UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Operation Christmas Child Data Management System Database

Author:

David Bretz

Supervisor:

Prof. Kenrick Mock, PhD

Anchorage AK, May 2016

Computer Science &
Engineering Department
UNIVERSITY of ALASKA ANCHORAGE

David Bretz

dmbretz@alaska.edu

Abstract

This paper is about the database for the Operation Christmas Child Data Management System. The database will be used to reduce the work required to collect and maintain donor information and donation statistics by Year-Round Volunteers. A website will be created as the interface for the database and will be used by both donors and Year-Round Volunteers. The database needs to be able to store location information for both Drop-Off Locations and events. This will be used by the interface to determine whether a donor is at a specific Drop-Off Location or event. The interface will do this by determining if the donor's coordinates and the Drop-Off Location or event's coordinates are within feet of eachother. The database will also need to store accounts for Year-Round Volunteers who are to be using the system. This system will save Year-Round Volunteers weeks worth of work processing paper Drop-Off Logs and improve data integrity.

Acknowledgments

Firstly, I would like to thank my Lord God for giving me the strength to go forward with this project. Without him, I probably would have not finished my degree nor would I have chosen to develop a project for Samaritan's Purse.

I would also like to thank my mom for all the support she has provided. She always knew what to say when I was stressed and was always willing to listen, even if she didn't understand what I was talking about.

I also want to thank my dad for his support for me and his concern for my betterment. He provided great support for me through these past few years, even when I stressed him out with my schedule and eagerness to overschedule.

Thank you to my girlfriend, Brittany Cave, for always knowing how to cheer me up when I was down.

To Dr. Kenrick Mock, I would like to express my thanks for his support in this project. He was always ready with feedback on the design implementation and was ready to tell me when I was adding to much complexity.

I want to thank Dr. Adriano Cavalcanti for his support on this project. He was helpful with his feedback and pushed me forward when I needed it.

My thanks goes out to Dr. Kirk Scott. It was him who introduced me to the world of databases and who inspired me to undertake this project.

I want to give my thanks to Deb Bronson for sponsering the project and giving feedback when necessary.

I would also like to express my gratitude to the Operation Christmas Child South Central Alaska Team for allowing me to develop this project for them.

Contents

List of Figures

Chapter 1
Introduction

1.1 Introduction

Operation Christmas Child Data Management System (OCCDMS) is a data entry system that will reduce the workload of year-round volunteers. The goal of the project is to develop a system that displays a form based on the user's location. The system will need to know where the user is and display a form based on the time of day and whether they are at a registered event or a shoe box Drop-Off Location and reject them otherwise. It will also need to allow a year-round volunteer, known as a Connect Volunteer, to view this data and use it for future decisions. This application can be modified for use by other organizations where there are multiple events and locations that need to be managed.

Figure 1.1 Donors dropping off shoeboxes at a collection center (Samaritan's Purse, 2015)

1.2 Application

The implementation of the new data management system will request a user's location to determine what form the user needs. There are two situations that a user would use the forms to record information. First, donors would visit the form page during collection week to log their donation. Second, there are multiple events throughout the year where participants would need to register their attendance.

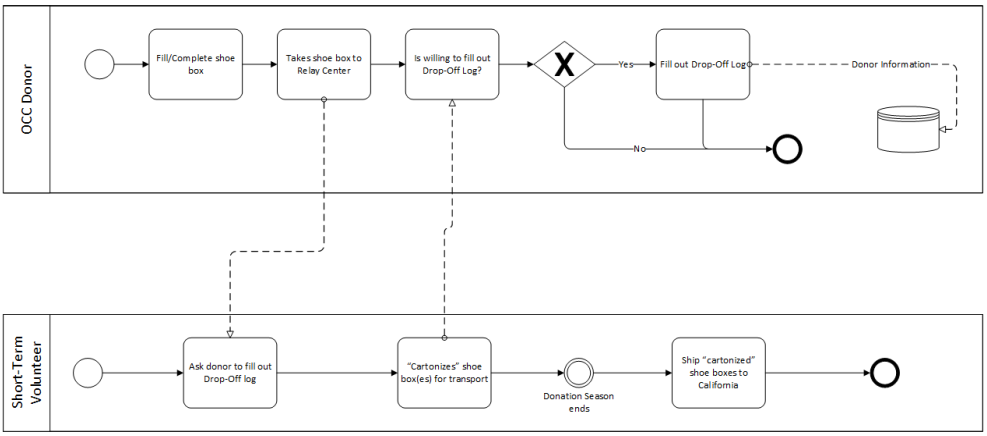


Figure 1.2: The proposed BPM for collecting a Drop-Off Log from a donor

Currently, companies use IP (Internet Protocol) geolocation and mobile gps location to display relevant ads, for giving directions, or even displaying special messages. This is done using either a phone's gps location or using IP geolocation databases. Using this information, a relevant form can be displayed based on their physical location.

The project will be developed under an MIT License.

1.3 Motivation

The motivation for developing the current work is my involvement with Operation Christmas Child and an interest in using a user's location to display information dynamically. Currently, when a donor drops off a shoebox gift at a relay/collection center, they fill out a paper drop-off log that stating how many boxes they are drop-off along with their contact information. This information is then entered by hand by volunteers into an excel spreadsheet that is kept by the area coordinator (figure 1.2).

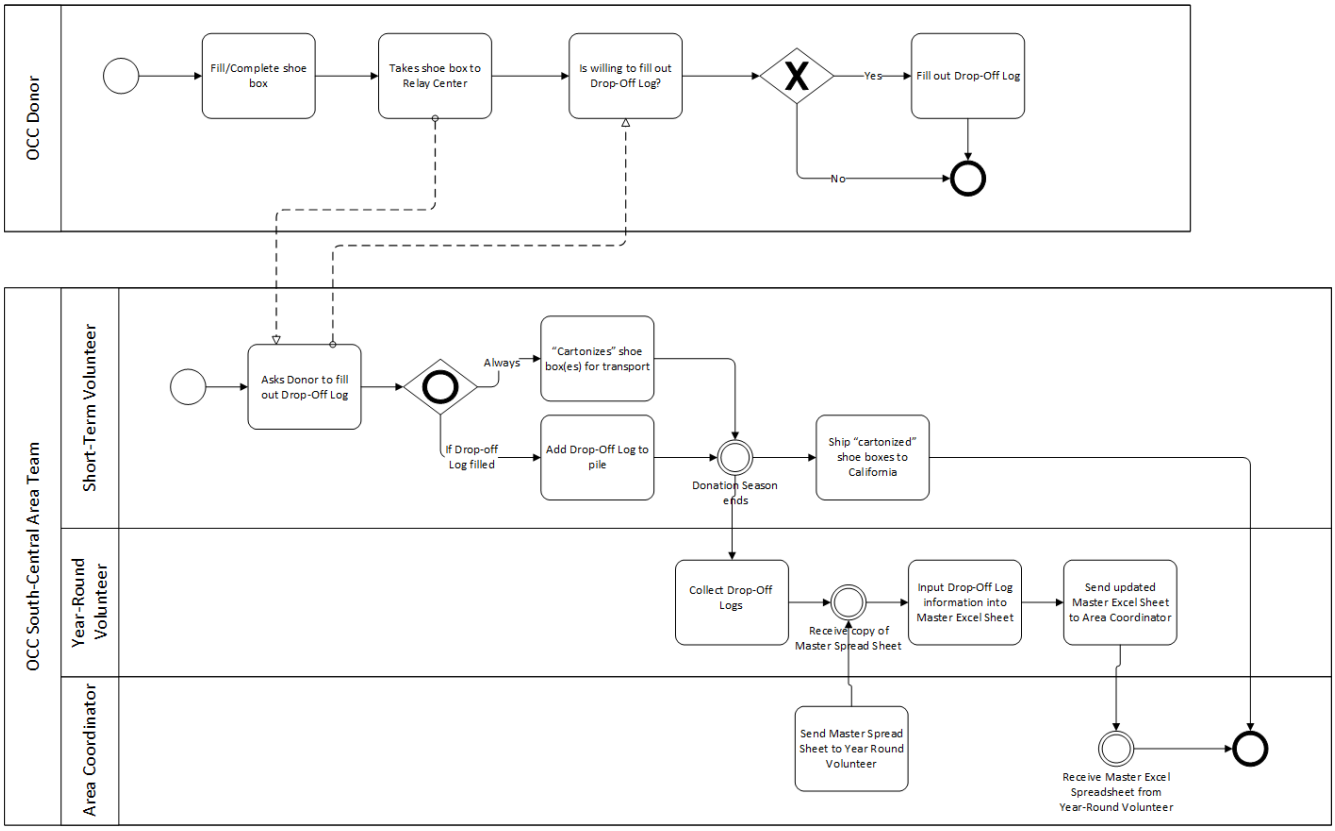


Figure 1.3: The current BPM for the collection of donor information

This would be an acceptable system if this was a small project, but this is not a small project. (Samaritan's Purse, 2015) (figure 1.2)[1]. This has become a lengthy process that takes up many hours of personal time and is prone to errors due to illegible handwriting, missing forms, or other human errors.

