Angelica Kim
Clara Hoey
Mia Jung
Tae Lee

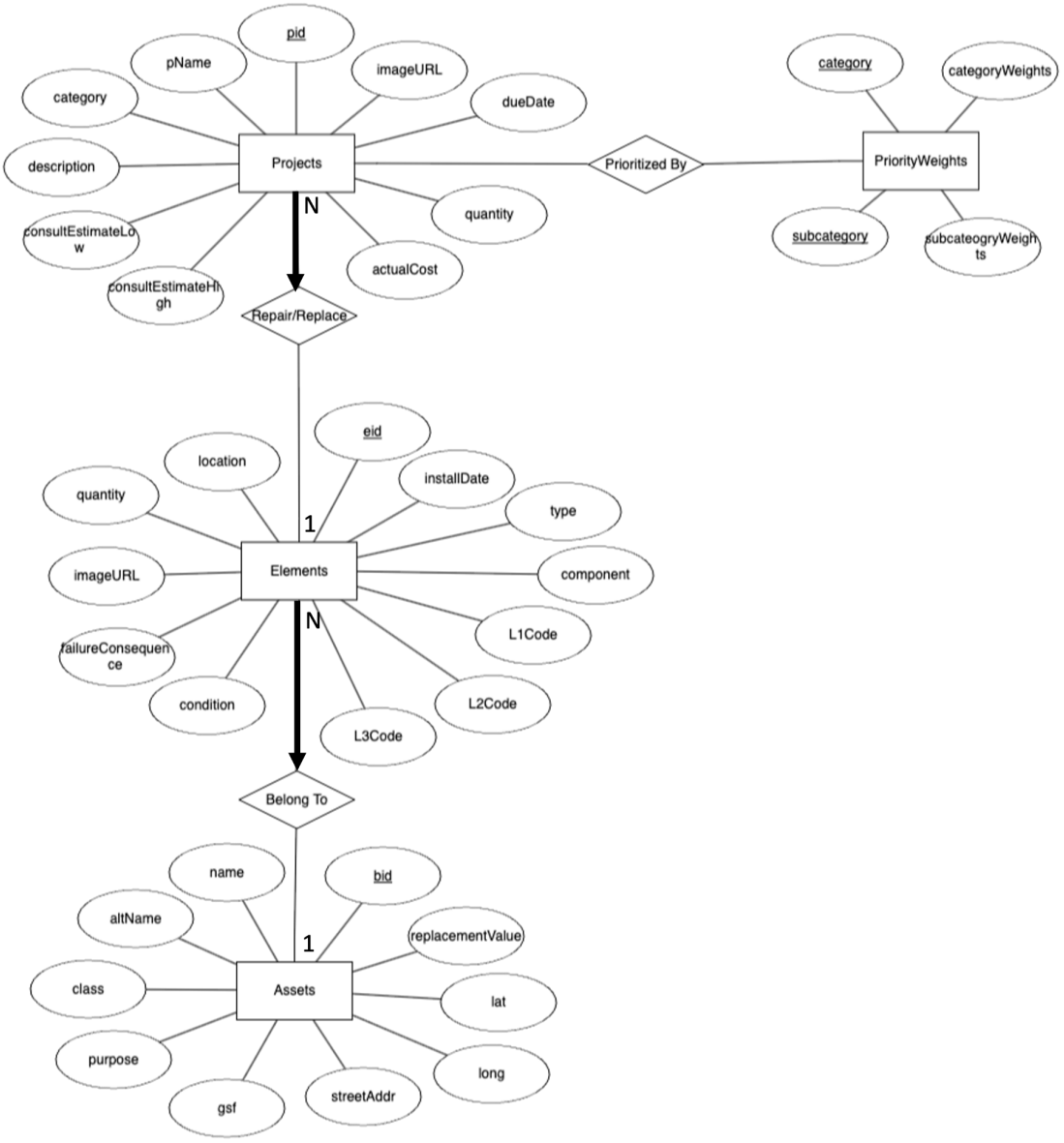
Facilities Team Project Phase 2 Document: Revised Project Plan and Schema

Revised project idea:

- Project Description:
    - Our team's project is dedicated to reorganizing, revamping and revisualizing the various tables of data that the Amherst College Facilities uses. The fundamental purpose of the data is to keep track of building and facility conditions in order to determine which facilities are at a need to be fixed or replaced. The efficient management of this data and implementing a method to calculate the priority of facilities that require repair is critical, as it involves billions of dollars of our college's money being channeled into these facilities projects.
- Interactions that the users will be able to have with the database:
    - Users can select one or multiple buildings at a time from a map of the entire campus and see their facility index scores (FCI).
    - Users can provide numerical priority weights for different criteria (i.e. condition, type of building, failure consequence) and then the database will calculate an overall priority score for each building.
    - Users can input potential amounts of funding (e.g. $10 million) to see how this will affect the facilities management in coming years — for instance, allocating more funding sooner might reduce the total amount of money spent over the next 20 years depending on how much buildings deteriorate and/or projected market conditions.
- Current state of the data:
    - The original data is stored in Google Tables, and the client is using Google Studios to calculate specific metrics because Google Tables does not support calculated fields.
    - The client also provided a preliminary data mapping. From this data mapping, we were able to see that the key information about the facilities falls into the following categories:
        - Assets: Describes information on buildings on campus
        - Elements: Describes metrics for status of buildings (e.g. condition, failure consequence)
        - Requirements: Describes projects at the element level
        - Cost-related data and formulas to calculate priority scores for projects
- Deliverables:

    - Database:
        - Right now, we're planning on having the main tables in the database be Projects, Assets, Elements, and several tables which will be referenced to calculate priority scores.
            - Projects table: contains information about repairing/replacing specific buildings or parts of buildings
            - Assets table: contains information about the buildings on campus (e.g. name, address, replacement value)
            - Elements table: contains information about parts of buildings (e.g. material type, unit type)
            - Several other tables will hold enumerated weights for different categories and subcategories of buildings, building conditions, and failure consequences.
    - Graphics: **client is still thinking about what will be most useful, so this are his preliminary ideas**
        - Map of the entire campus where user can select several buildings at a time to see their FCI (facility condition index)
        - Bar chart showing the projected total replacement values ($) of buildings for the entire campus for the next 20 years, with an overlaid dotted line of FCI and the option for the user to input different amounts of funding to see projected changes
- Any other detail that will help you actually completing the project:
    - We will need more details and information from our client about the graphic displays, but this can be discussed later once he decides what will be most useful. We also will work on finding a way to store photographs and longer text containing notes on the individual building elements within our database.

Refined ER diagram



Relational schema

CREATE TABLE **Projects**(
pid VARCHAR PRIMARY KEY,
--asset name abbreviation + auto-incremented row number by asset (e.g. COHA-001); create it using a TRIGGER and row_number() over(partition by aid) in the function so that pid is automatically created when the user inserts a new project
pName VARCHAR, --descriptive name of a project
bid CHAR(5) NOT NULL REFERENCES Assets ON DELETE CASCADE,
--asset name abbreviation; if an asset is deleted from the Assets table, delete all the projects overseeing that element
eid INTEGER NOT NULL REFERENCES Elements ON DELETE CASCADE
--element ID for the element being overseen by the project; if an element is deleted from the Elements table, delete all the projects overseeing that element
L1Code VARCHAR, --Uniformat Level 1 code of the element being overseen by the project
L2Code VARCHAR, --Uniformat Level 2 code of the element being overseen by the project
L3Code VARCHAR, --Uniformat Level 2 code of the element being overseen by the project
category VARCHAR, --category of the project e.g. maintenance, HazMat, sustainability
description VARCHAR, --more detailed description of a project
relatedProjects VARCHAR REFERENCES Projects,
--pid of the related projects; used foreign key to self-reference
complete BOOLEAN,
inProgress BOOLEAN
renewal BOOLEAN,
inspectedDate DATE,
dueDate DATE,
priorityScore DECIMAL,
--calculated by joining the weights table upon creation; using a TRIGGER will keep the values automatically updated even when the values in the weights table change
baseCost DECIMAL CHECK(baseCost > 0),
--calculated using a formula (Projects.quantity * Elements.unitCost); could calculate on the fly, but the client wanted it in the table
--use a TRIGGER to keep it up-to-date even when unit cost of an element changes
replacementFactor DECIMAL,
softCostFactor DECIMAL,
difficultyFactor DECIMAL,
consultEstimateLow DECIMAL,
--lower bound of the project cost estimated by consultants

```
    consultEstimateHigh DECIMAL,
        --upper bound of the project cost estimated by consultants
    calculatedCost DECIMAL,
        --project cost calculated using a formula
        (baseCost*replacementFactor*softCostFactor*difficultyFactor) when the
        consultant estimate is unavailable; using a TRIGGER will keep the values
        automatically updated even when the values in the weights table change
    totalQuantity DECIMAL, --total quantity of the element (from the Elements relation)
    quantity DECIMAL CHECK(quantity <= totalQuantity), --quantity of the element that
    require repair/replacement
    completedDate DATE,
    actualCost DECIMAL,
    actualCostNotes VARCHAR,
    imageURL VARCHAR
);

CREATE TABLE Elements(
    eid SERIAL PRIMARY KEY,
    assetName VARCHAR NOT NULL REFERENCES Assets (name) ON DELETE
    CASCADE,
        --an element must belong to an asset; if an asset is deleted, all the elements in
        the asset should also be deleted
    L1Code VARCHAR,
    L2Code VARCHAR,
    L3Code VARCHAR,
    location VARCHAR, --e.g. Building Exterior, Restrooms
    type VARCHAR, --type of an element e.g. PVC roof, TPO roof
    component VARCHAR,
    eul DECIMAL, – estimated useful life (yrs); same type of element has same EUL; used
    to calculate remaining useful life (RUL)
    unit VARCHAR,
    unitCost DECIMAL,
    quantity DECIMAL,
    installDate DATE, – used to calculate remaining useful life (RUL)
    condition VARCHAR, --e.g. Excellent, Failed
    failureConsequence, --e.g. Nuisance, Full Building Shutdown
    imageURL VARCHAR,
    UNIQUE(assetName, type, component)
);

CREATE TABLE Assets(
    bid CHAR(5) PRIMARY KEY, --abbreviation of the asset name (e.g. COHA)
    name VARCHAR UNIQUE,
    altName VARCHAR –alternative name if the asset has been renamed
```

```
        class VARCHAR –e.g. Building, Court, Road, etc.
        purpose VARCHAR(25) – e.g. Academic, Admin, Athletic, etc.
        streetAddr VARCHAR,
        gsf DECIMAL, --gross square feet
        replacementValue DECIMAL,
                -- estimated cost, at current prices as of the effective appraisal date, to construct
                a substitute asset that matches the current condition of the existing structure
        fci DECIMAL,
                --facility condition index (%) = sum(Projects.consultEstimateHigh or
                Projects.calculatedCost) + sum(Projects.baseCost) grouped by asset /
                replacementValue; client wanted to see FCI in the database, instead of
                calculating it on the fly
                --using a trigger would keep FCI updated even when cost information changes in
                the Projects table
        latitude DECIMAL,
        longitude DECIMAL
);

CREATE TABLE PriorityWeights(
        category VARCHAR,
        categoryWeight DECIMAL,
        subcategory VARCHAR,
        subcategoryWeight DECIMAL,
        PRIMARY KEY(category, subcategory)
);
```

