Museum Artifact Database

Semester Project Report

CS 470

Professor Kendall Bingham

Team Members:

Evan Baranowski

Jonathan Davis

Latrese Davis

Darec Stowell

TABLE OF CONTENTS

- Project Description pg. 3
- Product Description pg. 8
- Deficiencies and Future Scope pg. 9
- Appendix pg. 12

PROJECT DESCRIPTION FROM START TO FINISH

Brainstorming Decisions

The first step our group took in working on our database was to figure out what kind of database we wanted to implement. We had multiple ideas on the table, such as an auction, or a hospital scheduler, but we ended up choosing to create a database for a museum. This was the topic that interested the group the most. However, we may have been too gung ho about creating a museum database. We initially wanted to store all of the information about a museum, from information about artifacts to fundraisers. After we realized that we might be biting off more than what was feasible to implement, we decided to make our database specifically about the artifacts within a museum.

Research after Trial and Error

Once our group had the topic our database would be about, we started to try making entities and relationships. After a few excruciating hours we realized how hard it was to make entities and relationships about something we did not fully understand ourselves. This caused our group to restart from our topic idea. However, this time we decided to do research on our topic. To help us narrow our research down, we chose four users, Curator, Conservator, Archivist, and Tour Guide, our database would cater to. This allowed us to look up research specifically relating these users to artifacts. However, once again we developed

too many requirements for our database. So Jon had a great idea and decided that each of us would go through all of the four user's requirements and "spend" $100 on requirements that we deemed most important. The top 5 requirements from each user's requirements would stay. This allowed us to cut a lot of ideas from our project and help us focus on what was important about our project.

Architecture, Schema, and Platform Decisions

The next step in our project was developing the architecture, platform, and schema. We decided to implement a two-tier architecture for our back-end database. We decided this because we thought since our schema was so complicated that we should keep our architecture simple. With a two-tier architecture our program would just return the information requested. This strayed from our main goal a little because we wanted to be able to insert and delete entities, but for the time being we were just trying to not make things as complex as we had earlier. We also tried to keep the front-end as easy to set up as possible since we predicted spending a lot of time on the back-end developing our database. This lead us to choose Windows Forms for our front-end development because WinForms has the ability to drag and drop features right into your app. WinForms came with the .NET framework, which is relatively easy to use, and it came with C#, which is the language we wanted to implement. Our schema at the time was relatively small for what it turned into. At this stage in our project, we had entities named Artifact, Location, Collection, Exhibition, Time Period, and Museum. Once we got to normalizing our database our entities and relationships seemed to multiply!

Normalization Decisions

We then took these 6 entities and fleshed them out, which showed just how interconnected each entity was through relationships and foreign keys. So we began trying to iron out all of the values that we had in each entity. This led us to creating more entities and relationships just to sort out the mess that we had within our schema. Thankfully, a week after we encountered this problem, our class learned about normalization. From what we learned in class, we were able to disperse our information into more entities and relationships. However, we ended up normalizing a lot of the database already, but there were a few tricky spots, such as entities relating to the Conservator and our location entities, that normalization helped us iron out. For instance, we wanted to have display locations and storage locations, which both would include storage requirements for a particular artifact. However, since there could be any number of storage requirement attributes we violated 1st Normal Form. This led us to decide that we do not need separate tables for display and storage locations because both could be included as an attribute within a Location entity. We also had to change our Artifact entity. Within artifact there was an attribute called Artist, which would hold the name of the artist of the work. However, not every artifact had an artist, which would create many NULL values within our table. In order to cut down on the NULL values, we decided to create a new entity called Artist.

ASP .NET MVC and Azure

Even though we had done research and decided to use WinForms, we never really decided what database platform we wanted to use. Jon got a promotion from Microsoft Azure

granting new users $200 of free general access, which included up to 10 GB of storage. Azure gave us free database access and allowed us to create our database in the cloud, which sold the group due to how accessible Azure is. We also decided to drop WinForms and move to a three-tiered architecture system, ASP .NET MVC. This allowed us to migrate a database into Visual Studio and use the Entity Framework within Visual Studio. The migration process was easy because all we had to do was fetch our database schema and Visual Studio created C# classes modeled for our database tables. Also, the MVC framework made it extremely easy to create models of our entities, fill them with data, and push the data to the application through a view. Azure was also easy to use because it took in DDL statements just like a SQL Server Database. Each member within the group implemented the table statements they were assigned. As a group we entered in 33 table statements! With the relative ease of the project at this stage, we knew we made the right choice in switching platforms and architecture.

Stored Procedures and Implementation

Each member within the group had to create Relational Algebra statements for the user they had been assigned to throughout the project. Together we came up with 30+ statements! However, we did not have enough time to implement all of these relational algebra statements, but we did implement 12 of them as stored procedures in Azure. By implementing them as stored procedures, implementing our queries was very simple because Entity Framework imported these as well. After these queries were implemented into Azure, each member had to write C# code to call the stored procedure in the MVC. Once this was complete, we tested each and made sure the correct results were returned. However, we did run into a couple of issues such as when MVC creates a C# model class for the query result of a stored procedure, it expects that class to

have a primary key defined. We tried to add a view that corresponded to the particular C# model class in question, but to no avail. However, after searching the Stack Overflow and Google, we finally found our answer. Another issue was pressing ctrl-Z deletes everything that has been typed into the query. So if we did not save what we had typed, there was no way to recover that query. All in all, ASP .NET MVC and Azure made our implementation relatively easy, though.

