University of Maryland at College Park

CMSC 424 - Database Design - Fall 2017
Nick Roussopoulos

DATABASE DESIGN PROJECT
MMDA: A Multi-Media Data Aggregator

1 Purpose of the project

You are to analyze the requirements for, design, implement, document, and demonstrate a database system that supports a Multi-Media Data Aggregator (MMDA).

2 Application Overview

The MMDA allows the users to name, annotate, store, and organize collections of data files into data aggregates called Data-Aggregates (DAGR for short). A DAGR may contain an arbitrary number of multi-media files, e.g. text documents, images, audio, video, and other previously created DAGRs. The system should provide functionality for storing, searching and retrieving the DAGR database using the DAGR's name, or keywords in their annotation, and other metadata (see below) that is automatically generated at the time of its creation. Each DAGR must be assigned a Globally Unique Identifier (see Wikipedia) that can be generated by an online server like http://www.guidgen.com/ or by code that is available on the Internet such as http://wiki.gigaspaces.com/wiki/display/SBP/Global+ID+Generator, code which you can run on your machine. The components of the DAGRs are either locally stored files generated from scratch, or obtained via a web browser (URL), received through e-mail, or other DAGRs. Each of the DAGR components, including URLs, receives its own GUID which is then linked with the GUID of the containing DAGR. You can think of a DAGR as a collection component GUIDs some of which reference local files and some external URLs. A very important feature of a DAGR is that it is light weight. It contains only references to either local files or URLs to external content. A DAGR does not contain heavy weight content but still encapsulates all the metadata for materializing the referenced content. This makes a DAGR very versatile because it can be sent by email or short messages. For transfers, think of it as a variation of Dropbox or Filelink.

The MMDA will store and retrieve DAGR metadata using meta-data attributes (document creator/author, the creation/modification time, the size of the document, but not the complete content of it. When a document is stored in the MMDA database, a new DAGR is named for the document and can have its own annotations that describe it. The DAGR database will also capture metadata about DAGRs, creation and deletion time, creator's name, and whether or not it contains component DAGRs. Because Multi-media files may contain references to other files (e.g. an HTML file may contain images, audio, video, and URL references) it is important to capture the connections between the components of the HTML documents for maintaining and discovering broken links and other reachability queries.

It should be emphasized that the MMDA is mainly concerned with design of a database that stores metadata. The design should be general to model all possible files stored in a computer and named grouping of such files, DAGRs, and/or other DAGRs. You have to makes sure that the design does not have idiosyncratic

features to the particular dataset that you will use to demonstrate the project. The metadata database should be able to answer queries about the stored files and the user defined named DAGRs. The project would have to rely on the OS for opening and/or displaying the underlying objects.

3 Application Requirements

3.1 The Metadata DAGR Database

The MMDA has to deal with all sorts of documents such as text (HTML, XML, MS word, latex), images (jpeg,gif), videos (mov), audio (CDs, au), software (source, binaries), etc.[1]

3.2 Tasks

This section describes the main tasks only. You may add, modify, and/or substitute some of these with other more interesting ones depending on the emphasis of your project.

- **Insertion of a new document in the DAGR database:** When a document is inserted in the DAGR database, it is first assigned a GUID (it is GUIDidied!) and its metadata (document name, storage path, creator name, creation time, last modification time, etc.) is entered in the DAGR database. The name of the file name is used as the name of the DAGR. The user can later change the initially given name of the DAGR to be different than the underlying file name. Note that the underlying file names NEVER change when associated with a DAGR. A DAGR for a file is a self-documenting entry in the DAGR database. The name of every DAGR can be changed and annotated with keywords and phrases by the user. Such annotations will be used to search for DAGRs whose annotation matches given keywords. Because GUIDs are not memorable, e.g. 587a82f5-de1b-4886-b097-6b5fa9124f56, the user should be able to create a meaningful name for any DAGR. For a locally stored files, the metadata is obtained from the file system.

- **Bulk Data entry:** A bulk data entry utility must be implemented. For example, to digest all your files from your PC or include your music CDs , you should use lists (ls or dir) to obtain the metadata and automatically ingest it in your database. For each of these file, you have to generate a DAGR GUID, extract metadata from the file system and enter the appropriate records to the DAGR database. Obtaining the metadata from a web browser interface is not straight forward. You will be given separate instructions on how to do it and what to bulk insert. Naming and annotation of DAGRs can be done manually by the user as above.

