INST733 Final Report: 4Maryland

Team 10

Srinidhi Nag

Tsion Zewolde

INTRODUCTION

4Maryland is the University of Maryland's (UMD) COVID-19 information and monitoring website. One of the site's primary functions is to track the spread of the Coronavirus through daily monitoring of both students and faculty. University affiliated members are required to report any experienced COVID-19 symptoms daily, as well as obtain COVID-19 tests every 14 days. Additionally, each individual must complete a virtual training about Coronavirus protocol and safety. Members who report being symptom-free for 14 consecutive days, submit a negative COVID-19 test, and have completed the required training are deemed to be in compliance with 4Maryland protocol and are permitted to visit the campus. University members may also report obtaining a COVID-19 vaccination, along with the vaccination dates and vaccination type. All members of the university are required to obtain a COVID-19 vaccination before the Fall 2021 semester. The 4Maryland site also provides opportunities for voluntary close contact reporting. If an individual comes to find that they have been in close contact with a person who has tested positive for COVID-19, they may report this exposure on the 4Maryland site.

The created database titled 4UMD serves to simulate the 4Maryland COVID-19 tracking system on a smaller scale. The schema stores information regarding member identification, vaccination, testing, visitation status, and compliance. The project design was planned with questions regarding the spread of COVID-19 at UMD in mind, such as:

- How many individuals are not cleared to come to campus?
- How many individuals are fully vaccinated?
- How many individuals were tested?
- How many of the tests were reported by the contact tracing?
- What were the reported symptoms?

These questions have been answered in the queries and views written, rather than through the database itself.

