CS5200 Project - Ethan Olesinski, Alex Oh (OhAOlesinskiE)
NEU Utilities
Git Repository: https://github.com/alex-oh/neu-utilities

README.md

For the README.md portion of our project and to see installation instructions please visit the project's github page or view the README inside of the zip file root directory.

Technical Specifications

Our database project utilizes a range of technology including tools, libraries, and frameworks to implement a database, backend server, and frontend GUI component. More details and specifications regarding the exact technologies used for each portion of the project are explained below.

Database

The database portion of our project uses a traditional mySQL database. We constructed our complete database schema using the mySQL Workbench tool used for other projects during the class. Since elements of our backend server were written using Python we utilized the mySQL Python Connector (https://dev.mysql.com/doc/connector-python/en/).

Backend Server

As specified above, we utilized the mySQL Python connector for creating a self-contained driver to interface with our existing mySQL database. Similar to projects constructed with other drivers (like JDBC) we provided specifications for, and stored in variables, values for a username, password, and URL for establishing a connection to a mySQL database.

Additionally, for constructing the backend server, we utilized the Python Flask package for helping to construct the project's API interface. By using Flask, we were able to construct routing functions to handle specific URL requests sent by our front end users when they interact with our web application. This provides the functionality that allows us to directly call many of the mySQL procedures and functions that we developed in our database schema. Within Flask we also used JSON functionality in order to send JSON payloads to and from the server. We also utilized Node.js for helping to construct this API.

Regarding our overall design choices for this portion of the project, we chose to encapsulate as much of the database functionality (READ, UPDATE, DELETE) into functions and procedures at the database level. This allowed us to successfully minimize the amount of SQL code that was used in the front end (at the application level) to essentially only include the query calls to the existing functions and procedures.

Frontend Web Application

For the frontend portion of our project we utilized the React and Javascript framework.

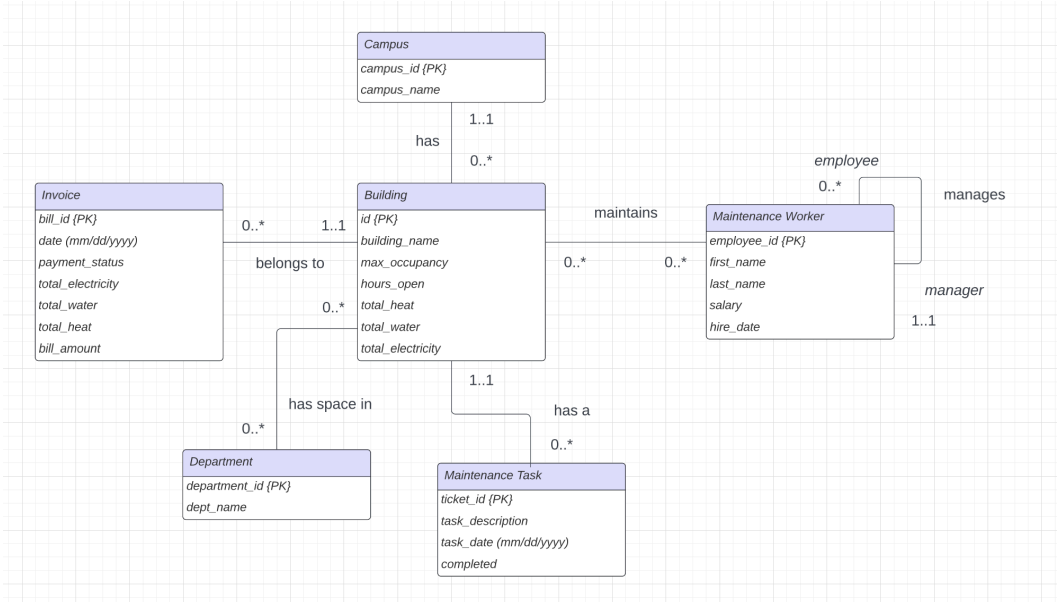
Conceptual Design / UML diagram

The schema for our database project includes 9 tables for representing a utility and building tracking database for a university campus. Entities in our UML model include an invoice, campus, building, worker, department, and maintenance task. Each of these contain specific attributes unique to each table.

