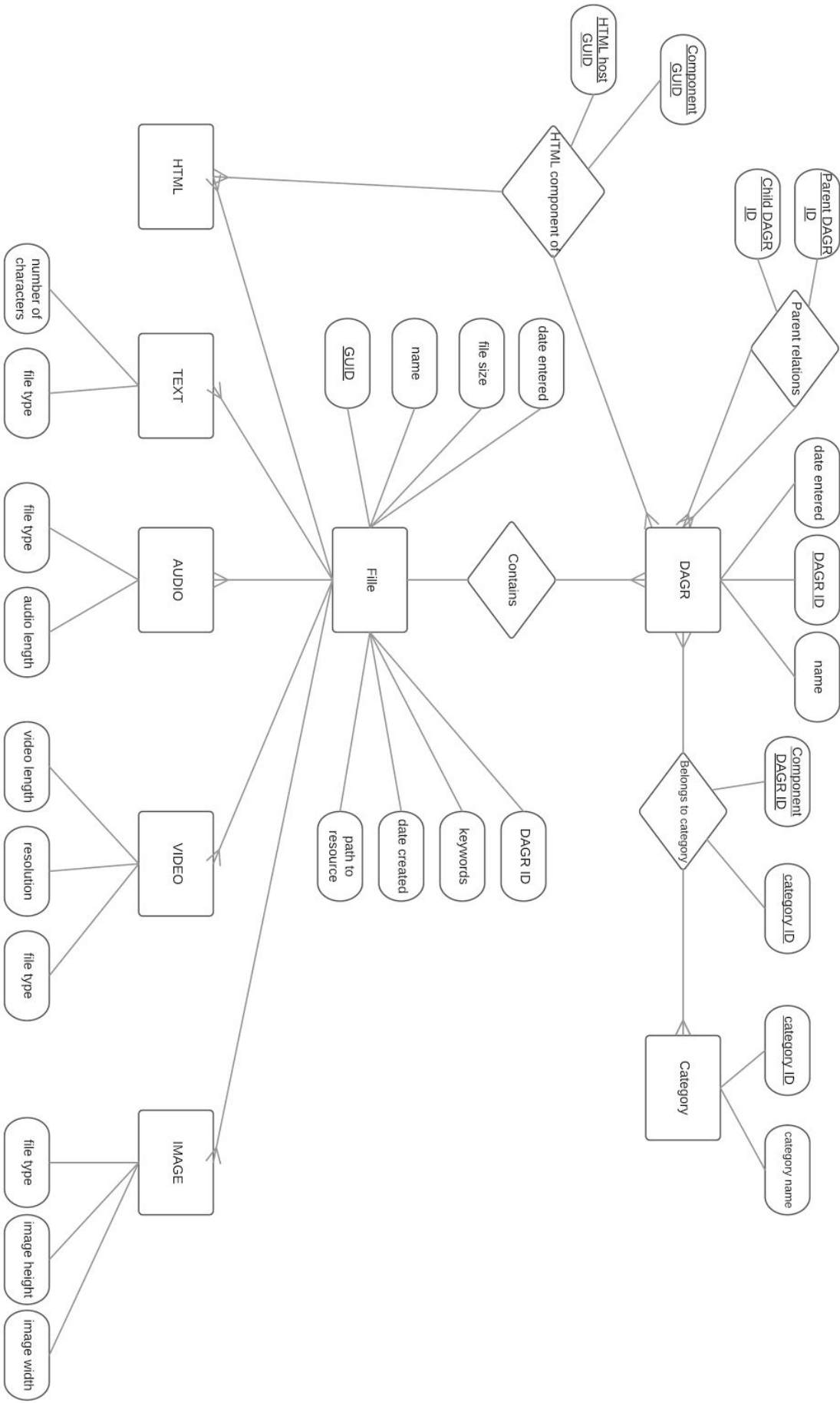
**Audio Length:** Seconds(INT)

**Video Length:** Seconds(INT)

**Resolution:** “width X height”(VARCHAR)

**Image Height:** Pixels(INT)

**Image Width:** Pixels(INT)



### 3.2 Functional Dependencies

These are the functional dependencies for the entities in our MMDA database. The primary key of each entity is underlined