DATABASE DESIGN & IMPLEMENTATION

LOGICAL DESIGN

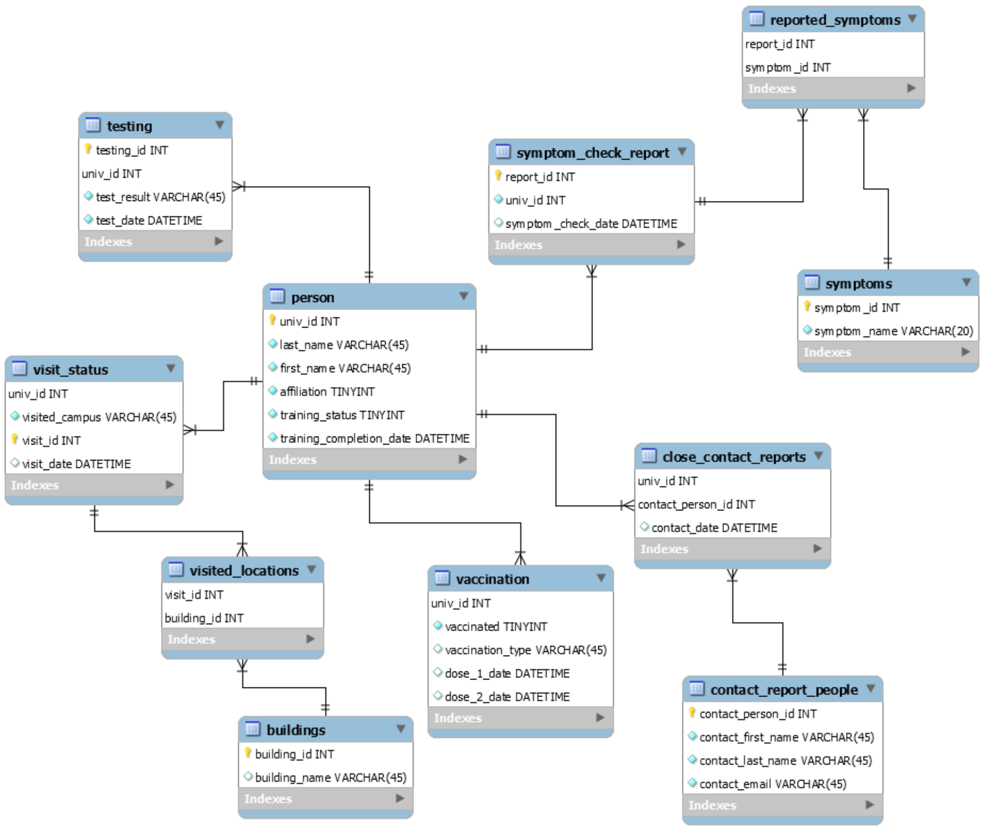


Figure 1. 4UMD Entity Relationship Diagram

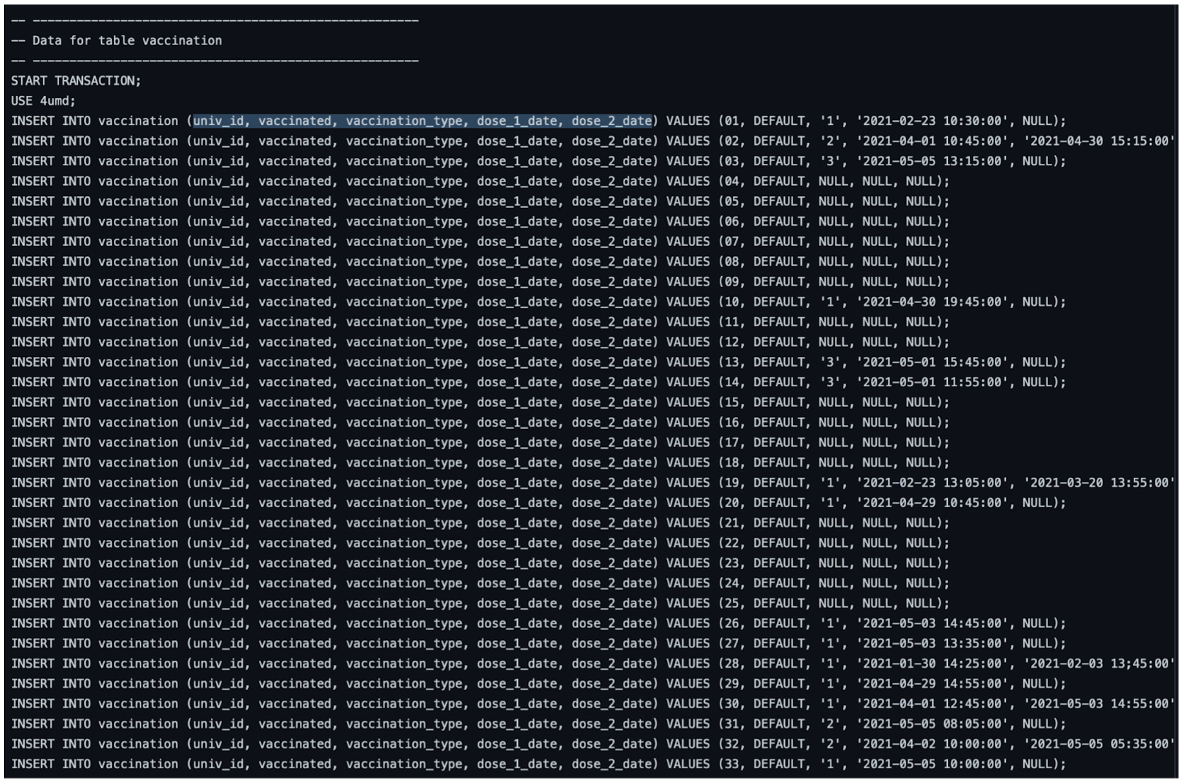
Database creation began with identification of required columns and tables such as person, univ_id, test, test_result, and visit_status. Once we identified the required entities, we organized them into tables and attempted to determine relationships between the tables. This was a process of trial and error, relying heavily on identifying existing mistakes in order to determine necessary columns and reevaluate relationships.

PHYSICAL DATABASE

Rather than writing SQL code to create the database tables, columns, and relationships, the database creation began on the MySQL Workbench interface. Once the tables, columns, and entity-relationship diagram (ERD) were finalized, the project was exported in SQL script form to be edited. Once it was confirmed that the script was running without errors, we proceeded to create sample data. 100 sample individuals were created with corresponding data. All characters are fictitious, as is the corresponding data. We did not attempt to obtain real data as it would be a violation of privacy, and likely would be unavailable to begin with.

