Final Project Report

By Devanshi Shah, Zackary Miles Richards

Professor Diker & Professor Duffy

INST733

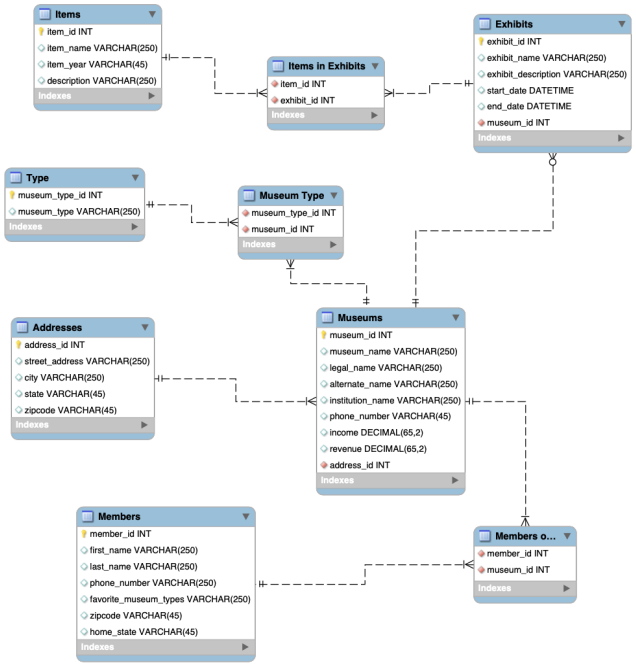
11 May, 2022

Introduction

Do you like museums, zoos, aquariums, planetariums, and similar attractions? Have you ever found it difficult to find fun and educational things to do by yourself or with family and friends? If you have answered yes to either of those questions, I have great news for you. We would like to introduce to you the database of all databases for finding museums, zoos, aquariums, historical museums, planetariums, and more. Our database contains extensive information regarding the locations, attractions, and museum types of museums in the United States. This database will make it easy to find exactly what kind of museum you are looking for, whether that is a specific museum type, a museum near you, a large museum, a small, local museum, or a specific exhibit or artifact at a museum that you are interested in. If you are the indecisive type, our database will also be able to provide recommendations for museums using user profile information as well. This database can also assist schools and teachers in finding exciting field trip locations for students.

Database Design and Implementation

ERD



When initially designing the ERD, we wanted to make the Museums table the central focus of the database. We feel like this is reflected in our ERD design as it has the most foreign key relations in other tables. On top of this, you can see clearly in the ERD that all other tables branch out from the Museums table. All of the relations between tables are one to many with the exception of the one to many or none relationship between Museums and Exhibits tables. The idea behind this was to allow for a situation where a museum's data has been included in the database but for whatever reason they do not have any exhibits(maybe the museum is new or is structured in a way where exhibits are not a part of the museum's experience).

Physical Database Design

In the following section, you can find our physical database structure for the tables.

Tables Key:

Primary Key, Foreign Key, Not Null(NN), Auto Increment(AI), Unique(UQ)

Addresses - address_id(NN)(UQ)(AI), street_address(NN), city, state(NN), zipcode(NN)
Exhibits - exhibit_id(NN)(UQ)(AI), exhibit_name(NN), exhibit_description, start_date, end_date, museum_id(NN)
Items - item_id(NN)(UQ)(AI), item_name(NN), item_year, description
Items in Exhibits - item_id(NN), exhibit_id(NN)
Members - member_id(NN)(UQ)(AI), first_name(NN), last_name(NN), phone_number, favorite_museum_types, zipcode(NN), home_state(NN)
Members of Museums - member_id(NN), museum_id(NN)
Museums - museum_id(NN)(UQ)(AI), museum_name(NN), legal_name, alternate_name, institution_name, phone_number, income, revenue, address_id(NN)
Museum Type - museum_type_id(NN), museum_id(NN)
Type - museum_type_id(NN)(UQ)(AI), museum_type(NN)

Sample Data

The sample data consisted of real data from our Kaggle dataset, data we gathered through Google searches, and data we created ourselves.

Views and Queries

In this section, we will go over all of the queries written for this project.