For the HTML filetype entity:

- GUID → name, DAGR ID, filesize, keywords, date created, date entered, path to resource

For the TEXT filetype entity:

- GUID → name, DAGR ID, filesize, keywords, date created, date entered, path to resource, filetype, number of characters

For the IMAGE filetype entity:

- GUID → name, DAGR ID, filesize, keywords, date created, date entered, path to resource, filetype, image width, image height

For the VIDEO filetype entity:

- GUID → name, DAGR ID, filesize, keywords, date created, date entered, path to resource, filetype, video length, resolution

For the AUDIO filetype entity:

- GUID → name, DAGR ID, filesize, keywords, date created, date entered, path to resource, filetype, audio length

For the Category entity:

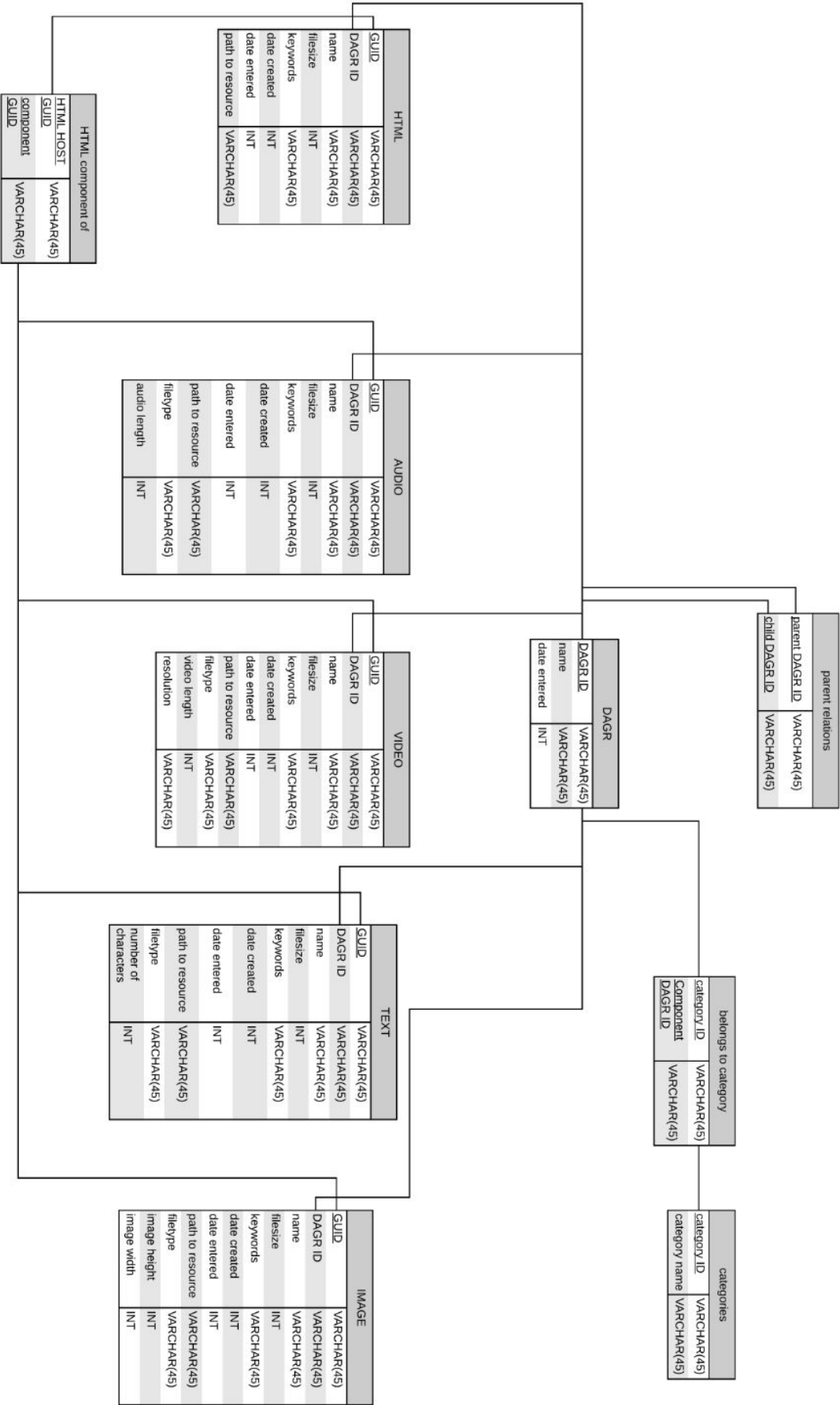
- Category ID → category name

For the DAGR entity:

- DAGR ID → date entered, name

### 3.3 Relational Schema

The relational schema shows how the tables are connected to each other and is derived from the E.R. diagram. All of the entities are turned into tables and the relationships that require attributes in the E.R. diagram are also turned into tables.



### 3.5 Normalization

It was important to normalize our database to increase consistency by reducing redundancy. It is for this reason that we made every attribute in our tables depend on the primary key, and only the primary key. In the four data type entities, the primary key is the GUID and every attribute is dependent on the GUID value and nothing else.

In category entity, the primary key is the category ID, and the only other attribute is solely dependent on this key.

Lastly in the DAGR entity, both time entered and name depend on the primary key, DAGR ID and nothing else.

This lack of dependencies on anything other than the primary key is what makes our MMDA database in BCNF

### 3.6 Task Emulation and Pseudocode

Below is a generalized procedure of how we accomplished each task. We wrote out the pseudocode in english rather than DML code as the tasks are implemented much too complex in our system to print them here.

#### DAGR Insertion Task

```
Identify file type of the new DAGR
    Extract metadata attributes based on file type
    Assign GUID to DAGR
    Record user-designated name and keywords if provided
    Store DAGR record in matching file type relation
```

#### Bulk Data Entry Task

```
Let user input a group of documents
    For each document
        Apply DAGR Insertion Task
```

#### HTML Parser and Automatic Data Entry Task

```
Extract metadata from HTML page
    Store information in the HTML table
    For each document/link in the HTML page
        Identify file type and extract metadata
        Parse HTML document to extract keywords
        Store information in appropriate file type table
```



### **Web Browser Interface Task**

Inside of browser, display a file system showing DAGRs currently in the database.

From a browser plugin be able to choose a DAGR to add to.

When viewing a HTML page, and selecting a DAGR to add to.

Perform HTML Parser and Automatic Data Entry Task

The input is the currently viewed page and the parent relation is recorded in the selected DAGR

### **Support of Categorization Task**

User creates a category

User selects DAGRS they wish to add to this category

For each DAGR selected for category

Create an entry in the categorization table with column 1 as category name and column 2 as DAGR GUID

(this table is not currently in the relation schema, will add on next revision)

### **Deletion of a DAGR Task**

User selects DAGR to delete

Check parent DAGR table and return list of parent DAGRs affected and ask user if daughter DAGRs should be shallow or deep deleted

Delete the selected DAGR from its appropriate table and remove all references to its GUID from the parent DAGR table

Perform the selected deletion method on daughter DAGR

### **DAGR Metadata Query Task**

User enters search parameters

Turn parameters into query and execute on metadata columns of the tables in the database

Return results of query to user

### **Orphan and Sterile Reports Task**

When asked to generate orphan report

Select all distinct GUID from child column in parent DAGR table

Return all GUID in the database not present on the selection generated in the previous step

When asked to generate sterile report

Select all distinct GUID from parent column in parent DAGR table

Return all GUID in the database not present on the selection generated in the previous step

### **Reach Queries Task**

Accept user defined search height and starting DAGR

Perform breadth first search on parent DAGR table to desired height

Return DAGR GUID of all DAGRs that can be reached from starting point

### **Time-Range DAGR Report**

User selects a date range and option for DAGRs entered in the range or files created in that range

Query the date-entered or date-created columns of the tables

Return all DAGRs that satisfy date range

### **Identify Duplicate Content Task**

Duplicate content can be identified by its identical metadata

Select DAGRs with identical metadata attributes(filesize,type,path to resource etc.)