- **HTML parser and automatic data entry:** HTML documents need to be parsed to identify their components and assign component GUIDs. This task must extract all the metadata of a component. For a locally stored file the above tasks will provide the required metadata fields. For an external URL reference, you can harvest some keywords from either the URL text or the text surrounding the URL. The extracted textual metadata will be automatically stored in the annotation of the corresponding

[1] You can select a reasonable subset for which you can design and implement a more thorough system than a general one that deals with everything but, in this case, it has to demonstrate substantially more functionality to account for the high weight of the project grade (33%). Over-simplifying assumptions will not be accepted.

DAGR component. The quality of this extraction process will only be applied to earn extra credit on the project. You can assume a maximum depth of 2 to extract all the links from a given html page which become new (contained) DAGRs.[2]

- **Web Browser Interface:** This is one of the most important tasks. It provides a friendly user interface to the DAGR database. It should provide a way to create a new DAGR for an HTML document displayed by the browser or allow to add the displayed HTML document to an existing DAGR. The best way to implement this is by extending the browser's API to add a plug-in (add-on). Firefox provides such an API https://addons.mozilla.org/en-us/developers/docs/reference. The add-on will keep a list of open DAGRs and the user may click to add the displayed HTML document to one of the open DAGRS. The naming and annotating function can be invoked as well to complete the entry.

  The same interface should be used to search and display the metadata of existing DAGRs. The display should be in a tabular format but with local or external URLs references to be clickable to access their underlying content.

- **Support of Categorization:** The user should be able to create a wide variety of categories using the DAGR mechanism. For example, a user may create a DAGR called NEWS for aggregating breaking news (CNN, WTOP, NYT), or one for COUPONS (groupon, Amazon), TRAVEL for travel promotions (Expedia, VacationToGo,Kayak), etc. The interface must support the creation of categorical DAGRs, subcategories with a category, insertion and/or deletion of DAGRs in them, and the hierarchical display of DAGRs.

- **Deletion of a DAGR:** When a DAGR is deleted all appropriate metadata and its links have to be updated to avoid the problem of "dangling references". Note that the deletion of a DAGR only eliminates the metadata in the database and does not delete any local files or external references. During a deletion of a DAGR, this action should be propagated up and down the hierarchy of DAGRs. First a list of the parent DAGRs affected by the deletion should be presented to the user and ask for a user confirmation before actually deleting it. Upon such approval, the affected parent DAGRs should be marked as deleted. Furthermore, when a DAGR is deleted, the user should also be presented with a list of daughter DAGRs. Again, the user should be presented with a list of daughter DAGRs and be asked for approving either a "shallow" or a "deep" delete for each daughter. The shallow delete will only remove the daughter from the deleted DAGR where as the deep will apply recursively the delete task for each of the daughters.

- **DAGR Metadata Query:** Typically, we would like to write queries to find DAGRs using the metadata attributes, such as date, author, type, size, or keywords in the annotations.

- **Orphan and Sterile Reports:** Generate all the DAGRs that are not referenced by a parent DAGR or have no descendant DAGRs.

- **Reach Queries:** Very often we would like to find all the DAGRs that can be reached by a given DAGR (descendants) or those that point to it (ancestors. The reach queries may extend to several

[2]The project does not require elaborate link mining or parsing other types of documents but if some group has some innovative ideas on how to extend the metadata extraction, it is possible to arrange but requires instructor's approval.

levels of depth which should provided as an argument, e.g. Down 2 or Up 1. But in this case you have to store appropriate information during the retrieval to avoid looping. The default for such queries should be 1 level.

- **Time-Range DAGR Report:** Generate a report with several preselected data items for all the DAGRs created or entered in a given time range.
- **Identify Duplicate Content:** Some of the GUIDS may correspond to the same content. This could happen if two users enter the same document in the MMDA database thus obtaining different GUIDs for the same data document. This task requires scanning and differencing the underlying documents. When two distinct GUIDs are referring to the same object, all occurrences of them should be replaced with one of the two.
- **Modify a DAGR (Optional):** For locally stored files, we can use the delete and insert function. However, for HTML documents, this is a more challenging task and requires an HTML editor. For this reason this is optional and can attacked for extra credit[3].

