CSDS 341: Final Project Report
Questionnaire Website

Oleksii Fedorenko, David Frost, Matthew Garcia, Preeti Naik

December 5, 2021

1 Background

Questionnaires and surveys are very important for understanding and shaping the world. Surveys are one of the main tools for collection and analysis of data for making decisions. For example, United States uses the Census and the American Community Survey to understand how country demographics is shifting and to make decisions about the allocation of resources. Researchers use questionnaires for scientific studies to understand both the social and natural worlds. Psychologists in particular make great use of Likert scales in questionnaires to help build our collective knowledge.

Therefore, streamlining the process of collecting and analysing surveys using technology is very beneficial for decision-making. Our application would allow users to create their own questionnaires, distribute and analyse them easily. Additionally, application would allow for streamlining the decision-making process based on the analysis of data to an advanced degree.

This will be accomplished through the design of the database allowing for multiple functionalities based on the user. With different roles and permissions a user will be able to participate in questionnaires or be the creator of a questionnaire. Then, questionnaires will have a certain amount of questions and these questions will have possible answers created by the maker of the questionnaire. Responses will be recorded and using the relationships in the database queries can be written to analyze the gathered data. This survey tool streamlines the process of creating and answering surveys and allows for data analysis to make decisions or understand the participants choices.

2 Data Description

Users contains information about all of the users, questionnaire creators and reponders, that interact with the website. This data will be obtained through a registration form on the website for creating accounts that asks users for their names, emails, region, and subscription type. At minimum, every user must have at least an email since it serves as a unique identifier that allows them to log in, so the email attribute of *Users* is UNIQUE and NOT NULL.

Permissions contains information about all of the permission levels of users for each questionnaire. This data is generated when a questionnaire is made and then the creator of the questionnaire specifies the permissions for the questionnaire for everyone else as they have control over whether other people can view, answer, or edit their questionnaire. *Users* and *Roles* do not have full participation in this ternary relationship.

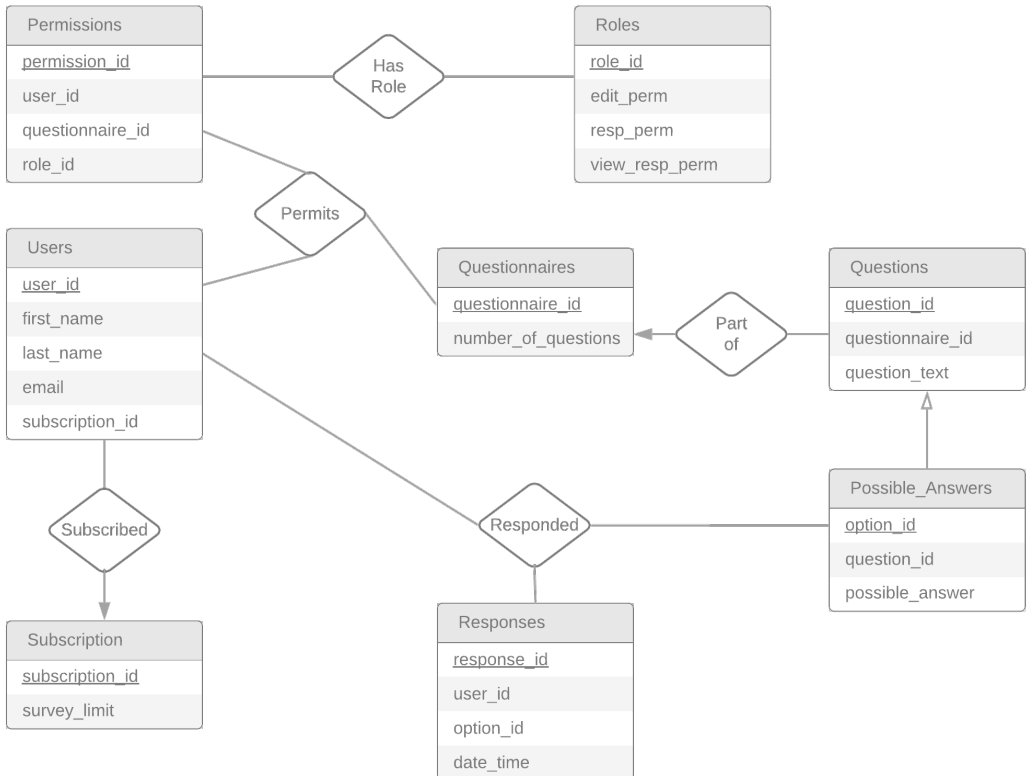
Roles contains information about all of the roles available on the questionnaire website. Each user that is related to a particular questionnaire must have a particular role. However, not every user is related to a particular questionnaire. There are seven different roles we have implemented: *root*, which has all permissions, *creator*, which is for the creators of questionnaires who can edit the questionnaire and view responses but cannot answer their own questionnaire, *moderator*, which can only edit the questionnaire but cannot answer it or view responses, *participant*, which can only respond, *analyst*, which can only view responses, and *guest*, which has no permissions. These roles only need to be added once but there is a table for roles to reduce redundancy. Therefore, we do not expect more tuples to be added to this table unless added by the website's administrator.