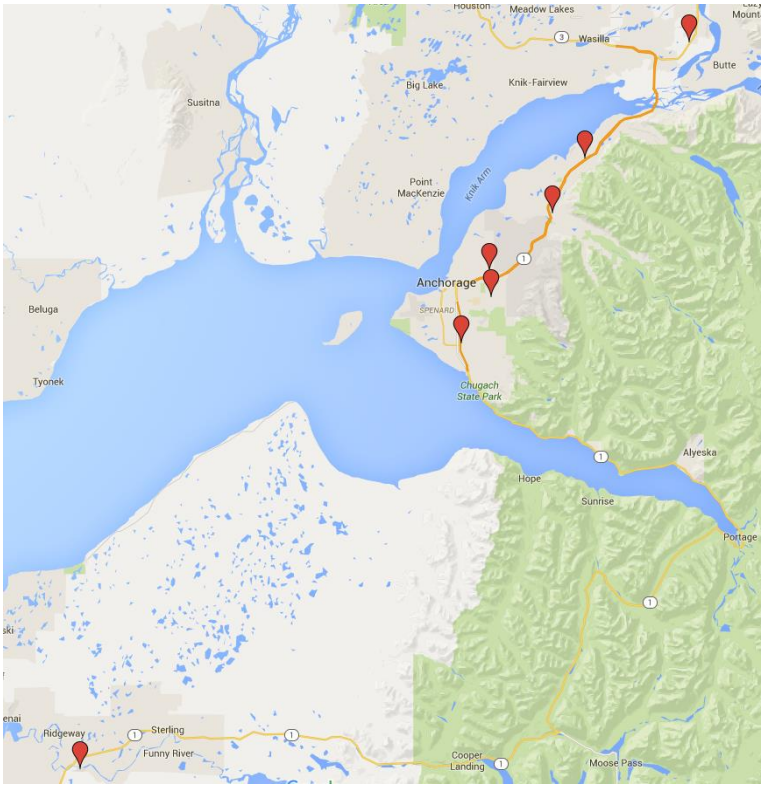


Figure 1.4: A map of seven relay/collection centers in South Central Alaska

If all the data is stored on a single database where the donor's information is entered by the donor at the collection/relay center, data integrity will improve and the strain on volunteer's time will be reduced, granting them more time to reach out to the community.

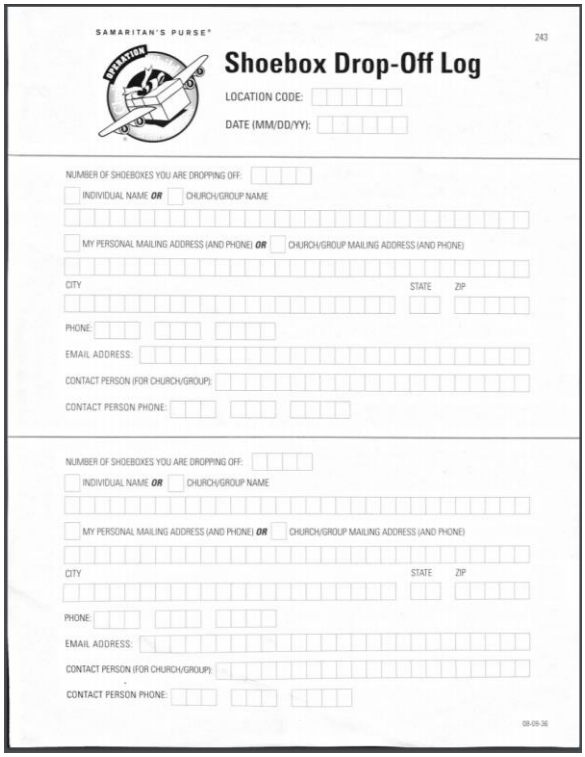
Chapter 2
System Integration and Modeling

2.1 System Overview

The system will consist of two main parts: the website and the database. The website will serve as the interface for the database. It will allow both donors and Connect Volunteers to input data to the database. It will also prevent unauthorized individuals from retrieving information from the database that they are not permitted to see.

2.2 Designing the Database Model

The focus of the project was improving data integrity and accessibility. To do this, we needed to review how the local area team was storing and collecting their data from their donors. An interview with the Regional Area Coordinator for Alaska and the Network Coordinator for Southcentral Alaska was performed in order to determine what data needed to be stored in the database as well as determine how the data would be used. We were also provided a copy of the form that they currently have donors fill out when they drop off shoeboxes at their collection center. I, as a Connect Volunteer, was given an old copy of their excel spreadsheet as an example of the current system being used.



SAMARITAN'S PURSE®

243

Shoebox Drop-Off Log

LOCATION CODE:

DATE (MM/DD/YY):

NUMBER OF SHOEBOXES YOU ARE DROPPING OFF:

INDIVIDUAL NAME *OR* CHURCH/GROUP NAME

MY PERSONAL MAILING ADDRESS (AND PHONE) *OR* CHURCH/GROUP MAILING ADDRESS (AND PHONE)

CITY STATE ZIP

PHONE:

EMAIL ADDRESS:

CONTACT PERSON (FOR CHURCH/GROUP):

CONTACT PERSON PHONE:

NUMBER OF SHOEBOXES YOU ARE DROPPING OFF:

INDIVIDUAL NAME *OR* CHURCH/GROUP NAME

MY PERSONAL MAILING ADDRESS (AND PHONE) *OR* CHURCH/GROUP MAILING ADDRESS (AND PHONE)

CITY STATE ZIP

PHONE:

EMAIL ADDRESS:

CONTACT PERSON (FOR CHURCH/GROUP):

CONTACT PERSON PHONE:

08-09-36

Figure 2.1: A blank Shoebox Drop-Off Log provided by Deb Bronson

We determined that the database needed to store contact information for individuals who either attended an event set up by the local area team or when they drop off shoeboxes at one of the drop-off locations throughout the state. The only difference between the data collected at events and at the drop-off locations was that the at the drop-off locations, the number of shoeboxes was counted and attributed to the drop-off location. Organizations could drop-off shoeboxes at drop-off locations as well and were stored in the same excel spreadsheet along with the information for individuals. It was decided that we needed to build the database to keep the information for individuals and organizations distinct. Even so, much of the information for individuals and organizations are similar, so we made the person table and the organization table share other contact tables that stored similar information.

Another aspect that needed to be considered in the design process was that the local area team needed to keep track of Pastors for churches and project leaders who coordinate relations between OCC and their organization. We were informed that it was important for the database to track when a project leader started or stopped being a project leader for a particular organization.

The local area team needed a way to access the database. A team member would need a password secured account. They would also need to have slightly different privileges based on their position within the team. Team members also needed to be associated with their specific area team. To accomplish this, we would need a table to store accounts, a table to store team members and a table to store teams. Not all Connect Volunteers are associated with a specific area team as they are in charge

of multiple area teams. We needed to create another table to track Regional Area Coordinators and a table to keep track of all the teams in their region.

There was another type of volunteer that needed to be tracked as well. These are known as Short-term volunteers and were to work at the Drop-Off Locations. These volunteers were not to have direct access to the contents of the database and were only to be tracked for liability and for potential future work. This meant we needed to know where they volunteered at and when they started or stopped.

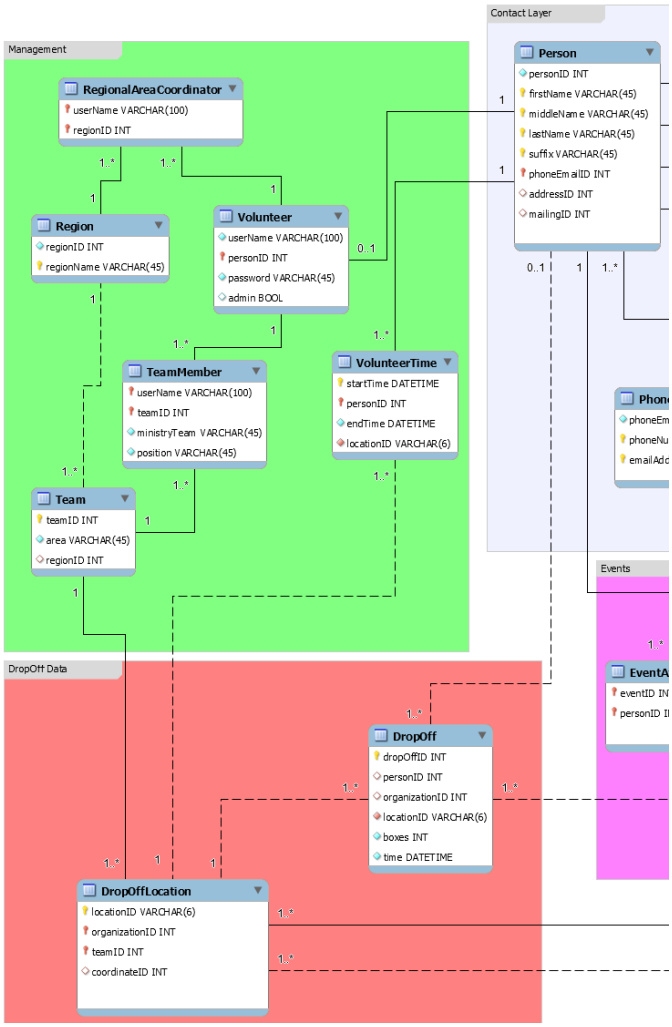


Figure 2.2: The tables to be used for both Year-Round and Short-Term Volunteers

We needed to keep track of shoebox drop-offs in a meaningful manner. In order to accomplish this, we created a table to store every drop-off. This table would need to store the number of shoeboxes dropped off, the individual or organization who dropped it off, when it was dropped off, and which Drop-Off Location it was dropped off at.