SAMPLE DATA

Within the database we created 100 fictitious individuals for the sake of sample data. These individuals each had their own corresponding pieces of information such as first and last names, university ID, COVID-19 vaccination status, vaccination type, and vaccination dose dates. We came to the conclusion of creating 100 pieces of sample data with the intentions of creating a database that accurately represents the 4Maryland system, even if it is on a considerably smaller scale. A very large sample data set would be impractical in the amount of time given for the project, especially considering the effort required into developing the actual schema, and would greatly increase the amount of room for error. A very small dataset would be unrepresentative of the 4Maryland concept and would not serve to answer any of the questions this database was designed to address. This 100-person sample dataset provided a solid representation of how this database would run on a large scale, for instance if all university members were included. The sample allows us to experiment with the created database as the University of Maryland population would receive it.



```
-- -----------------------------------------------------
-- Data for table vaccination
-- -----------------------------------------------------
START TRANSACTION;
USE 4umd;
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (01, DEFAULT, '1', '2021-02-23 10:30:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (02, DEFAULT, '2', '2021-04-01 10:45:00', '2021-04-30 15:15:00'
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (03, DEFAULT, '3', '2021-05-05 13:15:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (04, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (05, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (06, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (07, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (08, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (09, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (10, DEFAULT, '1', '2021-04-30 19:45:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (11, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (12, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (13, DEFAULT, '3', '2021-05-01 15:45:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (14, DEFAULT, '3', '2021-05-01 11:55:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (15, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (16, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (17, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (18, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (19, DEFAULT, '1', '2021-02-23 13:05:00', '2021-03-20 13:55:00'
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (20, DEFAULT, '1', '2021-04-29 10:45:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (21, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (22, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (23, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (24, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (25, DEFAULT, NULL, NULL, NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (26, DEFAULT, '1', '2021-05-03 14:45:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (27, DEFAULT, '1', '2021-05-03 13:35:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (28, DEFAULT, '1', '2021-01-30 14:25:00', '2021-02-03 13;45:00'
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (29, DEFAULT, '1', '2021-04-29 14:55:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (30, DEFAULT, '1', '2021-04-01 12:45:00', '2021-05-03 14:55:00'
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (31, DEFAULT, '2', '2021-05-05 08:05:00', NULL);
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (32, DEFAULT, '2', '2021-04-02 10:00:00', '2021-05-05 05:35:00'
INSERT INTO vaccination (univ_id, vaccinated, vaccination_type, dose_1_date, dose_2_date) VALUES (33, DEFAULT, '1', '2021-05-05 10:00:00', NULL);
```

Figure 2. 4UMD schema sample data, entered through INSERT statements

VIEWS & QUERIES

The views allowed for data optimization as well as query storage to answer our predetermined questions. The views served to utilize the data in all 11 tables and allowed us to come to conclusions based on the data and its relationships, even if changes were made. These views were created in correlation to the ERD and queries, when viewing one table it then correlates with the other tables in connection. The information correlates and transcends, permitting data manipulation to answer the predetermined questions set for the database. Answering these questions was the primary focus behind query writing.

While the project guidelines only required five views for our 11-table database, we created 10 working views and queries in order to display the database's functionality. These views and queries serve to do the following:

- Identify individuals who are/are not in compliance with 4Maryland protocol
- Identify individuals who visited campus when out of compliance
- Determine the percent of individuals still not vaccinated or not fully vaccinated
- Return a list of all fully vaccinated individuals, along with their corresponding vaccination types
- Return a list of close contacts
- Return a list of individual who tested positive for COVID-19
- Return a list of all visited buildings

CHANGES FROM ORIGINAL DESIGN

The original project design consisted of only five tables. This was largely due to inexperience and lack of understanding regarding database design. It was thought that the five simple tables we decided upon could be linked to each other and a functioning database would be formed. We failed to recognize that intermediary tables would often be necessary to identify relationships, that not all relationships would be recognized my MySQL Workbench simply because we attempted to manually form them, and that many different attributes (e.g. campus visits) need their own IDs in order to be properly logged and linked to their corresponding columns.

ISSUES EXPERIENCED DURING DEVELOPMENT

ISSUES ENCOUNTERED

Throughout database design, we relied on the ERD to make sense of data flow and as a resource for query writing. However, a repeated issue encountered was the inconsistency in key symbols on the ERD in MySQL Workbench. Confusion arose through some foreign and primary key icons not being present in the ERD, even when the MySQL Workbench schema identified a specific column as a foreign key or primary key. For instance, in figure 1, visited_locations.visit_id does not have the primary key icon next to the column name. However, figure 3 below shows that this column has been identified as a composite primary key.