Deployment

Deploying our application to the cloud was relatively straightforward. There were three steps:

i. Create a web-site in Azure that is associated with your database.

ii. Download a publish profile from Azure for your website.

iii. Simply click "publish" in Visual Studio and connect your solution to the publish profile from Azure. Sit back and watch the magic happen. Before you know it, your site is live on the internet, all for free!

Making changes to the front end or back end after deployment was not so straightforward. We discovered that any changes made to the MVC application must be migrated in a very particular way to the database, and vice versa. Otherwise, any part of the web-site that relies on the database simply will not work. Since our team members are all novices and self-taught with these technologies, it's no surprise that we hit such roadblocks. In the end, the entire MVC application had to be re-written and re-deployed to an entirely new site in Azure. This was extremely time-consuming and thus we were not able to implement as many features as we had anticipated. Not to mention that once we re-deployed a functioning web-site, we opted not to change anything lest we risk crippling the site gain.

PRODUCT DESCRIPTION

Our team created a database application to handle a museum's artifact-management needs. This database catalogs a museum's artifacts, storing a wide range of artifact-relevant information from the artifact's location all the way to tools used to experiment on or clean an artifact. This information can be queried by users of the system. Each user has limited access upon the information of an artifact. The system is designed to release information to certain employees based upon their role in the museum. We focused on four roles: curator, conservator, tour guide, or archivist. By limiting a user's access to the database based on their role, we had to implement a complex back-end architecture to handle the functionality of the website. By having such a complex architecture on the back-end we were able to create an easy to use website with a user-friendly interface. We then named our website accordingly, Artifact Express, due to the relative ease of requesting information about an artifact.

DEFICIENCIES AND FUTURE SCOPE

The scope of our project quickly grew as evidenced by our schema and Entity Relationship Diagram (ERD). We took steps to reduce the schema by limiting ourselves to entities and relationships that were artifact-centric. After normalization, our decision to implement via Microsoft's ASP.NET MVC 5 and Windows Azure allowed us to streamline the development process and present a total package display. In the end we were able to deploy many features to the MVC application, but the enormous scope of our project combined with our limited experience and time did not allow us to implement all our ideas, leading to certain deficiencies. Some ideas we cultivated that would benefit from further development:

1. Although our database is setup to handle pictures corresponding to artifacts, we did not have time to implement this functionality within the MVC application. Having a pictorial history of an artifact would be very useful to every museum user and would be a prime candidate for future work.
2. We created four user roles: Curator, Conservator, Archivist, and Tour Guide. To administer these roles, one must access the database on Azure and manually assign a registered user his/her role. A future implementation would include an Admin role in which an administrator could assign roles in the application itself.
3. We spent ample time tweaking the idea of "Location" and its meaning with regard to an artifact. Inherent in an artifact is a need for preservation which in turn assumes a specific type of environment. In our database we addressed these needs with the idea of a "Storage Requirement." An artifact has specific storage requirements while certain locations meet specific storage requirements. The tables that represent these ideas exist in our database, however, due to time we did not implement the functionality to access these tables in our MVC application. This would be valuable functionality for a museum conservator and a good candidate future work.

4. We tested our application thoroughly with valid data that we knew should produce positive results. Indeed, the application queried the database and produced the expected results every time. We did not, however, test thoroughly with invalid data. When such testing was performed, the application did not break down, but simply produced empty results, which, although not ideal, we deemed acceptable.
5. We created over 30 relational algebra statements yet had time to implement less than half. Here are a few that would be easy to implement and useful for users:
    a. Find all artifacts by a certain artist
    b. Find all artifacts from a certain country
    c. Find all exhibits, past, present, and future
    d. Find all artifacts by a certain artist that are currently on display
    e. Find all artifacts from a certain time period that are currently on display

Were we to expand our database to incorporate the greater concept "museum", the sky is the limit with regard to scope. This was originally our intent until we discovered how vast such a project would be. Here are examples of ideas we had to set aside early on:

1. We could extend to our application the ability to manage museum auctions. We would want to track the artifacts up for auction, their estimated values, information regarding the patron selling the artifact (if not owned by the museum), information regarding the patrons attending the auction, dates to track the auctions, and other necessary information.
2. Donors are of the utmost importance to a museum's success. A museum curator would want to know every major donor and the major items he/she has donated. In addition, what recognition(s) has the donor has received for his/her charitable donations? What gala events has the museum sponsored that the donor has attended? And whatever information he/she deems necessary to keep maintain a close

relationship with these important contributors. An application that makes this information easily accessible would be very valuable.

3. (Not in our original plan) One significant addition would be to extend the database's loaned and borrowed feature to accommodate transactions between museums on an international scale.
4. (Not in our original plan) Using our current model, the museum database could be extended to provide access to guests and other employee positions. For example, a guest could make use of an onsite terminal to access basic artifact information. Using a search box, a user could type in the name of an artifact and the results would provide a location and a minimal amount of information, if available. Otherwise, it would give a message stating that the artifact is not present or unavailable for viewing. In theory, we would curtail the application and database software to the meet the user needs of the specific museum purchasing our software.

APPENDIX

For report items not contained within this document, please see the following files included in the project submission:

- ER Diagram:
  - For our initial ER Diagram see file: ERD Initial.pdf
  - Per your suggestion, we broke our ER Diagram into three smaller diagrams for our final submission:
    1. Artifact Preservation, Treatment Regimen, Treatment Technique, Treatment Tools, Test.pdf
    2. Country, Collection, Exhibitions, Location.pdf
    3. Photograph, Time Period, Storage, Loaned-Borrowed.pdf
- Normalized Relations:
  - Artist Normalization Process.pdf
  - Location Normalization Process.pdf
- Test Data
  - Test Data.xlsx
- Relation Schema
  - Musuem Database Relation Schema - Final.xlsx