Additionally, our schema includes two tables that represent relations between multiple entities in the database. The table for the relation maintains allows us to represent the fact that there exists a many to many relationship between a maintenance worker and a specific building on our campus; a building may have many workers assigned to or working at it while a worker may be responsible for addressing maintenance tasks for many buildings. Similarly, the has space table captures the fact that, in our conceptual model, a building may be used (or house office, classroom, lounge, etc. space) for multiple departments. As is the case with most university campuses, a department may also have multiple buildings. Finally, there is a table for representing a managing relationship that may exist between employees. This is a recursive relationship.

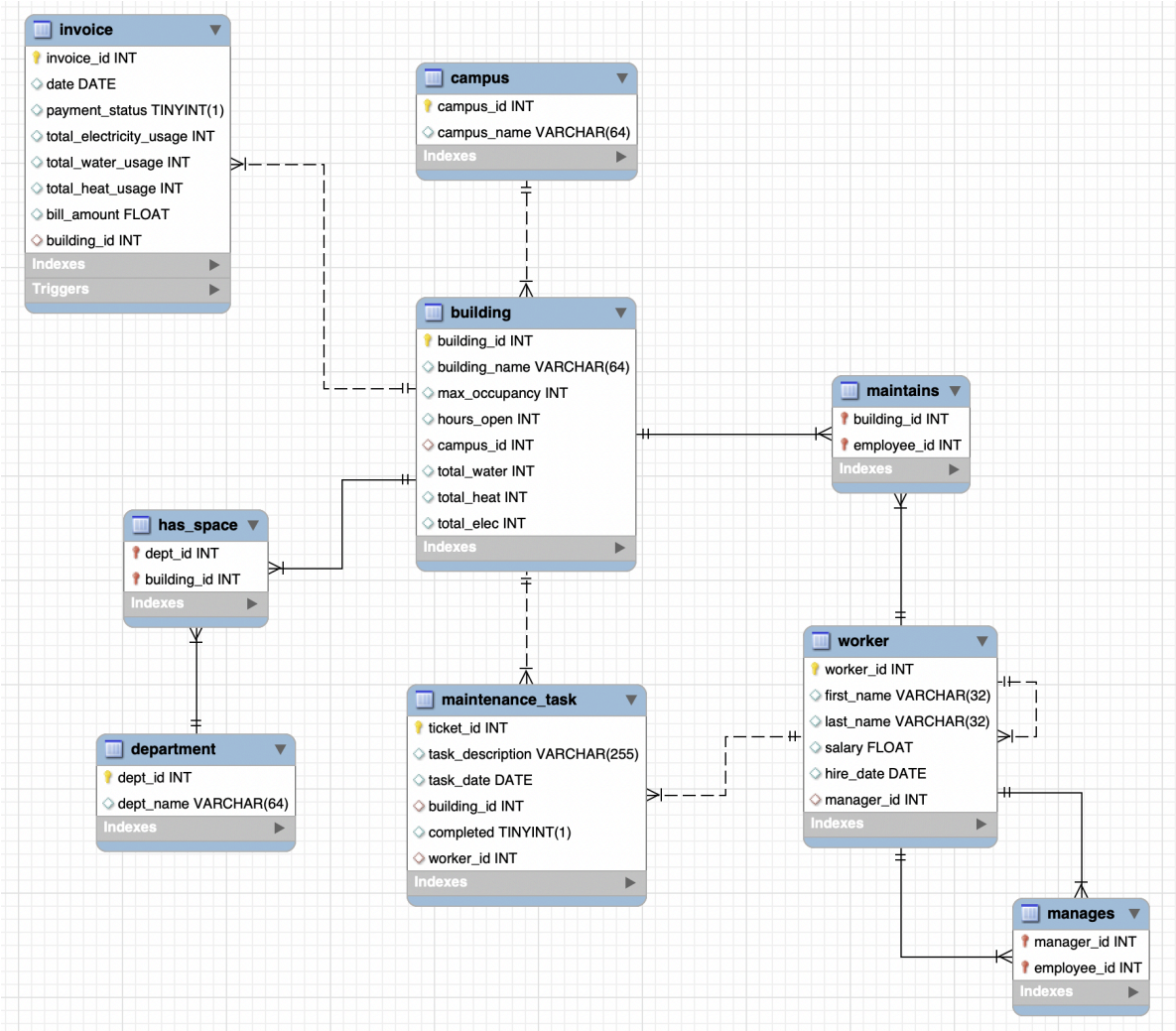
Foreign key relations exist between relevant entities in our model. For example a building is linked to a campus as buildings can be identified to exist on a particular campus. Similarly, a maintenance ticket is assigned to a maintenance worker and contains a task for a specific building. As a result, the ticket entity maintains a foreign key relationship between both the worker and the building tables. Utility invoices are also building specific and consequently maintain a foreign key relationship with a building as well. Finally, we also allow for maintenance workers to be able to take on two different roles: a normal maintenance worker or a manager. This recursive relation allows for maintenance workers to be able to manage other maintenance workers; however, having this managerial role is not required.