Delete all but the first occurrence of the DAGR

Update all of reference with the GUID of the first occurrence

(the decision to use the first occurrence is arbitrary and has no

Impact on the data, it is chosen for simplicity)

## 4. Description of the System

### 4.1 MMDA Accepted File Types

The MMDA is primarily a database concerned with storing metadata. When faced with the task of creating a system to hold metadata, we had to make a choice between creating a general system that stores very limited metadata for a huge variety of files, or a specialized system that stores lots of metadata for a subset of file types.

We designed a specialized system to store extra metadata about a smaller group of file types. This required much more work than storing general metadata as it required knowing the file type and writing functions to get file type specific metadata.

We believed that this was the best implementation based on the assumption that a normal user most often interacts with a core subset of files across a range of media types. For example, a user more often want to know the length of an audio file stored in the MMDA, such as an mp3 song, than they will want to insert an obscure software binary into the database.

For this reason we decided to offer extensive metadata support for 11 of the most common file types across four genres of media: text, audio, images, and video. Below is a list of file types supported by our MMDA and the metadata stored for each.

1. **.txt** - date created, date entered, file size, path to resource, name, keywords, number of characters
2. **.docx** - date created, date entered, file size, path to resource, name, keywords, number of characters
3. **.xml** - date created, date entered, file size, path to resource, name, keywords, number of characters
4. **.png** - date created, date entered, file size, path to resource, name, keywords, image width, image height
5. **.jpg** - date created, date entered, file size, path to resource, name, keywords, image width, image height
6. **.gif** - date created, date entered, file size, path to resource, name, keywords, image width, image height
7. **.wav** - date created, date entered, file size, path to resource, name, keywords, audio length
8. **.mp3** - date created, date entered, file size, path to resource, name, keywords, audio length

9. **.mp4** - date created, date entered, file size, path to resource, name, keywords, video length, resolution
10. **.mov** - date created, date entered, file size, path to resource, name, keywords, video length, resolution
11. **.html** - date created, date entered, file size, path to resource, name, keywords  
\*html entries also store the links that are contained in them

## 4.2 The Database

The database can be broken down into two “types” of tables, those representing entities and those representing relationships between entities. All of the media types, text, audio, image, video, and html are stored in a table so that we can keep media specific metadata together.

When a user inserts a new DAGR, the file type is detected, metadata acquired, and it is given an GUID and the tuple is added to the appropriate table based on the media type.

The other tables in our database are used to store relations between DAGRs. These tables exist with two attributes, both of which are GUIDs of DAGRs and a tuple in these tables shows that there is a relationship between the two entries, with the relationship being specified by the specific table the tuple is in.

## 4.3 The Website

The website was designed to be simple with only the features that a user would need to increase performance and ease of finding things.

To organize the site we added a taskbar to the top of the page that groups together relevant tasks. For example, bulk insert and insert are found on the same page, but unrelated tasks like identify duplicate content would be found on another page, easily identifiable by the task bar.

Another important part of the website was to create a way to visualize the data in the MMDA. Although the database itself can keep track of complex hierarchical relationships, these are ultimately hidden from the user. Our website features several ways to look at DAGRs and DAGR relationships. There are webpages that allow a user to execute queries and search for metadata attributes, view and create categories to organize DAGRS, and view all the linked components of a html DAGR.

We believe that the website is very intuitive to use and many of the features are self explanatory. Additionally we have included short descriptions with each task on the webpage to help users find their way around.

#### **4.4 Browser Extension and Automatic html Parsing**

One of the more robust features of our MMDA system is the ability for a user to enter a html webpage, and have the system automatically parse the page, and insert all the links to the database, and then carry out this task again one more time on all the extracted links. This is a great way to populate the database as it spiders out very fast capturing files.

In order to make this task of entering html webpages into the database even easier for users, we created a chrome extension. When a user clicks on the extension it opens a page where they can enter the URL of the webpage instantly. It has all the same features of inserting a DAGR on the main website, including adding a name and keywords and inserting into existing DAGRs.

## 5. System Limitations and Technical Assumptions

### 5.1 Technical Assumptions/Decisions

Earlier in the document we touched on basic assumptions about the enterprise that are the bare minimum to use the system. Now we will talk about the technical assumptions we made when creating our database structure. These are not incorrect implementations or an explanation for not including something, but simply the way we chose to implement certain tasks in our database, that could have been implemented in a number of ways.