Drop-Off locations needed to be associated with a team and an organization. They needed to be

associated with an organization because they were always associated with one and it allowed for easier look-up. They also had to have a six character location ID that is assigned by OCC. This would allow for the data collected by the local area team to be read into Samaritan's Purse's system.

Event information was also to be tracked, including who attended an event, where the event was, its start and end time, as well as a description and image.

We originally planned on using Google's API to use an organization's address to determine whether a person was at a Drop-Off Location or event. However, as the project developed, we decided that it would be more practicle to assign actuall geographical coordinates to the events and Drop-Off Locations. We then added a coordinate table the model. All of this was used to create the model shown in figure 2.2.

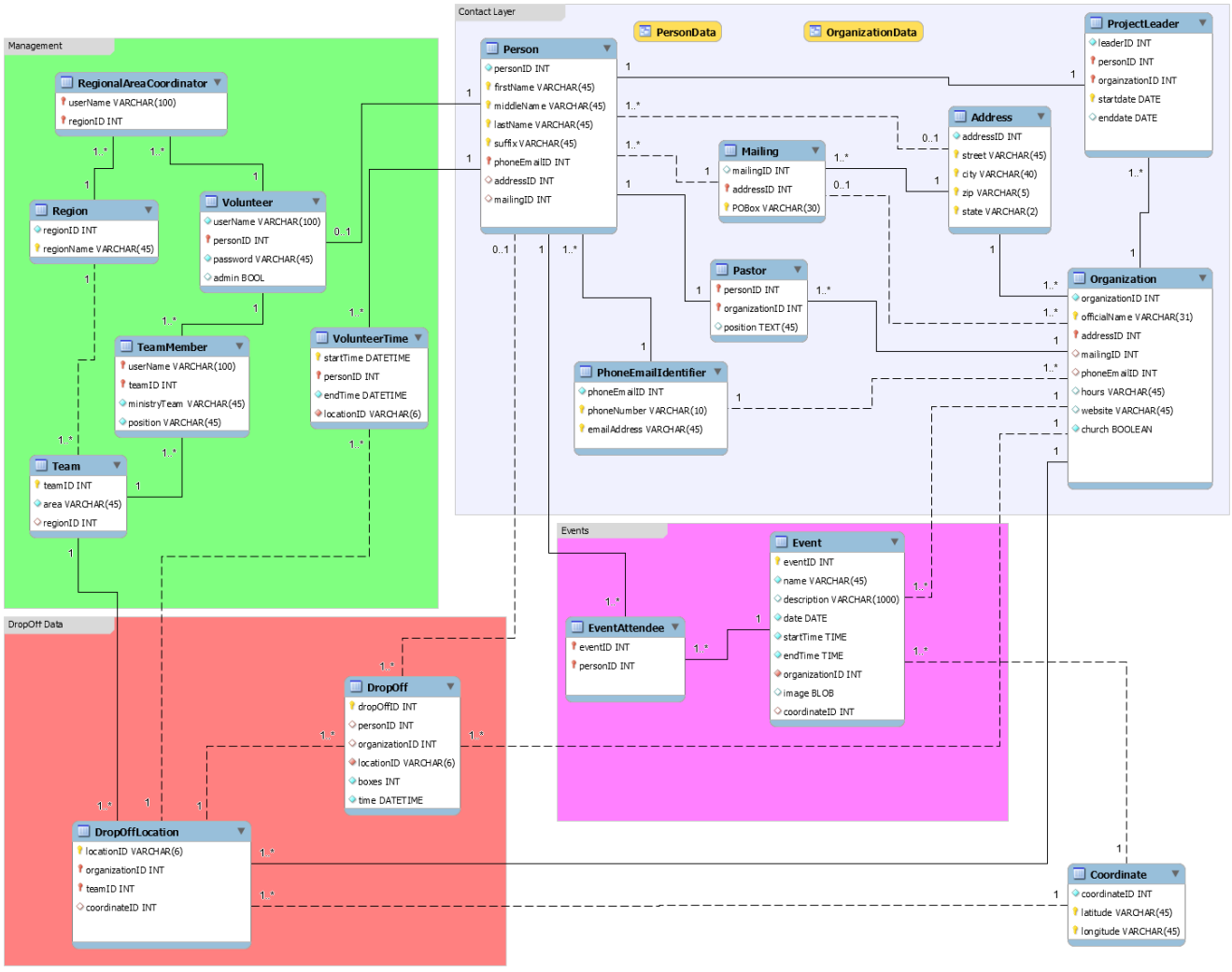


Figure 2.3: Database model for OCCDMS

2.3 Reducing Duplicate Data

One of the aspects of the project was to reduce or eliminate duplicate entries for individuals and organizations. We decided that organizations weren't going to be too much of a problem since we could use the name of their organization and their address as primary keys. This would allow us to still track organizations with multiple locations. This was important to track because organizations only had partnerships with OCC at the local level. It also allowed us to track donations from individual branches of the organization.

Tracking donations from individuals was a little more difficult. We initially thought of having each donor create an account using a username or email address and creating password. This would have eliminated duplicate entries for individuals within the database. This would mean that donors would take slightly longer when making a donation for the first time due to the need to create an account, but subsequent donations would be quicker since all they needed to do is enter their username, password, and the number of shoeboxes they were dropping off. This was later rejected due to how and when the information was to be collected. Most information would be collected during a one week period in November. This meant that donors would be likely to forget their information in the one year period between using their account. We were also informed that there was a good number of donors who still were not familiar with computers and didn't use email. We instead opted to build the database to reduce the number of duplicates rather than entirely eliminate them. To accomplish this, we decided that in order to create an entry for a person, it would be mandatory for the person to have a first name, a last name, and either an email address or phone number or both. Other optional information that would be used to create an entry for a person would be their middle name and a suffix. This would allow us to track individuals who lived in the same household and shared a phone number or email address. This would not allow individuals to change their contact information however, as any changes to a person's name, email address, or phone number would result in a new entry in the person table. This was deemed an acceptable compromise as this still would result in fewer entries for individuals than the current system being used by the Local Area Team and did not require donors to remember an account that they had created the year prior and hadn't used since.

Information for Connect Volunteers was easier to manage since these entries would require an administrator's approval to create an account. The account would be associated with an entry in the person table, so they'd be able to manage their own information.

Drop-Off Locations and events would be maintained by Connect Volunteers. As information for Drop-Off Locations was to be sent to the Local Area Team, the risk for duplicates was low. Even so, we made sure that a Drop-Off location with the same data could not be created. Events were treated much the same way as Drop-Off Locations.

2.4 Interacting with the Database

When the project was initially planned, the plan was to have all interaction with the database be done using PHP. However, as the database grew in complexity, it was decided that I would need to create a collection of MySQL routines and views for common actions that would be performed. This would make it easier when working on the website as the more complex queries would have already be created within the database. This greatly simplified the interaction between the database and the website as queries with multiple joins that would need to be run had been reduced much simpler queries. It also simplified inserting and updating information within the database as just creating an entry in individual required three separate insert queries.

```
CREATE PROCEDURE `insertperson`(IN firstNamein varchar(45), IN middlenamein varchar(45), IN lastNamein varchar(45), IN suffixin varchar(45), IN streetin
varchar(45), IN cityin varchar(40), IN zipIn varchar(5), IN statein varchar(2), IN phonein varchar(10), IN emailin varchar(45))
BEGIN
INSERT IGNORE INTO Address(street, city, zip, state) VALUES(streetin, cityin, zipin, statein);
INSERT IGNORE INTO PhoneEmailIdentifier(phoneNumber, emailAddress) values(phonein, emailin);
INSERT INTO Person(firstName, lastName, middleName, suffix, addressID, phoneEmailID)
SELECT firstNamein, lastNamein, middlenamein, suffixin, Address.addressID, PhoneEmailIdentifier.phoneEmailID
FROM Address, PhoneEmailIdentifier
WHERE Address.street = streetin
AND Address.city = cityin
AND Address.zip = zipIn
AND Address.state = statein
AND PhoneEmailIdentifier.phoneNumber = phonein
AND PhoneEmailIdentifier.emailAddress = emailin
ON DUPLICATE KEY UPDATE Person.addressid = (SELECT addressID FROM Address WHERE Address.street = streetin
AND Address.city = cityin
AND Address.zip = zipIn
AND Address.state = statein);

END
```

Figure 2.4: An example of a routine that inserts information for a person

2.5 Project Plan

The plan for the project was to use a waterfall method of development. When the project was first started, it was developed with intention of having three team members on the project and to develop it using MySQL, ASP.NET, and to create a bootstrap. A WBS was created with this in mind and a plan was set. This fell through at the beginning of the semester and the project was reduced to me as the only developer. The language was changed to PHP, CSS, and MySQL.