UML Diagram



Logical Design

The logical design of our model extends the conceptual model to include a full Entity-Relationship model diagram. For our project we chose to use a relational database model with a SQL model rather than a NoSQL one. This was primarily motivated by our particular use case. Given that we were modeling various entities across a university campus with clearly defined attributes and relationships, it made the most sense for us to implement this with a simple relational model.



User Flow

The user flow links together all of the parts of our database application. When the application is launched a sidebar and a view appear on the screen in a web browser. A user interacts with different views (pages) of the application through the sidebar. The sidebar allows a user to view different functions of the app.

Core database application functionality:

- Full building view (display all the buildings)
    - This allows a user to query the database to see a list of all of the buildings on a campus in a neatly formatted table view
- Campus view
    - This allows a user to view a building's utility usage as a graph for all of the buildings on a particular campus
- Building detail view
    - This allows the user to view details regarding a single building (specified by the user) from the database including maintenance workers assigned to that building
- Create a building
    - Allows a user to create a new building
    - When a user creates a new building it triggers a database insert into the has space table to keep track of building and department relationships
- Delete a building
    - Allows a user to delete a building
- View all of the invoices (view all of the invoices listed in a table)
    - Allows a user to view a table view of invoices
- Update invoice
    - Allows a user to alter the payment status of an invoice (if a invoice has been paid or its payment status is unpaid this value can be changed)
- Create an invoice
    - This allows a user to create a new instance of an invoice
    - When a new invoice is added it triggers the database to update the utility fields in the building table for the associated building
- Delete an invoice
    - Allows a user to delete a specific invoice from the invoice table

Additional information about functionality can be found in the mySQL database dump file.

Lessons Learned

There were many lessons that we learned about database design, construction, and implementation from working on this project. Many of the core lessons learned revolved around things like conceptual design and implementation decisions.

Another lesson we learned is that the overall schema (conceptual design/logical design) sometimes changes as you progress through a project. This is likely similar to real-world industry contexts where meetings with stakeholders may eventually change certain aspects of a project. We found that while we started out with an idea of which attributes would make the most sense for certain entities as we began building things we sometimes came up with new ideas about key attributes which should be included in our model.

This reflects the fact that at the end of the day some aspects of the design are dynamic and always changing as new information is added or new use cases/functionality implemented. The initial planning phase was incredibly valuable, but as developers of the system we need to be willing to make changes and remain flexible.

Technical Expertise Gained

Both of us gained a significant amount of experience with full application development after completing this project. Going into it, we had a general sense of what was required for building out a web application connected to a database, but greatly expanded our technical capabilities through this project. Specifically, we gained significant technical expertise with using the Python mySQL connector, Flask, and React. While we both built significant technical experience with SQL during class, this project really allowed us to master some of the core concepts covered in the second half of the course such as functions, triggers, and procedures.

Using a real-world project allowed us to think deeply about what the tradeoffs existed for different designs/architectures and to build out technical experience with working with APIs and their development. By the end of the project we both became very comfortable with picking up new tools quickly and building out/testing a full database application.

Insights - Time Management/Data Domain

Designing the schema gave us significant insight into the domain of the data used for this project. Thinking about integrity constraints, triggers, and the relationships between entities brought new understandings about the interactions/connections between/across different data items in our database. For example, when examining invoices and utility usage inside of buildings, there was extensive discussion surrounding what the domains of each of these attributes were and what the trade offs for different design decisions would be. Selecting different domains would essentially alter the meaning of these values to the end user - is electricity measured in kilowatt hours used, cost of electricity used for a building, etc.

Ultimately, we opted to select units specific to each utility rather than using a general “cost” for each as this would have led to a less rich representation of the data and would have been less relevant as an insight for visualization etc. for an end user who may be interacting with our application.