1) List all the art museums in the state of California where the revenue exceeds 50k.

```
USE museums;
```

```
DROP VIEW IF EXISTS art_museum_CA;
CREATE VIEW art_museum_CA AS
SELECT museums.museum_id, museums.museum_name, `type`.museum_type,  museums.revenue, addresses.state
FROM museums
INNER JOIN addresses USING(address_id)
INNER JOIN museums.`museum type` USING (museum_id)
INNER JOIN `type` USING (museum_type_id)
WHERE addresses.state = "CA" and `type`.museum_type = "ART MUSEUM" and museums.revenue > 50000;
```

This query was written to showcase how a business or a competing museum might use our database. This query returns one row and includes all columns visible in the screenshot.

2) Find the museum in the state of Ohio which has the highest income.

```
USE museums;

DROP VIEW IF EXISTS highest_incomeOH;
CREATE VIEW highest_incomeOH AS
SELECT museum_id, museum_name, MAX(income) AS 'highest_income'
FROM museums.museums
INNER JOIN addresses USING(address_id)
WHERE addresses.state = "OH"
GROUP BY museum_id
ORDER BY income DESC
LIMIT 1;
```

Again, this query was written to showcase how a business or a competing museum might use our database. This query returns one row and includes all columns visible in the screenshot.

3) Find all the museums with more than one item, return all exhibits they are hosting(with dates and their items)

```
use museums;
drop view if exists exhibits_w_item;
create view exhibits_w_item as
select museum_name, exhibit_name, start_date, end_date, item_name, item_year, `Items`.description
from `Museums`
inner join `Exhibits` using(museum_id)
inner join `Items in Exhibits` using(exhibit_id)
inner join `Items` using(item_id)
order by museum_name asc;
```

This query was written to showcase how the database can be used to find interesting items, or museums with a large amount of items. This query returns 50 rows in total and includes the name of the museum the item lives in, the name of the exhibit it is a part of, the starting and ending dates of the exhibit, and relevant information about the item itself(name, year, and description).

4) Find the number of exhibits that each museum has had, if they have had at least one, along with its income and revenue.

```
USE museums;

DROP VIEW IF EXISTS no_of_exhibits;
CREATE VIEW no_of_exhibits AS
SELECT museum_id, museum_name, COUNT(exhibit_name) AS "number_of_exhibits", income, revenue
FROM museums.museums
INNER JOIN museums.exhibits USING (museum_id)
GROUP BY museum_id
ORDER BY COUNT(exhibit_name) desc, revenue desc;
```

This query was written to showcase the competitive utility of this database for businesses and competing museums. Perhaps they would like to see if there is a relation between number of exhibits and revenue. This query would provide them with the data necessary to perform this analysis. This query returns 39 rows and includes all of the columns visible in the screenshot.

5) Rank the museum types from most popular to least popular(most members to least members)

```
use museums;

drop view if exists popular_museum_types;
create view popular_museum_types as
select count(member_id) as "Number of Members", favorite_museum_types
from `Members`
where favorite_museum_types != ""
group by favorite_museum_types
order by count(member_id) desc;
```

This table showcases the popularity of each type of museum by counting the number of members who have declared each one of the museum categories as their favorite type. This query could be useful for a myriad of reasons.

6) Find all exhibits at Science and Technology museums after 2022

```
use museums;

drop view if exists museums.sci_tech_exhibits;
create view sci_tech_exhibits as
select museum_name, exhibit_name, exhibit_description, start_date, end_date
from `Museums`
inner join `Exhibits` using(museum_id)
inner join `Museum Type` using(museum_id)
inner join `Type` using(museum_type_id)
where `Museum Type`.museum_type_id = 2 and
(`Exhibits`.end_date > "2021-12-31 00:00:00" or `Exhibits`.end_date is null);
```

The idea behind this query is as follows: Let's say you are interested in attending science and tech exhibits at some time in the future after the new year of 2022(for the sake of this query, let's

say the year is currently 2021). This query would return all of the science and tech exhibits that would still be active and available for you to visit after 2022.

Changes from Original Design

The majority of changes that we made had to do with our tables and entity relationships. We got rid of the original Income/Revenue table that we proposed in our project proposal. This was because the data did not need to be in its own table. We also narrowed the amount of data we were initially planning to include. We had plans of using the entire Kaggle set of over 60,000 museums but that would have been a bit too much data for us to work with. Finally, we made changes to the physical database design in terms of which values will be not-null, auto incrementing, unique, etc.