Halfway through the semester, Cody McWilliams joined my project as a frontend developer and JavaScript and jQuery was added to the languages. These changes made sticking with the project plan

difficult and we were not able to keep up with the plan.

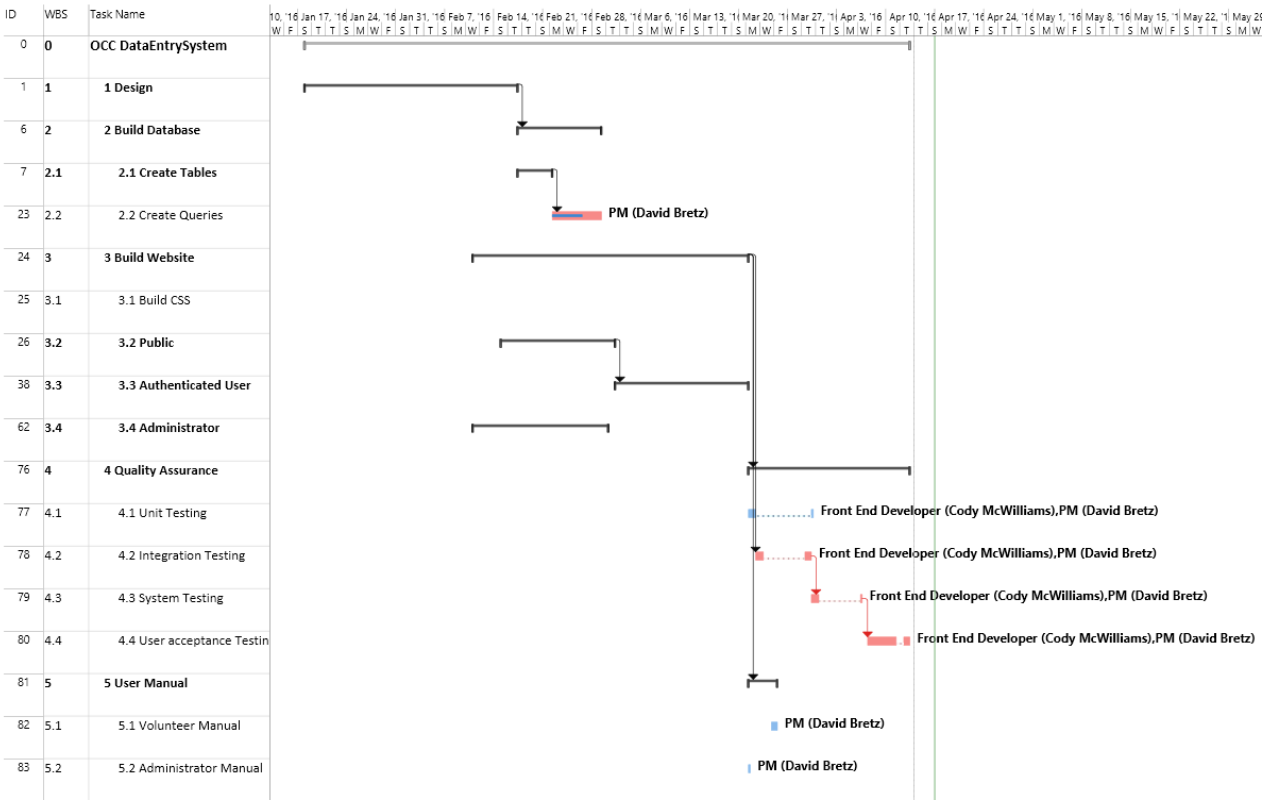


Figure 2.5: Project Gantt Chart

Chapter 3
Design and Testing

3.1 User Interface

The user interface for the database will be the website. The website will provide a way for donors to input their personal information as well as the number of shoeboxes they are donating if they are at a Drop-Off Location. The website will automatically determine which Drop-Off Location or event location the individual is at. It will not allow an unauthorized user from retrieving any information from the database.

Authorized users will be able to do all that an unauthorized user can do. Additionally, they can add information into the system manually in case there is a Drop-Off Location where there is no internet or there was some other instance that forces them to use the old paper forms. These users will also be able to view all personal and organizational information along with drop-off and event statistics. They will also be able to edit information that they know was entered in error. Users with high enough privilege will be allowed to create new user accounts for the system they will also be allowed to create and edit Drop-Off Locations. There will be no way provided for a user to view an account's password. An account's password is to never be displayed. The only interaction that a user will have with their password will be when they enter it to log in or when their password is reset by an admin.

3.2 Design Tools

In order to develop the layout for the database, multiple interviews were performed with members of the South Central Alaska Team. It was important that the database captured everything that they use

and provide information that would be beneficial to their team. We also reviewed copies of their current forms used for collecting donor information. (Deb Bronson, 2015-2016)

Most of the testing and development of the database will be through either command line or via the MySQL Workbench. Workbench makes it easy to visualize how the tables relate with each other and makes it easier to perform repeated tasks.

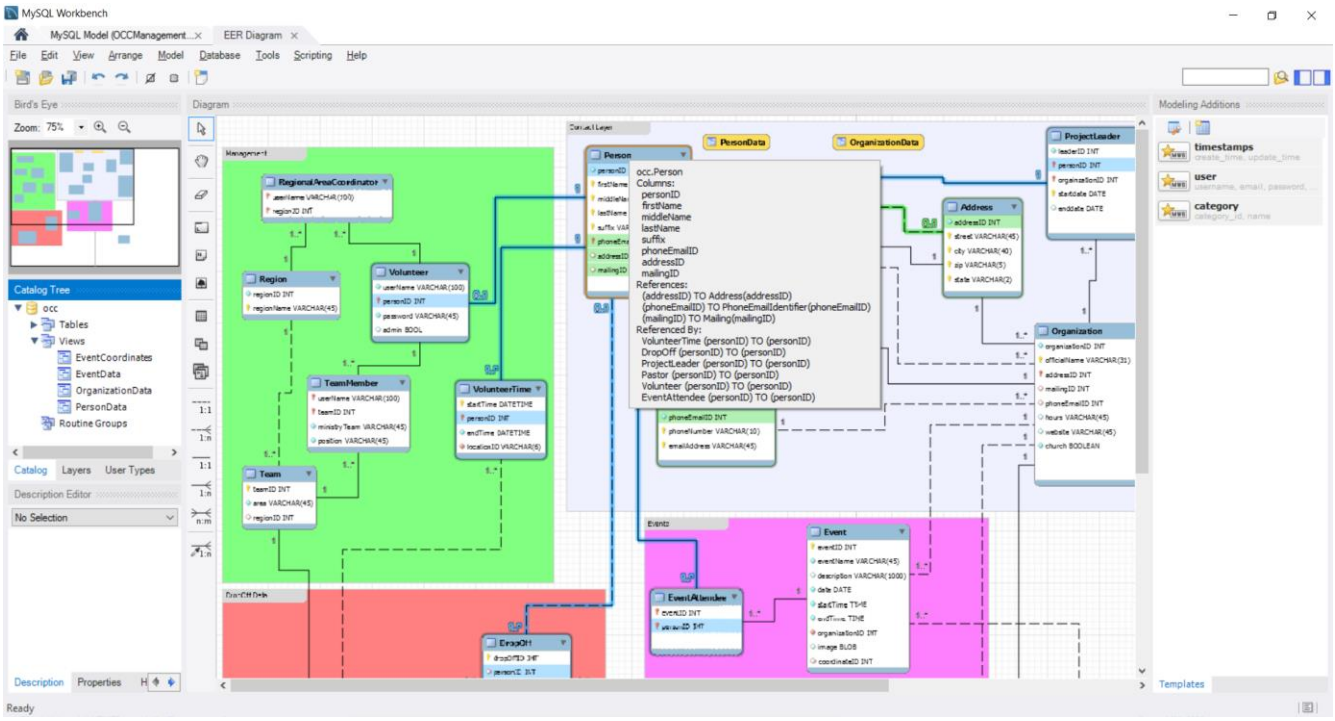


Figure 3.1: MySQL Workbench EER Diagram

3.2 Database Testing

We did not officially pick a testing method for the project. We did test the database using a series of unit tests. These tests were done mostly by hand with a few exceptions. At the beginning of the project, I compiled a list of common entries that would need to be supported as well as what would need to be prevented. As the project progressed and the database evolved, new tests were developed and in some cases old tests needed to be checked.

Each table was tested on insert, update, and delete operations individually. This was done to make sure that each table was capable of supporting the information that was to be stored. I then attempted to enter invalid data into the tables to confirm that the database would reject the data.

