Database Project Proposal:

1. **What is the goal of the database? How will this database be used?**

Right now, information is mainly being stored across multiple Excel files, at different locations inside our shared J drive. There are several problems with this. These files or the information inside of them can be accidentally deleted; inaccurate edits and inconsistencies may go unnoticed especially when we have multiple copies of the same file like the Equipment Status Excel file where there's a file for each day; information can be hard to find across hundreds of folders; updating these information can be time-consuming with the different file locations and the different formats across files; and some useful information may not be available or be up to date like equipment specifications.

The goal of this database is to fix these problems and enable additional helpful features that help with organization and information retrieval. Instead of using easily manipulable Excel files, we will store information in a database (could be MySQL, Microsoft SQL …) that allows us to retrieve the information we desire through queries - or instructions on what data to retrieve. We will use this database through a web application that is linked to this database. Data entry, updates, deletion will be supported by user input fields in the web application. The web application will be an all-in-one tool allowing various information updates, which, can be further coded to warn certain actions, allow for only specific data types and/or values, to ensure consistency across our data.

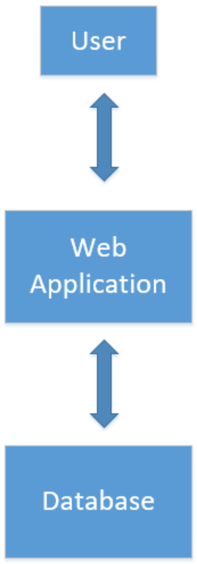

Figure 1 - Interaction schema

2. **How is information stored and used in our database (Why was the database designed this way)? How are JOINS useful?**

A database is made up of tables, the columns to these tables are the variables that we want to store. Each row of a table represents a set of data that we entered into the database, in this table specifically. I have created many tables in this initial database design. Some of these tables are to store information solely about a specific entity in our testing process - product type, programs, test maps, DARs, chambers, etc. Other tables are designed to achieve a certain feature that we want to implement in our web application or in data-presentation.

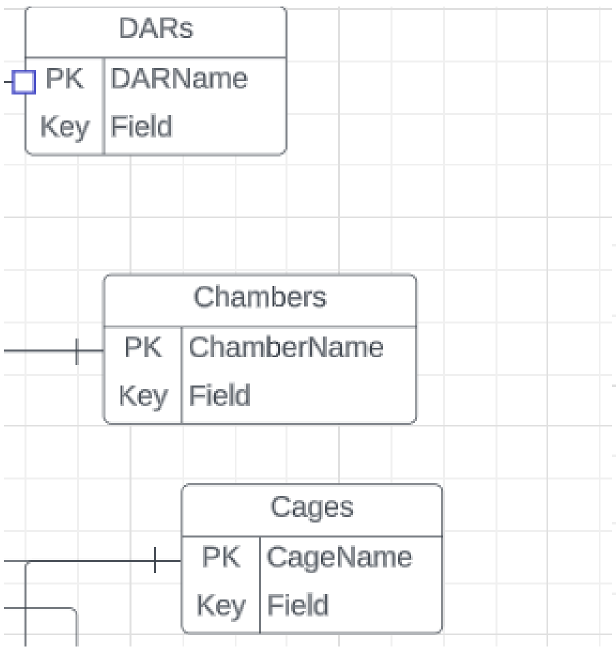

Figure 2 - Entity tables

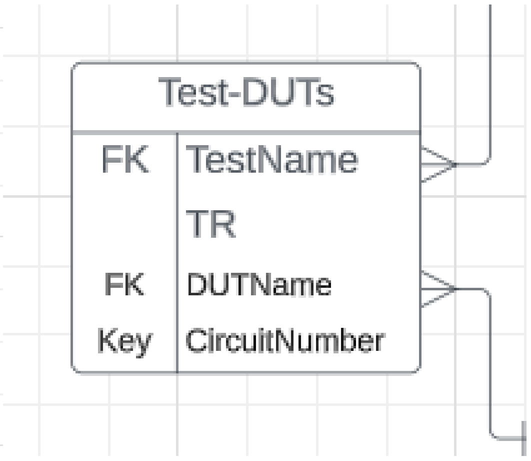

Figure 3 - A feature-specific table

The information within entity tables are specifications and details about these entities that we might want to be able to look up at a later time (e.g. power source of a chamber). Table columns can be appended to other tables by having a unique identifier column that is used to compare to a column with potentially matching values. In the example below, the column ID of Table 2 is used to compare with column ID of Table 1. The matching case of ID 1 is appended with a value of "red", the non-matching case of ID 2 does not get a value appended to it.