Subscription is the table that holds all of the info about all of the subscriptions available on the website. Users are limited about how many surveys they are allowed to create depending on their subscription plan. subscription_id is the primary key and each subscription level has a unique id. Each subscription also has a survey_limit associated with it that is the number of surveys a user with that subscription is allowed to create. This data is generated only by the administrator of the website as tuples are only added, updated, or deleted whenever the admin changes the subscription offering.

Since our project depends on user-generated data, we had to artificially generate random data to demonstrate how our database works. We generated data using a Python script which created hundreds of users with random

names, multiple choice possible answers to questions, and responses to questions and questionnaires.

3 ER Diagram



4 Functional Dependencies

For the Users table, the functional dependencies (FDs) are user_id → first_name, last_name, email, subscription_id and email → user_id ,first_name, last_name, subscription_id. The first functional dependency exists because user_id is the primary key for Users and the second functional dependency exists because each email is associated with only one user, making it another candidate key for Users.

For the Permissions table, the only functional dependency is permission_id → user_id, questionnaire_id, role_id. This functional dependency exists because permission_id is a key for Permissions.

For the Roles table, the only functional dependency is role_id → edit_perm,

resp_perm, view_resp_perm. This functional dependency exists because role_id is a key for Roles.

For the Subscription table, the only functional dependency is subscription_id $\rightarrow$ survey_limit. This functional dependency exists because subscription_id is a key for Subscription.

For the Questionnaires table, the only functional dependency is questionnaire_id $\rightarrow$ number_of_questions. This functional dependency exists because questionnaire_id is a key for Questionnaires.

5 Functional Dependencies and BCNF

Users
This entity represents users of the questionnaire website. All types of users, including people who create questionnaires or answer questionnaires, are in this table. first_name, last_name, email, subscription_id are attributes of the User and there is no FD between them except that email determines everything else since it is a candidate key as no two users can have the same email address.
R1 = (user_id, first_name, last_name, email, subscription_id)
F1 = {user_id $\rightarrow$ first_name, last_name, email, subscription_id; email $\rightarrow$ user_id, first_name, last_name, subscription_id}
Since, both user_id and email are superkeys Users is in BCNF form.

Permissions
This entity describes the permission level of a user for each questionnaire, this is part of a ternary relationship between users, questionnaire, and permission. user_id, questionnaire_id, role_id are attributes of Permission and there is no FD in between them.
R2 = (permission_id, user_id, questionnaire_id, role_id)
F2 = {permission_id $\rightarrow$ user_id, questionnaire_id, role_id}
Since permission_id is a superkey Permissions is in BCNF form.