Issues Experienced During Development

- Issues encountered
    1. The implementation of the logical design was completed with the help of the Entity-Relationship Diagram. While designing the logical design of the database, we initially did not add the correct Not-Null constraint to some of our entities in all the tables.
    2. After the completion of the logical design of the database, the next step was to populate the database with the data. Our sample database was in .xlsx initially. So the next issue we encountered was how to import the sample data into the database so that we could populate it.
    3. In our sample database in the excel sheet, certain columns had no data in them and were NULL. After we imported the data into our database, we found that the columns which were supposed to be NULL were instead treated as a string, and those columns were left blank.
    4. While importing the data of the Exhibits table, we were not able to implement all the rows of data that contained a NULL value. Only the data which had no NULL values were imported into the database using the Import CSV into MySQL feature.

- Solutions chosen
    1. After careful analysis of the ERD, the relationship between the tables and the type of entity we were dealing with, and the feedback we received from the professor, we decided to add the Not-Null constraint into the appropriate entities. None of our columns were non-null, so anybody could hypothetically create a new museum row with nothing but a museum_id and address_id. So we made the distinction over what is critical information that cannot be empty (i.e a museum needs a name but maybe not an alternate name) and added the Not-Null constraints to the appropriate entities.
    2. To import the data from the excel sheet to the MySql Workbench, we used the feature of Import CSV into MySQL. We first converted the data in the excel sheet into .csv format before we could use the import function to add data into the database. We chose to use the import data function of the MySql workbench because this process saved our time and it is not feasible to populate the database using the INSERT query due to a large amount of data.
    3. To solve the issue of the blank columns (data should be Null but it is being treated as a blank string) in the database after importing the data using the import function, we used the SET Field to NULL option to convert those cells into NULL records. Using this option saved us time as opposed to using the UPDATE function to convert each cell into NULL value would have been time-consuming.
    4. In the Exhibits table, there were 15 entries remaining that we could not add with the help of the import function as they each had a NULL column. We added those 15 queries manually by using the INSERT function.

Lessons Learned

1. We learned to create a whole database from scratch and got to learn about the various stages involved in the design of the database from logical and physical design to populating the database with data and writing the queries.
2. The process of transforming a visual data model into a physical database through the process of forward engineering wizard helps in generating all the SQL code automatically

and thus helps in eliminating the normal error-prone process of manually writing the complex SQL queries.

3. It is important to add not null constraints to the critical information in tables of a database as without using it anybody can add new data to the database with just a single key itself.

Potential Future Work

Extensions within the current technology and design (MySQL/MariaDB)

Because our database is a relational MySQl database, that allows us to take advantage of the features of popular commercially available MySQL databases. One such database is MariaDB. It is easy to migrate databases from MySQL to MariaDB and migrating the database to MariaDB offers more flexibility in terms of management of access rights of the data, faster loading speed and better database encryption. So essentially MariaDB offers tighter security measures as compared to MySQL.

Feasibility of Alternative Implementations

For our database transitioning to another paradigm like NoSQL would not be very beneficial. This is because switching to a NoSQL database would not only be a tedious process that would take a large amount of time but also because we would be losing many of the benefits of relational databases. Joins are not supported by the other paradigms like NoSQL, so when working with the NoSQL database, we would have to do multiple selections and then join the data manually. We don't need to normalize the data in the case of other paradigms so we will be missing important keys like primary keys and foreign keys which makes the handling of a vast database like Museums very difficult. Since other paradigms are non-structured, they need logical hierarchy in their database implementation which can be difficult to implement with a vast amount of data available in the Museums database.

Citations

Kirjoitti. S , n.d. . Reasons *for Migrating MySQL to MariaDB . Retrieved from.* https://somea.org/blog/reasons-for-migrating-mysql-to-mariadb/

Migrating from mysql to mariadb is easy! Geekflare. (2020, August 5). Retrieved May 10, 2022, from https://geekflare.com/mysql-to-mariadb-migration/

Institute of Museum and Library Services. (2017, March 6). Museums, aquariums, and zoos. Kaggle. Retrieved May 10, 2022, from https://www.kaggle.com/imls/museum-directory