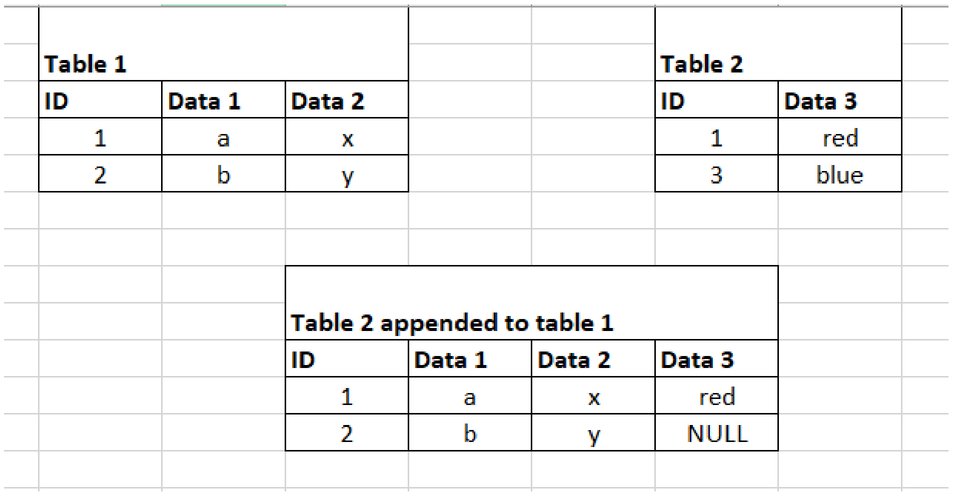

Table 1

ID	Data 1	Data 2
1	a	x
2	b	y

Table 2

ID	Data 3
1	red
3	blue

Table 2 appended to table 1

ID	Data 1	Data 2	Data 3
1	a	x	red
2	b	y	NULL

Figure 4 - Appending tables

Appending tables (or JOINING tables – an SQL database term) is how we can retrieve a specific set of data that we want, without having to create customized tables and potentially having redundant data.

Feature-specific tables simply reference one or more other tables to allow for retrieval of a set of data that might be difficult to obtain by only combining entity tables. We can also select which columns are fetched from our database.

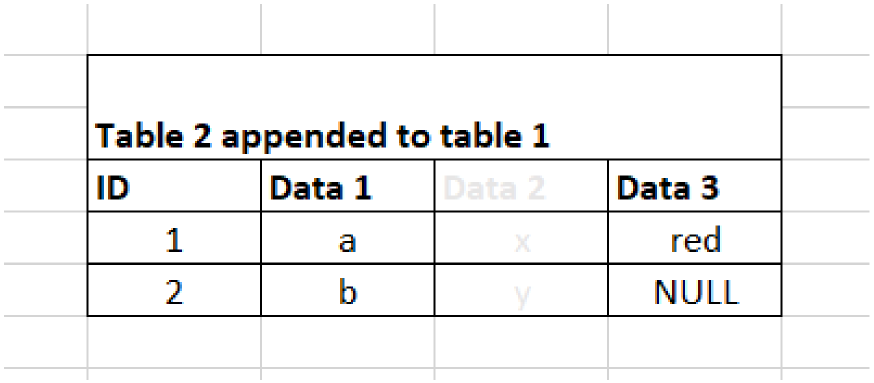

Table 2 appended to table 1

ID	Data 1	Data 2	Data 3
1	a	x	red
2	b	y	NULL

Figure 5 - The previous table, but only showing the first and last column

Using this concept, we can create many tables to store different data of our testing process and equipment that we can later download and combine to view on our web application. The current database design has a table or each entity in our testing process and some other tables to allow for features that will be detailed later.