One decision that we made in our MMDA system was about the implementation of parent DAGRs and hierarchical data. In our system, when DAGRs are inserted into a previously existing DAGR, they become a child of all ancestor DAGRs. For example if we insert DAGR2 into DAGR1. Next we insert DAGR3 into DAGR2, creating a 3 DAGR deep system. In our system, DAGR3 is a child of DAGR2, as well as a child of DAGR1.

A similar effect occurs with html parsing. When our html parser crawls links two pages down, all found links become noted as an html component of the original html document we parsed first.

This choice still accurately reflects the parent structure of the MMDA database, but decreased complexity of many of the tasks by several orders of magnitude, creating a much more reliable system.

Our second major technical decision we made related to finding duplicate content in the database. We believed that duplicate content should only be removed when it is present in the same DAGR. While it could be considered redundant to have the same data stored in database twice, we believed that this was only truly an issue if it was present in the same DAGR. Since we made many analogies to a file system already, we believed this was similar to not being able to have the same file in a directory(DAGR) twice, but putting the same file in different directories(DAGRs) would be okay.

### 5.2 Limitations of the Implementation

Although we have added all the functionality required in the project, there are some areas that we believe hold back the implementation slightly.

One limitation we have identified, that does not affect general functionality but could be improved upon is lack of file system support on the website. It would be much nicer for a user to open a pop up on the website and select documents from their file system rather than explicitly declaring a filepath to the directory or document.

Finally, one small limitation that has no bearing on functionality would just be improving the style of the website a small bit. The website is 100% functional, and does use CSS to style it giving it a generally good look, but it is still rather plain with standard fonts. It is feature packed but we could perhaps manage the screen space better to make it more appealing to users, as well as implementing features of Bootstrap to make the site more interactive.

### **5.3 Future Improvements to the System**

Given more time to work on the MMDA database we believe we could add several more features to the site that might be appealing to users and increase its functionality. First in a future implementation we would introduce improvements to the limitations we identified in the above section.

Another improvement we would like to add to the system is support for more file types with detailed metadata. As we mentioned previously, we were very concerned with storing detailed metadata and as a result limited the accepted file types. The system however is very modular and adding support for new file types simply requires adding new methods to one file on the server and updating the webpage to show we support it.

Another improvement that I believe we could add to the system would be creating a more visual system for categorization. Currently users interact with their categories from a drop down menu. In a future update this could be improved to offer drag and drop functionality to move around DAGRs between categories.

## 6. User Manual

We would like to give a few descriptions on how to use some of the main features of the MMDA database system.

### 6.1 Insertion

From the insertion page a user can choose to enter a locally stored file, a URL link to a file, or an entire directory.

A user also has the ability to insert the new file as a brand new DAGR or insert into an existing DAGR.

Home/About	<b>Insert and Bulk Insert</b>	Query Executioner	HTML Components	Categorization	Deletion
Misc. Tasks					

**Insert via local path:**

**Insert via URL:**

**Bulk Insert (enter directory path):**

\*Supported file types are: .docx, .xml, .txt, .mp3, .wav, .jpg, .png, .gif, .mp4, .mov

Should a user choose to insert the file as a new DAGR, they will be presented with the following screen.



Home/About	<b>Insert and Bulk Insert</b>	Query Executioner	HTML Components	Categorization	De
Misc. Tasks					

## Inserting file(s) at 'C:\Users\Frank\Documents\portfolio.html':

DAGR Name:  Existing DAGRs to Inherit: 

frank  
imgur1  
imgur2

Keywords:

\*Supported file types are: .docx, .xml, .txt, .mp3, .wav, .jpg, .png, .gif, .mp4, .mov

The a meaningful name should be chosen, and some keywords to help describe the contents. If a user wishes to create the DAGR as a parent to a previously existing DAGR, they should choose from the list on the right.