List (and short description) of the software you installed and configured:

- PostgreSQL (version 13.11): we got PostgreSQL up and running on our Linux server and completed the setup of our project database.
- pgAdmin4: Following Andy Anderson's slides, we also installed pgAdmin4, a PostgreSQL-specific GUI, locally on our computers and connected it to our database.
- DBeaver Community (version 23.2.2): While we could interact with PostgreSQL through pgAdmin, we found DBeaver to be more user-friendly as an IDE. We installed it locally on our computers and added connections to our database.
- Tableau Desktop (version 2023.2): We discovered that Tableau offers a 1-year trial for student accounts validated through school credentials, so we would use Tableau for our deliverables as our original plan. We successfully connected Tableau Desktop to our PostgreSQL project database.
- Due to memory constraints, we opted not to install Tableau server on our Linux server. We proactively requested additional memory from Andy Anderson and received an allocation of 15GB to our VM disk, bringing our total to 22GB, but this falls short of 64GB required by Tableau server. We chose not to request further memory because it is still possible to deploy a dashboard from Tableau Desktop connected to our database.

Role Assignments for Phase 3:

After our group discussion, we have decided that a collaborative approach will be more effective than assigning specific roles for the next phase. This approach allows all team members to remain informed and engaged across all facets of the project.

Embracing collaboration throughout Phase 3 will also foster a valuable exchange of expertise and insights among our team members, enabling us to learn from each other's unique strengths and knowledge bases.

Nevertheless, if we had to assign roles for each aspect of Phase 3, we would designate roles for people to *lead* discussions and ensure the successful completion of each component within this phase. Below is our proposed leadership role assignments for this project phase:

- Completing the conversion of the ER schema into the relational schema
  → All members take the lead, participating in the meeting with Matteo during which we will ask our questions from phase 2 and get further feedback / clarification about our schema.

- Creating and populating the database and the tables
  → Mia takes the lead.

- Devising example SQL queries that showcase the different ways that the user would interact with our database:
  → Angelica takes the lead

- Starting to implementing the app or website, to the point that it can perform the example SQL queries:
  → Tae takes the lead

- Preparing for the 10-minutes in-class "midterm" presentation on our project (which should include, if possible, a small demo about which we will be receiving feedback from the other groups)
  → Tae takes the lead

- Curating list of open questions about our project that we plan to answer in phase 4:
  → Clara takes the lead

- Creating a document containing the final relational schema, making it available in the GitHub repository, submitting it via email to Matteo
  → Clara takes the lead

Open questions:

Here are the key questions that we have moving forward to the next phase:

1. Is it a good approach to have fields that are involved in the calculation of another fields as foreign keys? How else could we ensure that a calculated field stays updated when fields that are needed for its calculation are changed? Could we use the TRIGGER statement? Could we use the trigger statement if the fields involved in the calculation function are not from the same table?

   e.g. Should we compute the calculatedCost of a project on the fly, or should we use the TRIGGER statement so that we can keep the calculatedCost a column in the projects table, and still keep it updated whenever the fields involved are changed?

2. For now, we thought of making a name/weight table for all the enumerated information. This is mainly because we want category and subcategory weights in the elements and projects table, but keeping all of the category and subcategory weights (and their "names") made the tables massive. Are creating all these lookup tables for weights a good approach? If we can use the TRIGGER statement even when the fields involved in the calculation functions are from other tables?

3. What is the best way to store pictures in the database? For now, we thought of storing links to the google drive file (or folder) that contains the pictures. Is there a better way to store images in the DB aside from links?

4. Must we install the Tableau server on the linux server, or could we just run it from the Tableau desktop?