Chapter 4
User Manual

4.1 Overview

The Operation Christmas Child Data Management System (OCCDMS) is a database system with a web interface. This system will work on both mobile and desktop environments and will not require the user to install any additional software. This system is designed to store all donor information for both organizations and individuals. This will also be used to collect shoebox drop-off information straight from donors.

Figure 4.1 Donors dropping off shoeboxes

4.2 Donors

Once the website is deployed, donors will be able to visit the website. Once the donor arrives at the home page, they will be greeted by a screen similar to figure 4.1. Once there, the donor will have three options to choose from; Drop-Off, Event Check-In, and Short-Term Volunteers. It is recommended that a direct link to one of the pertinent options is provided at the Drop-Off Location or event either in the form of a QR code or text. The home page is provided as a general access point.

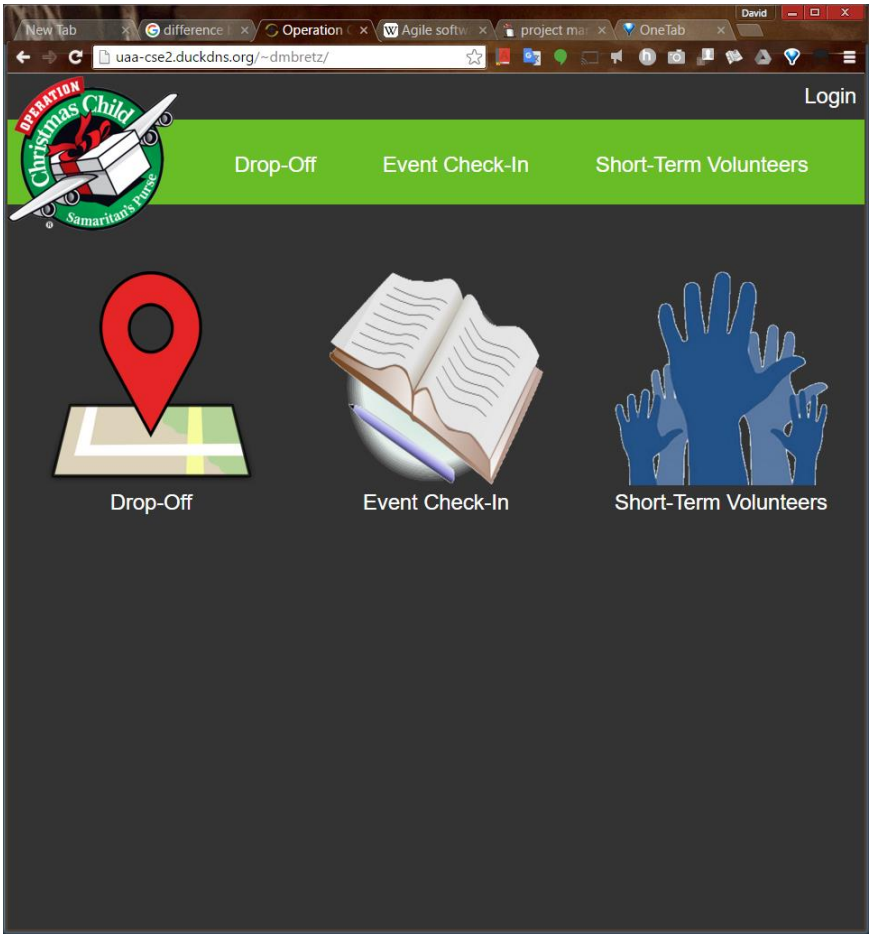


Figure 4.2: Home page for OCCDMS

If a donor is at a Drop-Off Location and they want to drop off a shoebox, they would visit the Drop-Off link. When there, they will be provided the option to donate either anonymously, as an individual, or as an organization. In order for a donor to fill out the form, the donor must be within 500 feet of a Drop-Off Location. If they are not within 500 feet of a Drop-Off Location, they will be told to proceed to the

nearest Drop-Off Location and be directed to a map of active Drop-Off Locations. In order to determine whether the donor is within this range, OCCDMS must know the donor's location. This means that when the webpage requests the donor's location, the donor must grant the website access to their location. This is critical to knowing which Drop-Off location is receiving the shoeboxes as location codes are handled automatically by OCCDMS. If the donor denies the website from viewing their location, they will not be allowed to fill the Drop-Off Log. If a donor has concerns about OCCDMS knowing their current location, assure them that their location is not being tracked continuously and will only be used to determine which Drop-Off Location they are at.

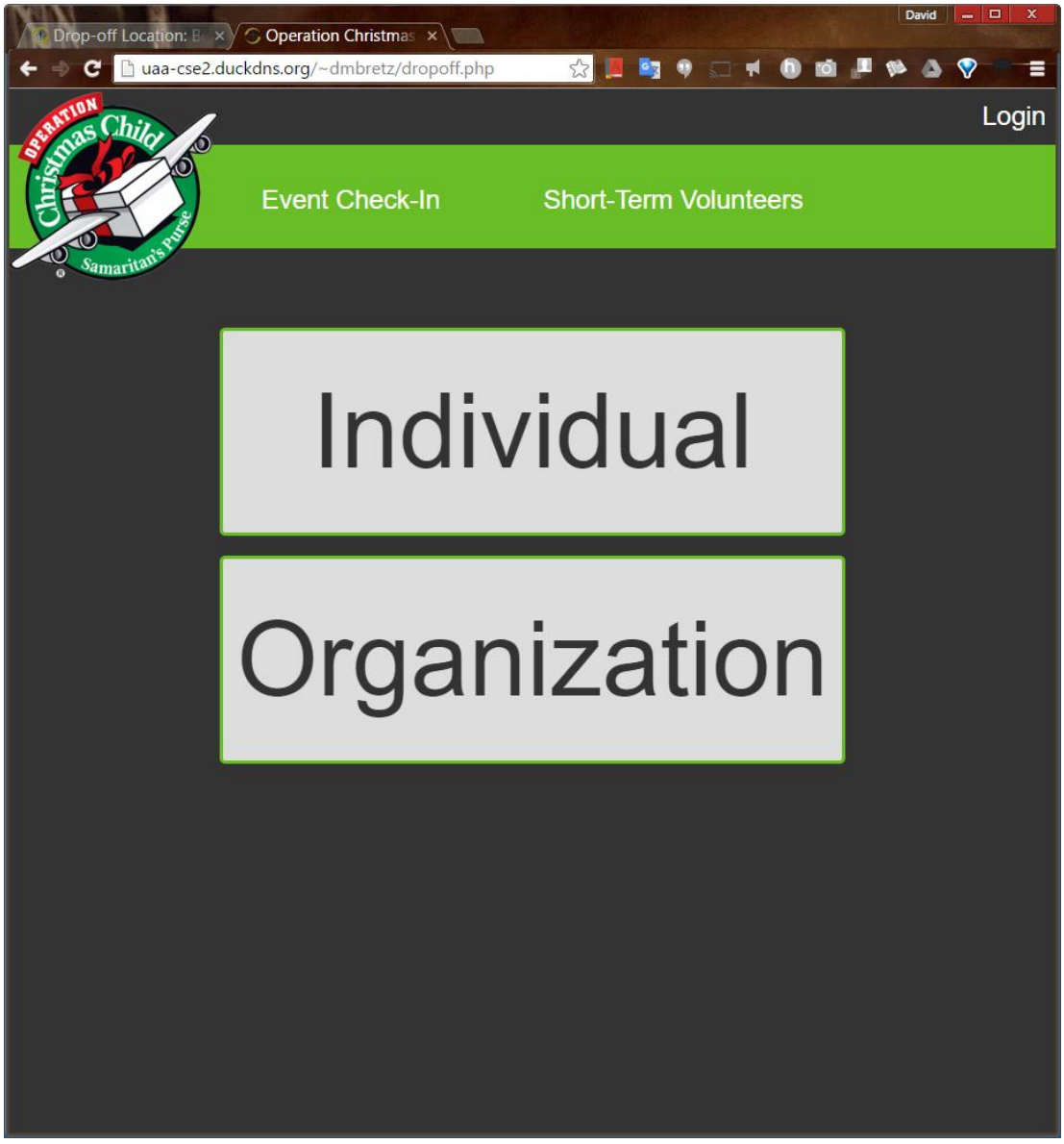


Figure 4.3: Donors will be able to fill a Drop-Off Log either as an individual or an organization

Event Check-In is handled in the same way as Drop-Off is. The only difference between Drop-Off and Event Check-In is that it only handles individuals and does not direct users to a map of events in their area. The user will still need to grant OCCDMS access to their location, as this is used to determine which event they are attending.