4 Rules of the game

- **Groups:** Due to the size of the class, the project is to be done in groups of two students. Therefore, you are expected to produce a substantial design and implementation of the MMDA. Each student will be randomly assigned to a group. You will find your partner on ELMS. The groups are "self-policing" (e.g., each group is responsible for its own division of labor, scheduling, etc.). *Note: If an irreconcilable problem arises in your group, it is your responsibility to contact both the professor and the TAs as soon as possible. After the project is due, it will be too late.*
- **Assumptions:** In cases where the above description of the application is incomplete, it is acceptable to make assumptions about the application providing that: 1) they are explicitly stated in the report, 2) they don't terribly conflict with any of the requirements specified above, and 3) they are "reasonable" (e.g., it is *not* reasonable to assume that the your MMDA can only store a single type of document which have no linking to others. If you have a question about the acceptability of any of your assumptions, check with the TAs or the professor.
- **Reports:** A report should be handed in for checking at the end of each phase. The report must be formatted in a reasonable manner (i.e., using a text processor and a decent printer). Reports are due during class on the date specified in the "Due Dates" section below.
- **Implementation:** The final phase of the project requires a working implementation of the system to be built, tested, and demonstrated. A large part of the project grade depends on the quality of this implementation. The implementation will be done using either the ORACLE database system or MySQL running on your PC.

[3]The maximum extra credit you can earn in this project is 5 points.

5 Project Phases

The three phases of the project cover the following work-processes from the paper *An Adaptable Methodology for Database Design* [1] by Roussopoulos and Yeh and its implementation http://www.openmodels.at/web/sdbd in the Open Models Initiative.

Phase	Phase Name	Due Date
I.1	Environment, Requirement and System Analysis with Specification,	Oct 3, 2017
I.2	Implement a web server on a demo machine (i.e. your laptop)	Oct 3, 2017
II.1	Conceptual Modeling and Task Emulation	Nov 2, 2017
II.2	Establish a DBMS system (e.g. Oracle, MySQL or PostgreSQL) and Implement the connection to the web server of I.2	Nov 2, 2017
III	Implementation, Final Report, and Demo	Dec 8, 2017

6 Reports

All reports must be submitted on ELMS by the midnight of the Due Dates. No hard or email copies will be accepted.

The Phase I report must contain:

1. a description of the scope of the project
2. a description of the technical/conceptual problems encountered in this phase and justification for the solutions.
3. the assumptions that you have made about the enterprise.
4. all the documentation produced in this phase, i.e.
   - the top-level information flow diagram, (*very* important)
   - the list of tasks and subtasks
5. one or two lines saying if you were able to implement the web server. Also add any configuration details of the web server you used.

The Phase II report must contain:

1. Phase I- with corrections addressing the TA's feedback.
2. a *short* description of the purpose of this phase of the project.
3. a description of the problems encountered in this phase and justification for the solutions.
4. the documentation produced in this phase, i.e.,
   - the graphical schema using the E-R model. Again the E-R should be general and not specific to any particular dataset.
   - the relational schema obtained by mapping the E-R to relations, and their Boyce-Codd or 3rd Normal Form with keys.

- the pseudo-code for each task and the embedded DML code.

5. one or two lines saying if you were able to setup the database used(e.g. Oracle, MySQL or PostgreSQL). Were you able to implement the connection to the web server implemented in Phase I.

The Phase III report must contain:

1. Phase I and Phase II reports with corrections addressing TA's feedback.
2. a description of the purpose of this phase of the project,
3. a description of the problems encountered in this phase and justification for the solutions.
4. any revisions made to the relational schema definition from Phase II,
5. documentation produced in this phase, i.e.:
   - a short users manual for the system.
   - your testing efforts: erroneous cases that your system can detect and handle reasonably.
   - a description of the system's limitations and the possibilities for improvements.
6. In addition, a demo of the system is required. All members of the group should attend this demo, to explain the aspects of the project for which they were responsible.

References

[1] N. Roussopoulos and R. Yeh. An adaptable methodology for database design. *IEEE Computer*, 17(5):64–80, 1984.