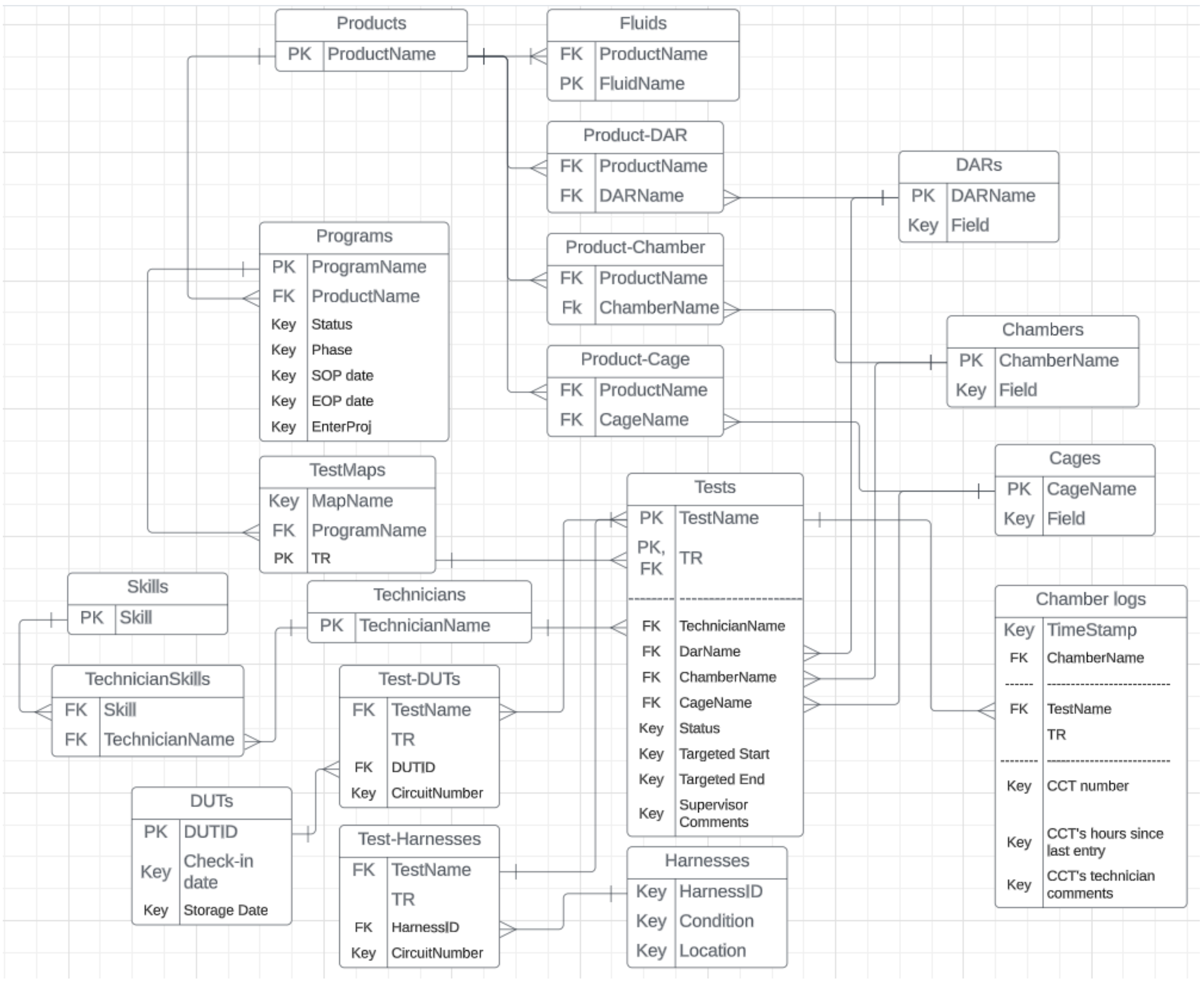

Figure 6 - Current database design

The table columns are referenced to each other by relationship lines. We could say the relationship is one-to-one, which means each row of a table (comprised of however many columns of that table we might want to append) is referred to only by one other row of the other table. A one-to-many relationships is when a row of a table is referred to by many rows of another table, we can append the same information to these different rows for example. A many-to-many relationship is represented by tables like the "Test-DUTs" table, which we called feature-specific tables, can refer to as many combinations of different rows of other tables as we want. Each combination would result in a new row, or entry, in the feature-specific table. A row of one table can be appended to many rows of a different table, but that same row can have many versions where different rows of the other table is appended to it.