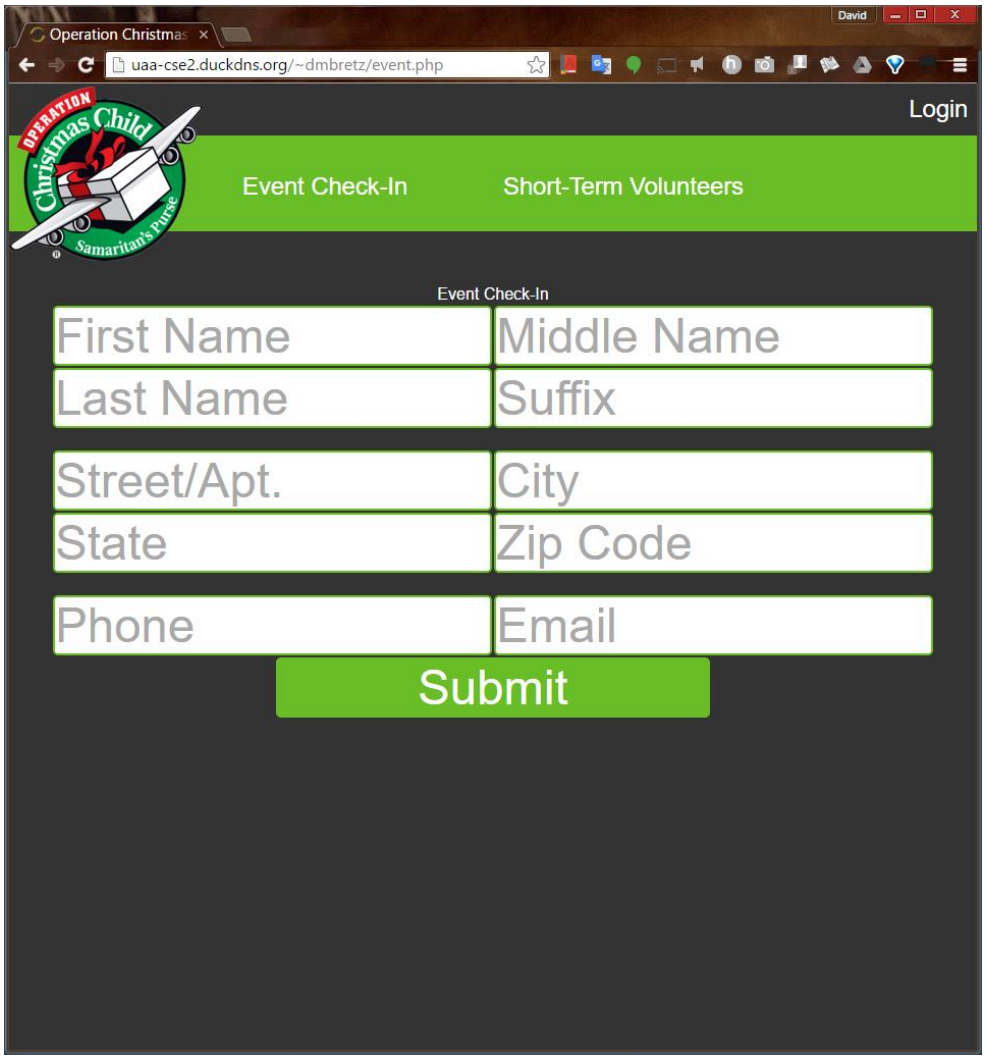


Figure 4.4: Event Check-In form

Short-Term Volunteers will be used for allowing volunteers to sign-up for short-term volunteer roles at Drop-Off Locations. They will need to select a Drop-Off Location to work at, a date, and a time. When they arrive at the Drop-Off Location, they will need to sign in, much like you would for a time clock. At the end of their shift, they will then check out. This is important for the safety of volunteers at the Drop-Off Locations.

4.3 Connect Volunteers

Connect Volunteers who have been given an account by their Regional Area Coordinator or Area Coordinator will need to login to the OCCDMS by clicking on the login link at the top right of the screen.

Once they are logged in, they will be greeted by the Volunteer Hub. From here, they will be able to view all donor information, look-up individual organizations, persons, project leaders, and pastors. They will also be able to view Drop-Off statistics for their area or for individual Drop-Off Locations.

Volunteers with an account will be able enter Drop-Off Logs manually from here as well. This is provided in case donors need to resort to paper forms for a particular reason or shoeboxes were donated after hours and were not collected by a volunteer.

Connect Volunteers will be able to create events from here as well. When creating an event, the volunteer will need to name the event, provide a description, a date and time, an image, and geographical coordinates. The coordinates will be used by OCCDMS for checking users into the event.

Connect Volunteers with a rank of Regional Area Coordinator, Area Coordinator, or Network Coordinator will be allowed to create Drop-Off Locations in much the same manner as an event. Drop-Off Locations, however, only require a Location, an organization to associate the Drop-Off Location with, and its geographical coordinates.

Connect Volunteers with a rank of Regional Area Coordinator or Area Coordinator will be able to create new user accounts for volunteers under them.

Chapter 5
Summary and Conclusion

5.1 Summary

The Operation Christmas Child Data Management System is still needs work, but the framework has been set. The database is has been built and works and has been tested. All that is needed is for the user interface to be completed.

The system has been shown to be capable of storing all of the data that is currently used by the Local Area Team. It also reduces the number of duplicate entries with its use of concatenated keys.

Figure 5.1: A worker at the California Processing Center which also serves as a Drop-Off Location

OCCDMS will reduce the workload of Connect Volunteers who enter donor information every year. The goal of the project is to develop a system that displays a form based on the user's location. The system will need to know where the user is and display a form based on whether they are at a registered event or a shoe box Drop-Off Location and reject them otherwise. It will also allow a Connect Volunteer, known as a Connect Volunteer, to view this data and use it for future decisions. This application can be modified for use by other organizations where there are multiple events and locations that need to be managed.

References

Bretz, D. (n.d.). Anchorage, Alaska, United States of America.

Deb Bronson, S. B. (2015-2016). (M. M. David Bretz, Interviewer)

Google.com. (2016, April). *maps.google.com*. Retrieved from google.com.

Samaritan's Purse. (2015). Collection Center Coordinator Ministry Handbook. Boone, North Carolina, United States of America.

Samaritan's Purse. (2015, November 9). Drop-off Location: Behind the Scenes. Boone, North Carolina, United States of America.

Appendix A

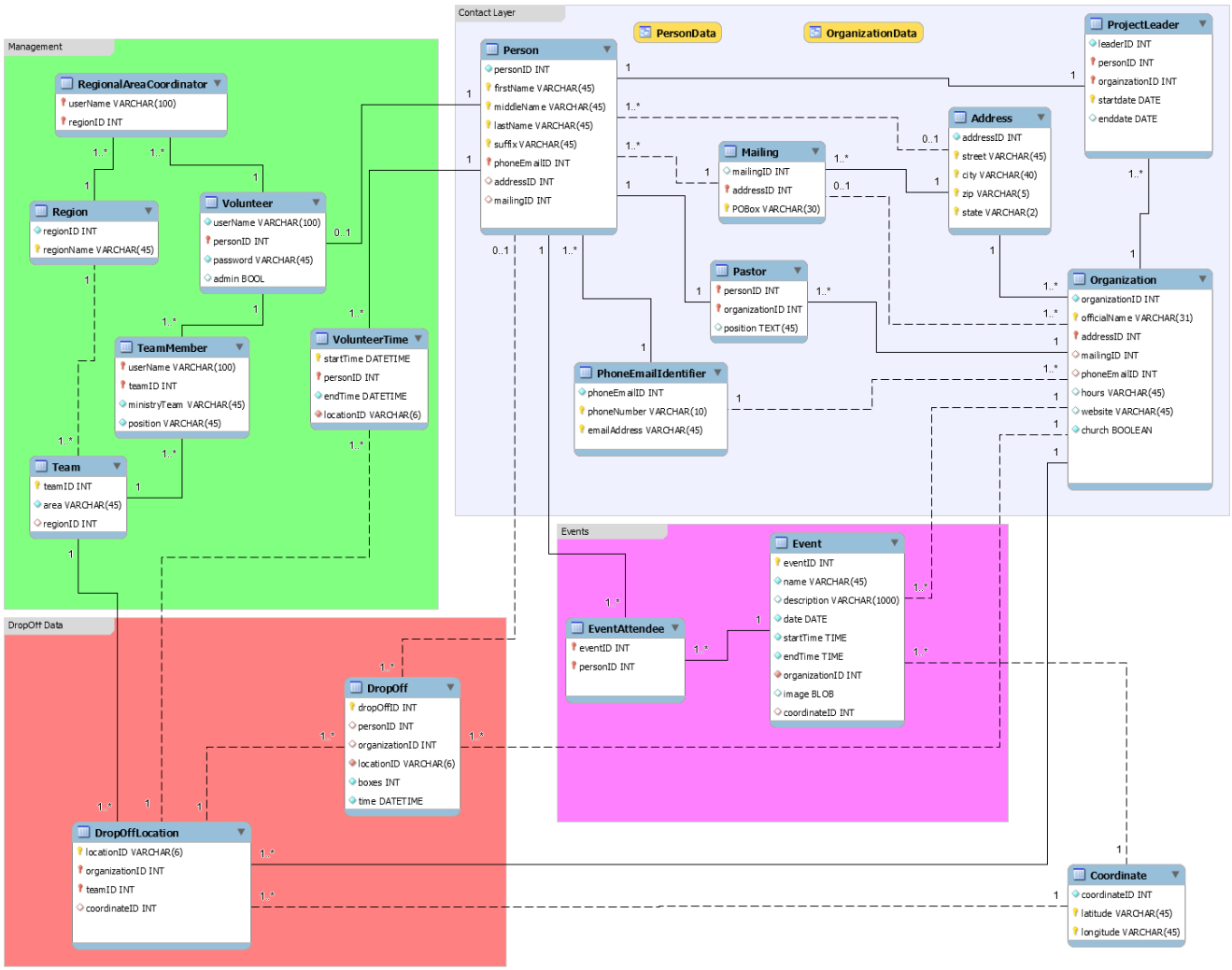


Figure A.0.1: Database Model

Appendix B

https://github.com/Dwiddwid/OCCGeolocationBasedForms

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema occ
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema occ
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `occ` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci ;
USE `occ` ;