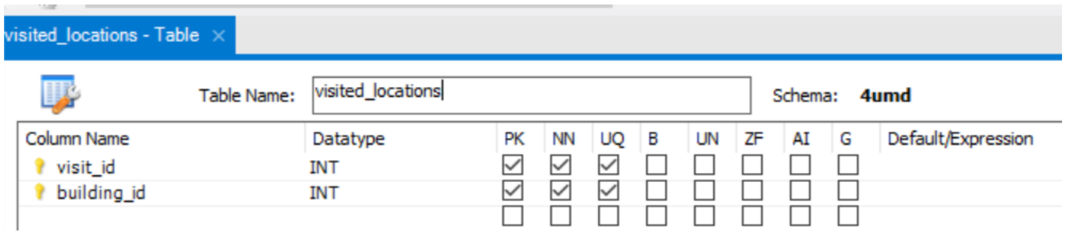

visited_locations - Table

Table Name: visited_locations

Schema: 4umd

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
visit_id	INT	✓	✓	✓						
building_id	INT	✓	✓	✓						

Figure 3. visited_locations table on MySQL Workbench

Additionally, multiple many-to-many relationships were accidentally created throughout the design process. In attempting to relate many factors and create these links through multiple tables, we often made the mistake of not understanding the types of relationships we had created between the tables. This led to false connections and an improperly functioning database at times, which would return improper results upon running queries.

SOLUTIONS ENCOUNTERED

Without a primary key there is no column that identifies only one row, serving as an identification method for the table. Solutions were minimal and primary keys were crucial to the functionality of the database. One solution that was considered was the use of composite primary keys when some foreign keys were not being recognized as relationship identifiers by the software.

When many-to-many relationships were realized, we were faced with the options to alter the columns, try and manually manipulate the relationships between tables, or alter the identified foreign and primary keys. These options could all result in identifying relationships between different tables, but they also allowed for increased possibility of error.

SOLUTIONS CHOSEN

The composite primary keys were a chosen solution to MySQL Workbench not identifying certain foreign and primary keys with their corresponding icons. The composite primary keys were used to uniquely identify each row in the table when the columns are combined. We chose to use this in tables such as visited_locations, where two foreign keys were not being recognized by the dataset and a formal relationship between tables could not be identified.

When we discovered many-to-many relationships, we often solved the issue through the creation of another table in between the two tables in question. For instance, a many-to-many relationship was recognized between the symptom_check_report and symptoms tables. In response we created a reported_symptoms table to identify the link between individual, report, and symptoms. Ultimately, much trial and error was required to determine the appropriate key and relationship identifications in the ERD.

LESSONS LEARNED

Working through this assignment in a global pandemic where in-person meetings are not permitted was a challenge, disabling us from viewing each other's work firsthand. Relying on Zoom, iMessage, GitHub, and Google Docs proved difficult as there was increased opportunity for ideas to get lost in translation. Through utilizing GitHub, files were constantly being downloaded and uploaded to different programs and software, making it difficult to keep track of the current versions of different project components. Communication was key in keeping one another updated, and we as a team made efforts to text one another nearly every day and hold Zoom meetings at least once a week.

This project was useful in providing a first-hand experience of the trial and error behind logical database design. It revealed to us a more in-depth understanding of how primary keys, composite primary keys, and foreign keys are used within a database. This experience effectively put different theories learned into practice, some successfully and some not. Learning to design a database with a particular goal and questions in mind allows for different lessons than creating a very small database with a pre-selected dataset and a small number of tables. Being given the opportunity to create this from scratch not only provided chances for academic understanding, but also a newfound appreciation for our own abilities in database design.

POTENTIAL FUTURE WORK

The University of Maryland likely uses a database of similar essence on a much larger scale to track 4Maryland efforts. This database could continue to serve as a tracking mechanism in the future, especially considering that it is hypothesized that COVID-19 will likely be a long-term occurrence such as the flu, and that individuals may have to be re-vaccinated on a yearly basis.

As the pandemic progresses and UMD plans to reopen at a larger capacity, additional attributes may be added to the database. For instance, university affiliates could input visitor information in the 4Maryland site, which would be entered into a non_umd_visitor table, which would link to the visited_locations and buildings tables. Additionally, the non_umd_visitor table would have to link to the univ_id of the non-affiliate's reporter.

The general design and flow of this database could also be used for other contagious infections, such as the flu, common cold, and even select sexually transmitted infections. The schema allows for data changes and manipulations so that the database can be easily updated, providing very up-to-date information.