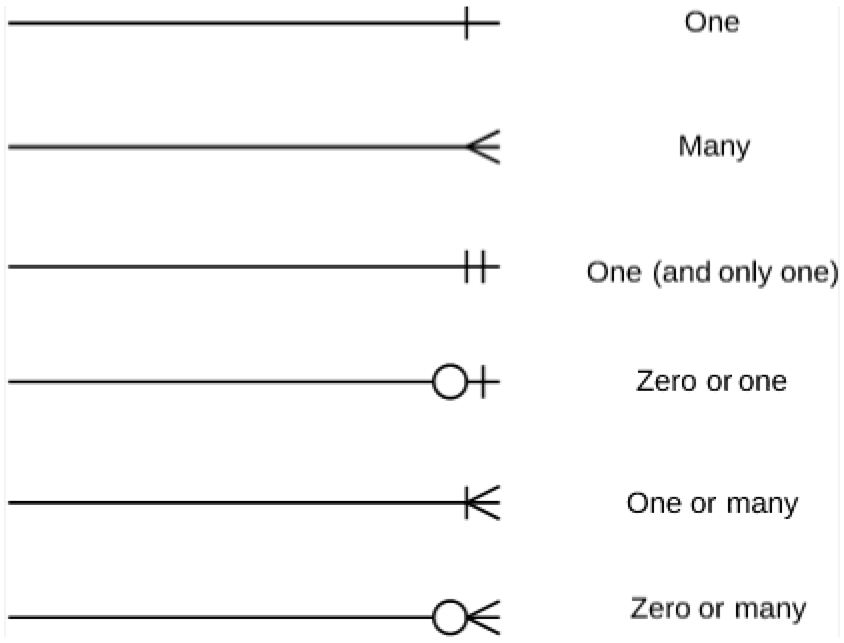

Figure 7 - Relationship line symbols

In our database, the primary relationship is one-to-many. The tables on the 'one' side can be appended to the 'many' side tables. This means that we can keep appending tables, as long as the table we append is on the 'one' side of one of the tables we have combined. For example, take the "Tests" table, we can append to this the "TestMaps" table, then append to this chain of tables the "Programs" table, then the "Products" table. This way, we can create a super table that we may actually want, or pull out only specific details within this super table through our queries. Like knowing what the product type a test with some TR is for. Having different tables, and appending and filtering them this way eliminates the need to create large tables where we may have to state the product type of multiple tests under the same TR for example. This helps save memory space in our database while still allow us to get the same information that we want.

3. **What are the features that we can implement with this database design that can help with our work?**

The sets of data that we want to be able to get in the end dictates how we link tables with each other, and how many tables we need. This current database design is based on several data sets and features of the web application. We will list them out here and show how they are achieved.

- Know what equipment are being used

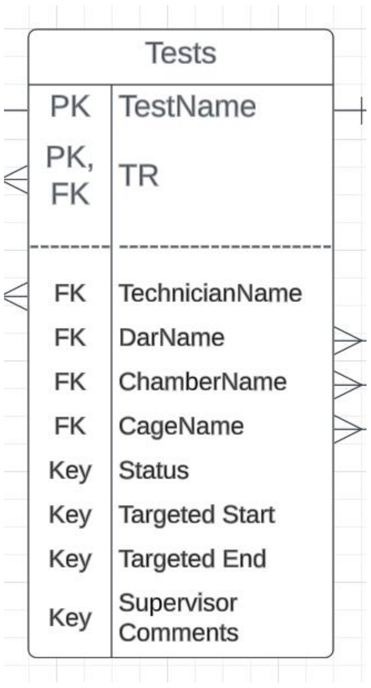

Figure 8 - "Tests" table

We will have a “Tests” table that will list all the current, upcoming, and finished tests; along with the equipment assigned to them. To know the equipment that are being used, we will search for only the current tests and we will be able to see all the equipment currently assigned. We can go more detailed like searching only for DARs, or Chambers.

- Know where equipment are

Using the same “Tests” table, we can search for current tests, search for a specific DAR or cage, and see the chamber at which it is located.

- Know when equipment are available

Say we want to plan assignment of equipment to future tests. Using the same “Tests” table, we have a “Targeted Start” and a “Targeted End” date, which we will use display when an equipment is unavailable.

- Know the hours that a chamber has been used, the programs that used those hours, the test circuits that ran, for equipment utilization calculations.

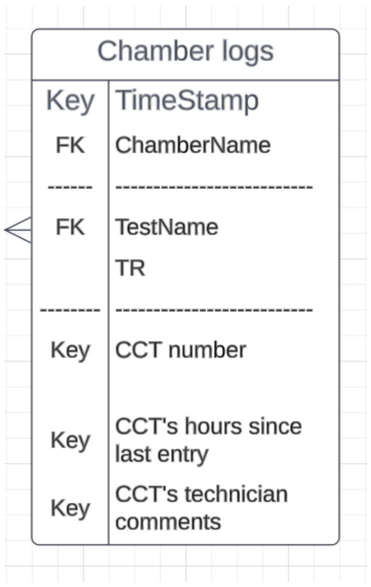

Figure 9 – "Chamber Logs" table

We will have a "Chambers Log" table that will record the hours of the circuits of a test along with a date and time stamp. Equipment hours can be calculated by getting the entries within a specific time period, and filtering out hours for calculations.