Roles
This entity describes the roles that are allowed for each permission evel. For each role there are different permissions that are dependent on a true or false value (BIT) in SQL. For each tuple of user and survey the permission level determines what roles they have. edit_perm, resp_perm, view_resp_perm are all attributes of a role with no FD between them.
R3 = (role_id, edit_perm, resp_perm, view_resp_perm)
F3 = {role_id $\rightarrow$ edit_perm, resp_perm, view_resp_perm}
Since role_id is a superkey Roles is in BCNF form.

Subscription
Describes the subscription level and how many surveys users are allowed to create at each subscription level.
R4 = (subscription_id, survey_limit)
F4 = {subscription_id $\rightarrow$ survey_limit}
Since subscription_id is a superkey Subscription is in BCNF form.

Questionnaires

Defines a questionnaire with the unique id that will be bound to a user using the permissions table.
R5 = (questionnaire_id, number_of_questions)
F5 = {questionnaire_id, number_of_questions}
Since questionnaire_id is a superkey Questionnaires is in BCNF form.

Questions
Determines questions as they are related to each questionnaire. questionnaire_id, question_text are attributes of Question with no FD between them.
R5 = (question_id, questionnaire_id, question_text)
F5 = {question_id → questionnaire_id, question_text}
Since question_id is a superkey Questions is in BCNF form.

Possible_Answers
Defines possible answers to each question. e.g. for multiple choice questions, this will have entries for each possible answer, while for questions like rating questions, this place will have the range of rankings available. question_id, possible_answer are attributes of an response option with no FD between them.
R6 = (option_id, question_id, possible_answer)
F6 = {option_id → question_id, possible_answer}
Since option_id is a superkey Possible_Answers is in BCNF form.

Responses
Defines the user response to each of the questions and ties them to the unique response option identifier and user id. user_id, option_id, date_time have no FD between them.
R7 = (response_id, user_id, option_id, date_time)
F7 = {response_id → user_id, option_id, date_time}
Since response_id is a superkey Responses is in BCNF form.

6 Schemas

```
CREATE TABLE Users
(
    user_id INT AUTO_INCREMENT,
    first_name VARCHAR(40),
    last_name VARCHAR(40),
    email VARCHAR(100) NOT NULL UNIQUE,
```

```
        subscription_id INT,
        state VARCHAR(10),
        PRIMARY KEY(user_id),
        FOREIGN KEY(subscription_id) REFERENCES
            Subscription(subscription_id) ON DELETE CASCADE
    );

CREATE TABLE Permissions
(
    permission_id INT AUTO_INCREMENT,
    user_id INT,
    questionnaire_id INT,
    role_id varchar(30),
    PRIMARY KEY (permission_id),
    FOREIGN KEY(user_id) REFERENCES Users(user_id) ON
        DELETE CASCADE,
    FOREIGN KEY(questionnaire_id) REFERENCES
        Questionnaires(questionnaire_id) ON DELETE CASCADE,
    FOREIGN KEY(role_id) REFERENCES Roles(role_id) ON
        DELETE CASCADE
);
```

7 Example Queries

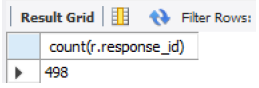
Find the number of responses to a specific question. Replace the first predicate in WHERE to specify which question. Query in SQL:

```
SELECT count(r.response_id)
FROM Questions as q, Possible_Answers as p, Responses as r
WHERE q.question_id = "question_id" AND p.question_id = q.
    question_id AND r.option_id = p.option_id
```

Query in relational algebra:

$$\gamma_{count(r.response\_id)}(\sigma_{q.question_{id}="question\_id" \wedge p.question\_id=q.question\_id \wedge r.option\_id=p.option\_id}(\rho_q Questions \; x \; \rho_p Possible\_Answers \; x \; \rho_r Responses))$$

Result of running this query with q.question_id = 1:



Find all of the questions that were responded to by people from a specific state and show what multiple choice answer they picked. Change state as necessary, it is Illinois for example here:

SELECT q.question_text, p.possible_answer **as** answer, u.user_id,
first_name, last_name, state
FROM Users u, Responses r, Questions q, Possible_Answers p
WHERE r.option_id = p.option_id **AND** p.question_id = q.
question_id **AND** u.user_id = r.user_id **AND** state = "IL";

Query in relation algebra:

$$stateResponses \leftarrow (\sigma_{\substack{r.option\_id=p.option\_id \wedge \\ p.question\_id=q.question\_id \wedge \\ u.user\_id=r.user\_id \wedge u.state="IL"}} (\rho_u Users \; x \; \rho_r Responses \; x \; \rho_q Questions \; x \; \rho_p Possible\_Answers))$$

$$\Pi_{q.question\_text,p.possible\_answer,u.user\_id,first\_name,last\_name,state}(stateResponses)$$

Query in tuple relational calculus:

$$\{t|(\exists q)(Questions(q) \wedge q[question\_text] = t[question\_text] \wedge (\exists p)(Possible\_Answers(p) \wedge q[question\_id] = p[question\_id] \wedge t[possible\_answer] = p[possible\_answer] \wedge (\exists u)(Users(u) \wedge t[user\_id] = u[user\_id] \wedge t[first\_name] = u[first\_name] \wedge t[last\_name] = u[last\_name] \wedge t[state] = u[state] \wedge u[state] = "IL" \wedge (\exists r)(Responses(r) \wedge u[user\_id] = r[user\_id] \wedge p[option\_id] = r[option\_id])))))\}$$

Result of running this query with state = "IL":

question_text	answer	user_id	first_name	last_name	state
What is your child's proudest accomplishment?	nfyqM	237	Cheryl	Hunter	IL
What motivates you to work hard?	JjNxs	237	Cheryl	Hunter	IL
Are you related or distantly related to anyone f...	UHYje	237	Cheryl	Hunter	IL
Who would you want to be stranded with on a ...	Fzuhb	237	Cheryl	Hunter	IL
If you could go back in time, what year would y...	ArFGJ	237	Cheryl	Hunter	IL

8 Implementation

We used MySQL as our database management system. We used Python to create a console app that allows for many interactions with the system from the command line, such as adding a new user, viewing the list of questionnaires on the website, answering a questionnaire, viewing the answers to a particular questionnaire, and creating a questionnaire. This Python app uses MySQL Connector to connect to MySQL, so it may be necessary to run *python -m pip install mysql-connector-python*. We also started developing a Python Flask app using SQLAlchemy to connect to the MySQL database. To run this app, you need to go to the directory csds341_server and then run *pip install -r requirements.txt* to be able to run it, and then run *flask run*, and then go to your web browser of choice and visit 127.0.01:5000 or localhost:5000 to see it.

9 Demo

We did a presentation in the last week of class. The slides are in the writeup folder in a file called "CSDS 341 Project Presentation".pdf. We also did a demo to our TA where we showed him our schemas which were significantly updated from our initial report with many additional constraints. We showed him that we successfully created our MySQL database for the project named "csds341_questionnaire" and our progress on our Flask web app and our Python console app for interacting with the questionnaire system from the command line. We also showed him our sample queries and the successful results of running them.

10 Contributions of Each Team Member

Oleksii:

- Database structure
- Initial ER diagram
- ER to SQL Tables
- Helped with Functional Dependencies
- Wrote script to generate sample data
- Set up and worked on the Python Flask
- Helped with final report

David:

1. Sample queries, including SQL, RA, and TRC queries
2. Background section of report
3. Data description

Matthew:
Preeti:

11 What We Learned

During the course of this project, we learned about how to design our own database system and how to connect the many parts of our project and application together. The lectures taught us how to analyze and describe databases, schemas, and ER diagrams that other people have already made, but the project required us to create our own project and then design a database from scratch for it. The course helped a lot with the theory behind designing a database such as through the rules for ER diagrams and normal forms for schemas, but working on the project gave us the practical experience of actually creating our own database system. We also learned how to use MySQL for our project. Additionally, we learned about how to create web apps using Flask and how to connect MySQL to Python through SQLAlchemy.