During the construction of this project we gained some valuable insights regarding time management. During the initial ideation phase our scope was slightly larger at the beginning. As we began working on building out different parts of the project we scaled back some small parts of the project to simplify them. We also made sure to make use of functions to speed up the development process especially when building out the API so that we were not repeating code that was being reused multiple times.

Contemplated Alternative Design/Approaches to the Project

There were a few alternative design choices that we discussed when initially deciding how best to construct the project. During the earliest discussions, we examined and contemplated the differences between implementing a simpler command line approach (similar to the homework assignment from HW 8) versus building out a more complete (albeit slightly more complex) web application/GUI interface. We ultimately decided to go with the GUI interface as this would allow us to provide a more intuitive way to interact with our database application especially for non-technical audiences.

Given the fact that our database application is specifically designed to be used by university administrators, maintenance workers, and department leaders, which may or may not have deep technical expertise or knowledge, we opted to go with a more extensive graphical user interface for interacting with our application. This expands the aperture for the amount/types of users who can interact and glean insights from our application.

During the build stage of the application development – for the server specifically - we contemplated using two different programming languages for actually building the server side code for our application. At first, we started building out some of the API functionality using Java so that we could leverage the JDBC mySQL connector. This raised some interesting challenges as some of the API construction methods were less familiar to us in this language (establishing the API functions, managing the JSON payloads/parsing, etc.).

After building out a small portion of the server using Java, we opted to switch to Python instead. This allowed us to speed up our development time and build out a full API much faster. We were also able to leverage built in functionality from Python's Flask to help manage payload sending between our front end and back end server.

Non-functional Code

Although there are some areas of our application which provide fertile ground for further work/exploration all of the code that is provided in our application on github is functional. Within our SQL schema, there are a few procedures which were written for testing, exploration, and for completeness which are not available directly to the user.

Future Work

Planned usage of the database

The database that we constructed can be used by university administrators to track the utility usage of water, electricity, and heat for individual buildings on a particular campus. Further, it provides an easy way for administrators to manage utility bills and invoices for facilities on campus.

Potential areas of added functionality

While our existing project contains a significant amount of functionality, there are other areas where it might be expanded in the future. One potential area of future expansion is related to our maintenance worker tracking data inside of our database. Some examples of some additional functionality that could be added in the future is to expand our application to have more features related to managing workers. For example, we could add update functionality to manage reassignment of workers to tasks. Another idea could be to expand the update features for tickets to include status updates for in progress tasks (to more closely reflect ticket platforms like those used in the IT department at Northeastern). While our application was mainly focused on utilities tracking, there is room in the future to build out the ticket management aspect more.

Future uses/work

A university-wide view into data for all of the buildings across multiple different campuses has increasing relevance for universities like Northeastern as they continue to expand to have a global presence. Being able to have a single view into all of the relevant metrics for an entire university can help university administrators better track and make decisions regarding key metrics regarding utility usage across a campus. Future work may involve continuing to build out/expand this application to incorporate more and more functionality for university managers.

Please use your project proposal report as a starting point to create your project's final report. Provide a single document that contains the following sections:

1. Provide a README section for creating and running the project. We need complete specifications for building your project on our computer. Specify all libraries, software, etc. needed to run the application. Specify expected installation directories. If you use a specific technology for the project, the technology's download page must be listed.
2. Provide the Technical Specifications for the project. This is entails the software used to build the project such as the host language and any other frameworks used.
3. Provide the current conceptual design as a UML diagram for the project .
4. Provide a logical design for the submitted database schema (Feel free to Reverse Engineer your final schema in the MySQL workbench). If you are submitting a Mongo database, please provide a description of your collections.
5. Provide the final user flow of the system. List the commands or methods the user performs to interact with the application.
6. Provide a "Lessons Learned" section that contains report sections for the following:
    1. Technical expertise gained
    2. Insights, time management insights, data domain insights etc.
    3. Realized or contemplated alternative design / approaches to the project
    4. Document any code not working in this section
7. Provide a "Future work" section containing:
    1. Planned uses of the database
    2. Potential areas for added functionality
    3. No future uses or work can be documented if justification is provided.

The final report file should be name
d canvas_group_name_final_report.pdf, where canvas_group_name is your group name in canvas. Remember, this is a writing exercise. Please take the time to write a cohesive report on your semester's project.