- Know the product type that each equipment is meant for

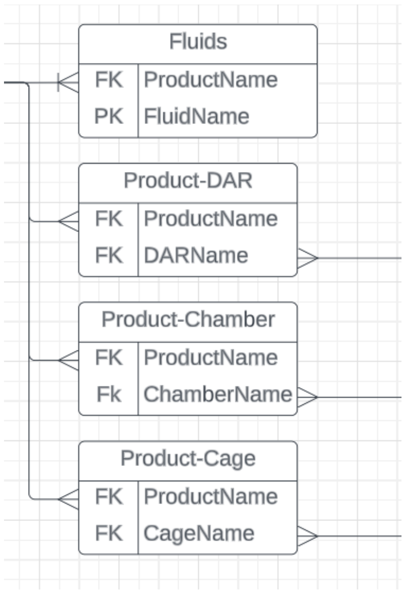

Figure 10 - Product-Equipment tables

We will have tables that pair each product type with the compatible equipment

- Know the capability of technicians

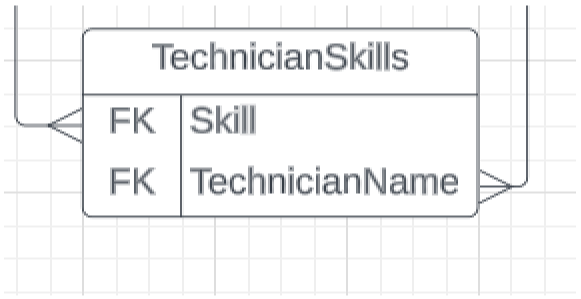

Figure 11 - "Technician Skills" table

We will have a table that pair each technician with the skills they have.

- Know the history of DUTs (what tests it has been through and which circuits it was in)

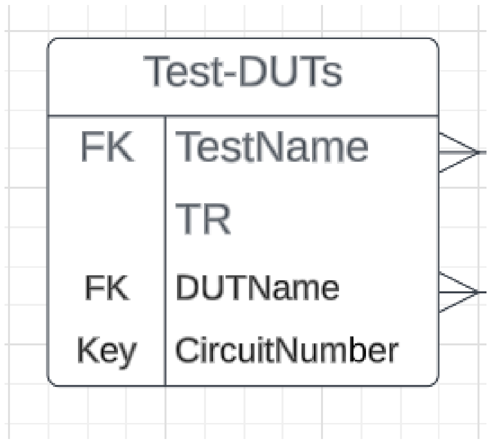

Figure 12 - "Tests-DUTs" table

We will have a table that pairs each DUT with the tests it has been through.

The web application will have the following additional features:

- Have usernames and passwords to the website
- Be able to add administrators to website. Administrators will be able to make administrative comments on tests, add tests, do things that regular users wouldn't be able to do.

4. Project Costs

Item	**Cost**
Harness Scanner	Unknown
Database Server	\$8.598/month + \$0.160(GB/month) (for 1 vCPU with 20% baseline performance, 1000 Inputs/Outputs per second – Azure MySQL)
Web Application Server	\$13.149/(month/site) (1GB Ram, 6 CPU hours/day – Azure App service)

5. **What is the plan and timeline going forward?**

The steps to completion for this project are as follows:

1) Determine the concurrency requirement of usage. Concurrency means the ability to read, write to the database at the same time; we want to determine what should happen if this situation occurs, what happens when two writes occur at the same time. (2 days)
2) Research the database management system suitable for application based on the concurrency requirement. It should be integrable with Python, and preferably have a management application. (3 days)
3) Create the database, set up links within the database, and fill the database tables with sample data. (2 weeks)
4) List the pages of the web application, how they link to each other, what should display on them, what features should be on them. (2 week)
5) Write the queries needed, test the queries. (1 week)
6) Write the pages of the web application, link them to each other, link them to the database, and test the web pages. (4 weeks)
7) Add log-in capability, user types. (2 weeks)
8) Format and improve aesthetics of pages. (2 weeks)
9) Find a host server or in-house server to run the web application. (1 week)

Total time: 15 weeks ≈ 3.5 months