Should a user choose to insert the file into an existing DAGR they will get the following screen

Home/About	<b>Insert and Bulk Insert</b>	Query Executioner	HTML Components	Categorization
Misc. Tasks				

## Inserting file(s) at 'C:\Users\Frank\Documents\portfolio.html':

DAGR Name: 

frank

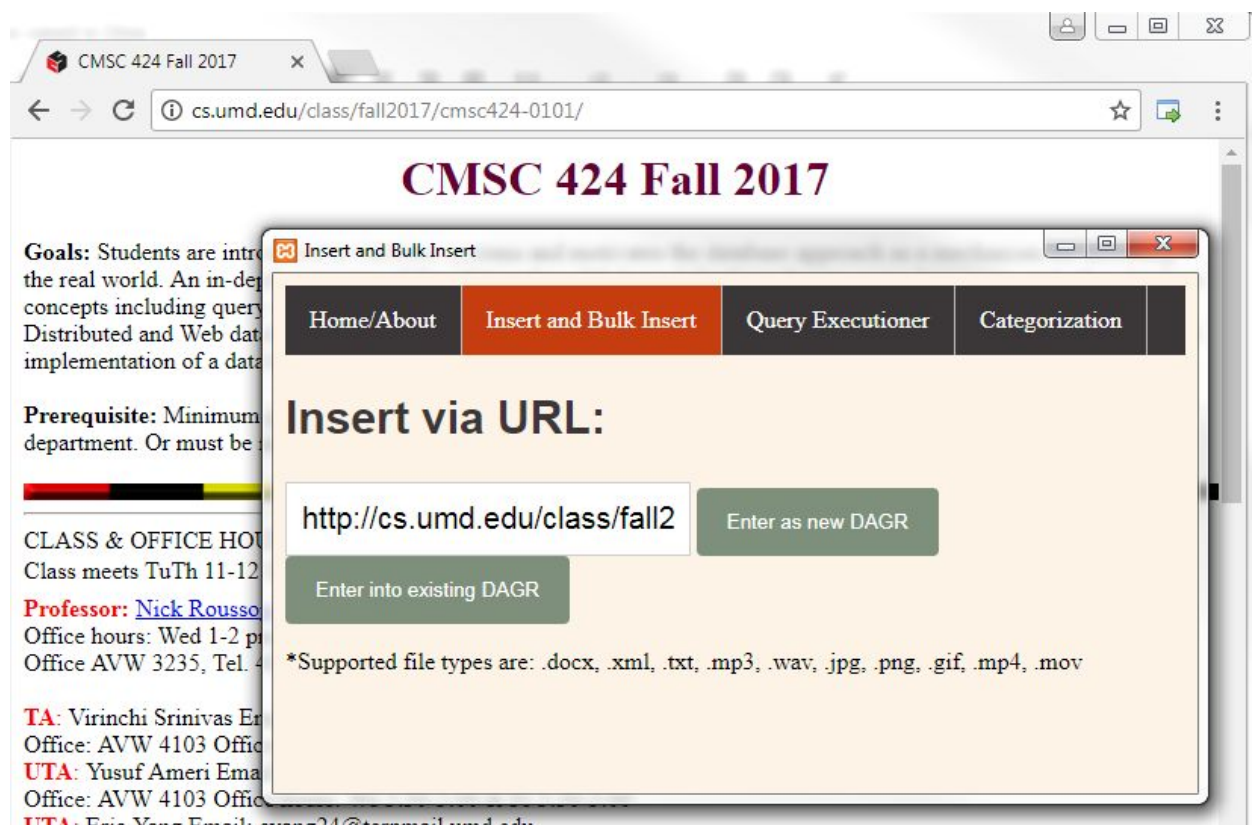
 Keywords:

\*Supported file types are: .docx, .xml, .txt, .mp3, .wav, .jpg, .png, .gif, .mp4, .mov

The user does not get to declare a name, as it is becoming part of another DAGR which has previously been named. They can still add keywords however. The DAGR they wish to insert into should be chosen from the drop down on the left.

## 6.2 Insertion via Browser Extension

If a user adds the chrome browser extension to their computer, they can insert the current page on the browser. To begin the insertion, the browser extension icon(located in the top right of the webpage in the below picture) is clicked. When clicked, the following page will pop up on the screen.



This page is simply a modified insertion page with the URL path pre filled in with the current browser page. The steps for inserting are the same as above.