-- -----------------------------------------------------
-- Table `occ`.`Address`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `occ`.`Address` (
  `addressID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `street` VARCHAR(45) NULL DEFAULT '' COMMENT '',
  `city` VARCHAR(40) NULL COMMENT '',
  `zip` VARCHAR(5) NULL COMMENT '',
  `state` VARCHAR(2) NOT NULL COMMENT '',
  UNIQUE INDEX `addressID_UNIQUE` (`addressID` ASC)  COMMENT '',
  PRIMARY KEY (`street`, `city`, `zip`, `state`)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`PhoneEmailIdentifier`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`PhoneEmailIdentifier` (
  `phoneEmailID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `phoneNumber` VARCHAR(10) NOT NULL COMMENT '',
  `emailAddress` VARCHAR(45) NOT NULL COMMENT '',
  PRIMARY KEY (`phoneNumber`, `emailAddress`)  COMMENT '',
  UNIQUE INDEX `phoneEmailID_UNIQUE` (`phoneEmailID` ASC)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Mailing`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Mailing` (
  `mailingID` INT UNSIGNED NULL AUTO_INCREMENT COMMENT '',
```

```
  `addressID` INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '',
  `POBox` VARCHAR(30) NOT NULL DEFAULT '' COMMENT '',
  PRIMARY KEY (`addressID`, `POBox`)  COMMENT '',
  UNIQUE INDEX `AddressPOBoxID_UNIQUE` (`mailingID` ASC)  COMMENT '',
  CONSTRAINT `mailing_address`
    FOREIGN KEY (`addressID`)
    REFERENCES `occ`.`Address` (`addressID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Organization`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Organization` (
  `organizationID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `officialName` VARCHAR(31) NOT NULL COMMENT '',
  `addressID` INT UNSIGNED NOT NULL COMMENT '',
  `mailingID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  `phoneEmailID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  `hours` VARCHAR(45) NULL DEFAULT '' COMMENT '',
  `website` VARCHAR(45) NULL DEFAULT '' COMMENT '',
  `church` TINYINT(1) NOT NULL DEFAULT 0 COMMENT '',
  PRIMARY KEY (`officialName`, `addressID`)  COMMENT '',
  UNIQUE INDEX `churchID_UNIQUE` (`organizationID` ASC)  COMMENT '',
  INDEX `organization_addressFK_idx` (`addressID` ASC)  COMMENT '',
  INDEX `organization_phoneEmailFK_idx` (`phoneEmailID` ASC)  COMMENT '',
```

```
  INDEX `organization_mailingFK_idx` (`mailingID` ASC)  COMMENT '',
  CONSTRAINT `organization_addressFK`
    FOREIGN KEY (`addressID`)
    REFERENCES `occ`.`Address` (`addressID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `organization_phoneEmailFK`
    FOREIGN KEY (`phoneEmailID`)
    REFERENCES `occ`.`PhoneEmailIdentifier` (`phoneEmailID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `organization_mailingFK`
    FOREIGN KEY (`mailingID`)
    REFERENCES `occ`.`Mailing` (`addressID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Person`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Person` (
  `personID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `firstName` VARCHAR(45) NOT NULL COMMENT '',
  `middleName` VARCHAR(45) NOT NULL DEFAULT '' COMMENT '',
  `lastName` VARCHAR(45) NOT NULL DEFAULT '' COMMENT '',
  `suffix` VARCHAR(45) NOT NULL DEFAULT '' COMMENT '',
```

```
  `phoneEmailID` INT UNSIGNED NOT NULL DEFAULT 1 COMMENT '',
  `addressID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  `mailingID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  PRIMARY KEY (`firstName`, `middleName`, `lastName`, `suffix`, `phoneEmailID`)  COMMENT '',
  INDEX `person_addressFK_idx` (`addressID` ASC)  COMMENT '',
  UNIQUE INDEX `personID_UNIQUE` (`personID` ASC)  COMMENT '',
  INDEX `person_phoneEmailIdentifierID_idx` (`phoneEmailID` ASC)  COMMENT '',
  INDEX `person_mailingFK_idx` (`mailingID` ASC)  COMMENT '',
  CONSTRAINT `person_addressFK`
    FOREIGN KEY (`addressID`)
    REFERENCES `occ`.`Address` (`addressID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `person_phoneEmailIdentifierID`
    FOREIGN KEY (`phoneEmailID`)
    REFERENCES `occ`.`PhoneEmailIdentifier` (`phoneEmailID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `person_mailingFK`
    FOREIGN KEY (`mailingID`)
    REFERENCES `occ`.`Mailing` (`mailingID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Pastor`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Pastor` (
  `personID` INT UNSIGNED NOT NULL COMMENT '',
  `organizationID` INT UNSIGNED NOT NULL COMMENT '',
  `position` TEXT(45) NULL COMMENT '',
  PRIMARY KEY (`personID`, `organizationID`)  COMMENT '',
  INDEX `personID_idx` (`personID` ASC)  COMMENT '',
  INDEX `pastor_organizationFK_idx` (`organizationID` ASC)  COMMENT '',
  CONSTRAINT `pastor_personFK`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `pastor_organizationFK`
    FOREIGN KEY (`organizationID`)
    REFERENCES `occ`.`Organization` (`organizationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`ProjectLeader`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`ProjectLeader` (
  `leaderID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `personID` INT UNSIGNED NOT NULL COMMENT '',
  `orgainzationID` INT UNSIGNED NOT NULL COMMENT '',
```

```
  `startdate` DATE NOT NULL COMMENT '',
  `enddate` DATE NULL COMMENT '',
  INDEX `personFK_idx` (`personID` ASC)  COMMENT '',
  INDEX `organizationFK_idx` (`orgainzationID` ASC)  COMMENT '',
  PRIMARY KEY (`personID`, `orgainzationID`, `startdate`)  COMMENT '',
  UNIQUE INDEX `leaderID_UNIQUE` (`leaderID` ASC)  COMMENT '',
  CONSTRAINT `project_personFK`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `organizationFK`
    FOREIGN KEY (`orgainzationID`)
    REFERENCES `occ`.`Organization` (`organizationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `occ`.`Region`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Region` (
  `regionID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `regionName` VARCHAR(45) NOT NULL COMMENT '',
  UNIQUE INDEX `regionID_UNIQUE` (`regionID` ASC)  COMMENT '',
  PRIMARY KEY (`regionName`)  COMMENT '')
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `occ`.`Team`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Team` (
  `teamID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `area` VARCHAR(45) NOT NULL COMMENT '',
  `regionID` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`teamID`)  COMMENT '',
  INDEX `team_regionFK_idx` (`regionID` ASC)  COMMENT '',
  CONSTRAINT `team_regionFK`
    FOREIGN KEY (`regionID`)
    REFERENCES `occ`.`Region` (`regionID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Coordinate`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Coordinate` (
  `coordinateID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `latitude` VARCHAR(45) NOT NULL COMMENT '',
  `longitude` VARCHAR(45) NOT NULL COMMENT '',
  UNIQUE INDEX `coordinatesID_UNIQUE` (`coordinateID` ASC)  COMMENT '',
  PRIMARY KEY (`latitude`, `longitude`)  COMMENT '')
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`DropOffLocation`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`DropOffLocation` (
  `locationID` VARCHAR(6) NOT NULL COMMENT '',
  `organizationID` INT UNSIGNED NOT NULL COMMENT '',
  `teamID` INT UNSIGNED NOT NULL COMMENT '',
  `coordinateID` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`teamID`, `organizationID`, `locationID`)  COMMENT '',
  UNIQUE INDEX `locationID_UNIQUE` (`locationID` ASC)  COMMENT '',
  INDEX `dropOffLocation_coordinateFK_idx` (`coordinateID` ASC)  COMMENT '',
  CONSTRAINT `dropOffLocation_organizationFK`
    FOREIGN KEY (`organizationID`)
    REFERENCES `occ`.`Organization` (`organizationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `dropOffLocation_teamFK`
    FOREIGN KEY (`teamID`)
    REFERENCES `occ`.`Team` (`teamID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `dropOffLocation_coordinateFK`
    FOREIGN KEY (`coordinateID`)
    REFERENCES `occ`.`Coordinate` (`coordinateID`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `occ`.`DropOff`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`DropOff` (
  `dropOffID` INT UNSIGNED NOT NULL COMMENT '',
  `personID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  `organizationID` INT UNSIGNED NULL DEFAULT 1 COMMENT '',
  `locationID` VARCHAR(6) NOT NULL COMMENT '',
  `boxes` INT NOT NULL COMMENT '',
  `time` DATETIME NOT NULL COMMENT '',
  PRIMARY KEY (`dropOffID`)  COMMENT '',
  INDEX `personID_idx` (`personID` ASC)  COMMENT '',
  INDEX `dropOff_locationFK_idx1` (`organizationID` ASC)  COMMENT '',
  INDEX `dropOff_locationFK_idx` (`locationID` ASC)  COMMENT '',
  CONSTRAINT `dropOff_personFK`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `dropOff_locationFK`
    FOREIGN KEY (`locationID`)
    REFERENCES `occ`.`DropOffLocation` (`locationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
```