## 6.3 Querying Metadata

Documents can be searched for in the database by a number of properties such as file size, date entered into the system, date the document was originally created, etc.. A user will navigate to the “Query Executioner” section of the webpage and identify their search parameters.

Home/About	Insert and Bulk Insert	Query Executioner	HTML Components	Categorization	Deletion
------------	------------------------	-------------------	-----------------	----------------	----------

## Query Execution

Types of Files to Display: ☐ Image ☐ Text ☐ Audio ☐ Video ☐ HTML

Search by DAGRs:    or by Category:

Date Created:  to

Date Entered:  to

Size:  to

Keyword:

Order by:

If a query is successful they should see a page similar to the picture below.

Execute Query									
Images									
DAGR Name	Name	File Size	Keywords	Date Created	Date Entered	Path	File Type	Image Width	Image Height
imgur1	saDYhT5	2375053	found online	2017-12-07	2017-12-08	https://i.imgur.com/saDYhT5.jpg	jpg	3492	4656

## 6.4 Additional Instructions

The above instructions were included as they highlight the main features of the database, adding data and querying data. There are many more tasks that can be performed on the system but we have provided helpful descriptions on the webpage rather than documentation here.

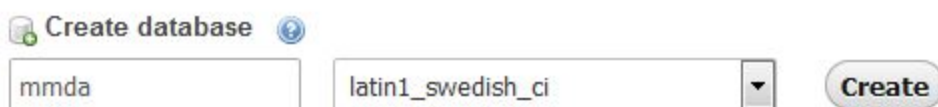
Should a need for assistance in using the MMDA system be required, either of us can be contacted

## 6.5 Instructions to set up the database

Should a user or developer need to reset the database, or created a new one, the task can easily be accomplished. The first step is to navigate to localhost/phpmyadmin/

A new database should be created from phpmyadmin with the settings shown below

### Databases



This will create a new MMDA database. However, this contains no tables. To generate the tables, the user should next navigate to “<http://localhost/CMSC424/main.php>”

This webpage is designed so as soon as you visit it, a function in the background creates the appropriate tables in the MMDA database we just made. Upon success you will see the following message

## The MMDA (Multi-Media Data Aggregator)

Created by Austin Piel and Frank Tiburzi

Your database is set up, use the menu bar to navigate the features of our application.

## 7. Testing and Error Handling

### 7.1 Testing

During the implementation of the system, we put it through extensive testing.

For every file type supported in our system, we went through the process of the four insertion methods. For example, a .png file. To ensure that .png files were inserted correctly, we tested

1. Inserting a .png file from a local directory
2. Inserting a online hosted .png file from a URL
3. Bulk inserting a directory of .png files
4. Inserting a online hosted .png file from the browser extension

This testing process was done for each file type.

Other functionalities of our MMDA system were tested just as thoroughly. All types of hierarchical data organization was tested by using the web interface to perform the desired task, and then confirming the results in the MySQL database by checking that the appropriate tuples were generated

### 7.2 Error Handling

It is difficult to 100% eliminate errors in a complex system such as the MMDA database. To combat this, we have tried to design a system that causes no issues for a user should there be an error.

If an error occurs during a database transaction, the database connection is dropped and nothing is performed. This is important if you run into an error when getting metadata from a file and inserting. For example, if for some reason the system gets an error extracting metadata from a file, the system will not commit some of the metadata that it successfully got and just leave other fields blank, but will just return an error message and insert nothing.

Another situation we handled a possible error situation was in parsing html. The current method of parsing, extracting metadata and inserting to the document has a high performance requirement. If too large of a web page is entered, the system could spend minutes to hours even, depending on the size of the site, trying to parse every single page linked. This would lock up the user's system enabling it useless. We have added a sensible timeout period to prevent extreme processing times

## 8. Conclusion

We have created a large scale database system with a dynamic front end and many features. Although our implementation may not be perfect, it address the tasks of the project and most importantly, we believe it demonstrates a great range of skills learned in both CMSC424 and the computer science program as a whole. This project has helped to show how powerful a properly designed database system is, and how much work it takes to implement one. We hope to have a chance to work with databases again in the future and we hope you enjoy our implementation of the MMDA.