```
  CONSTRAINT `dropOff_organizationFK`
    FOREIGN KEY (`organizationID`)
    REFERENCES `occ`.`Organization` (`organizationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `occ`.`Volunteer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Volunteer` (
  `userName` VARCHAR(100) NOT NULL COMMENT '',
  `personID` INT UNSIGNED NOT NULL COMMENT '',
  `password` VARCHAR(45) NOT NULL COMMENT '',
  `admin` TINYINT(1) NULL DEFAULT 0 COMMENT '',
  PRIMARY KEY (`personID`)  COMMENT '',
  INDEX `personID_idx` (`personID` ASC)  COMMENT '',
  UNIQUE INDEX `personID_UNIQUE` (`personID` ASC)  COMMENT '',
  UNIQUE INDEX `userName_UNIQUE` (`userName` ASC)  COMMENT '',
  CONSTRAINT `personID`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `occ`.`VolunteerTime`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`VolunteerTime` (
  `startTime` DATETIME NOT NULL COMMENT '',
  `personID` INT UNSIGNED NOT NULL COMMENT '',
  `endTime` DATETIME NOT NULL COMMENT '',
  `locationID` VARCHAR(6) NOT NULL COMMENT '',
  PRIMARY KEY (`personID`, `startTime`)  COMMENT '',
  INDEX `volunteerTime_locationFK_idx` (`locationID` ASC)  COMMENT '',
  CONSTRAINT `volunteerTime_locationFK`
    FOREIGN KEY (`locationID`)
    REFERENCES `occ`.`DropOffLocation` (`locationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `volunteerTime_personFK`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`Event`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`Event` (
  `eventID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
```

```
  `eventName` VARCHAR(45) NOT NULL COMMENT '',
  `description` VARCHAR(1000) NULL COMMENT '',
  `date` DATE NOT NULL COMMENT '',
  `startTime` TIME NOT NULL COMMENT '',
  `endTime` TIME NOT NULL COMMENT '',
  `organizationID` INT UNSIGNED NOT NULL COMMENT '',
  `image` BLOB NULL COMMENT '',
  `coordinateID` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`eventID`)  COMMENT '',
  INDEX `event_organizationFK_idx` (`organizationID` ASC)  COMMENT '',
  INDEX `event_coordinateFK_idx` (`coordinateID` ASC)  COMMENT '',
  CONSTRAINT `event_organizationFK`
    FOREIGN KEY (`organizationID`)
    REFERENCES `occ`.`Organization` (`organizationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `event_coordinateFK`
    FOREIGN KEY (`coordinateID`)
    REFERENCES `occ`.`Coordinate` (`coordinateID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`EventAttendee`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`EventAttendee` (
```

```
  `eventID` INT UNSIGNED NOT NULL COMMENT '',
  `personID` INT UNSIGNED NOT NULL COMMENT '',
  PRIMARY KEY (`eventID`, `personID`)  COMMENT '',
  INDEX `eventAttendee_personFK_idx` (`personID` ASC)  COMMENT '',
  CONSTRAINT `event_eventAttendeeFK`
    FOREIGN KEY (`eventID`)
    REFERENCES `occ`.`Event` (`eventID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `eventAttendee_personFK`
    FOREIGN KEY (`personID`)
    REFERENCES `occ`.`Person` (`personID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `occ`.`TeamMember`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`TeamMember` (
  `userName` VARCHAR(100) NOT NULL COMMENT '',
  `teamID` INT UNSIGNED NOT NULL COMMENT '',
  `ministryTeam` VARCHAR(45) NOT NULL COMMENT '',
  `position` VARCHAR(45) NOT NULL COMMENT '',
  PRIMARY KEY (`userName`, `teamID`)  COMMENT '',
  INDEX `volunteerTeam_team_idx` (`teamID` ASC)  COMMENT '',
  CONSTRAINT `teamMember_volunteer`
```

```
    FOREIGN KEY (`userName`)
    REFERENCES `occ`.`Volunteer` (`userName`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `teamMember_team`
    FOREIGN KEY (`teamID`)
    REFERENCES `occ`.`Team` (`teamID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `occ`.`RegionalAreaCoordinator`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`RegionalAreaCoordinator` (
  `userName` VARCHAR(100) NOT NULL COMMENT '',
  `regionID` INT UNSIGNED NOT NULL COMMENT '',
  PRIMARY KEY (`userName`, `regionID`)  COMMENT '',
  INDEX `regionalAreaCoordinator_region_idx` (`regionID` ASC)  COMMENT '',
  CONSTRAINT `regionalAreaCoordinator_volunteer`
    FOREIGN KEY (`userName`)
    REFERENCES `occ`.`Volunteer` (`userName`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `regionalAreaCoordinator_region`
    FOREIGN KEY (`regionID`)
    REFERENCES `occ`.`Region` (`regionID`)
```

```
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;


USE `occ` ;


-- -----------------------------------------------------
-- Placeholder table for view `occ`.`PersonData`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`PersonData` (`lastName` INT, `firstName` INT, `middleName` INT, `suffix` INT, `street` INT, `city` INT, `zip` INT, `state` INT, `phoneNumber` INT, `emailAddress` INT);


-- -----------------------------------------------------
-- Placeholder table for view `occ`.`OrganizationData`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`OrganizationData` (`officialName` INT, `street` INT, `city` INT, `zip` INT, `state` INT, `phoneNumber` INT, `emailAddress` INT, `hours` INT, `website` INT);


-- -----------------------------------------------------
-- Placeholder table for view `occ`.`EventData`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `occ`.`EventData` (`eventID` INT, `eventName` INT, `description` INT, `date` INT, `startTime` INT, `endTime` INT, `image` INT, `latitude` INT, `longitude` INT, `officialName` INT, `street` INT, `city` INT, `zip` INT, `state` INT);


-- -----------------------------------------------------
-- procedure insertperson
-- -----------------------------------------------------
```

```
DELIMITER $$

USE `occ`$$

CREATE PROCEDURE `insertperson`(IN firstNamein varchar(45), IN middlenamein varchar(45), IN
lastNamein varchar(45), IN suffixin varchar(45), IN streetin varchar(45), IN cityin varchar(40), IN
zipIn varchar(5), IN statein varchar(2), IN phonein varchar(10), IN emailin varchar(45))

BEGIN

INSERT IGNORE INTO Address(street, city, zip, state) VALUES(streetin, cityin, zipin, statein);

INSERT IGNORE INTO PhoneEmailIdentifier(phoneNumber, emailAddress) values(phonein,
emailin);

INSERT INTO Person(firstName, lastName, middleName, suffix, addressID, phoneEmailID)

SELECT firstNamein, lastNamein, middlenamein, suffixin, Address.addressID,
PhoneEmailIdentifier.phoneEmailID

FROM Address, PhoneEmailIdentifier

WHERE Address.street = streetin

AND Address.city = cityin

AND Address.zip = zipIn

AND Address.state = statein

AND PhoneEmailIdentifier.phoneNumber = phonein

AND PhoneEmailIdentifier.emailAddress = emailin

ON DUPLICATE KEY UPDATE Person.addressid = (SELECT addressID FROM Address WHERE
Address.street = streetin

AND Address.city = cityin

AND Address.zip = zipIn

AND Address.state = statein);


END

$$


DELIMITER ;
```

```
-- -----------------------------------------------------

-- procedure insertAttendee

-- -----------------------------------------------------


DELIMITER $$

USE `occ`$$

CREATE PROCEDURE `insertAttendee`(IN firstNamein varchar(45), IN middlenamein varchar(45), IN lastNamein varchar(45), IN suffixin varchar(45), IN streetin varchar(45), IN cityin varchar(40), IN zipin varchar(5), IN statein varchar(2), IN phonein varchar(10), IN emailin varchar(45), IN eventin INT)

BEGIN

INSERT IGNORE INTO Address(street, city, zip, state) VALUES(streetin, cityin, zipin, statein);

INSERT IGNORE INTO PhoneEmailIdentifier(phoneNumber, emailAddress) values(phonein, emailin);

INSERT INTO Person(firstName, lastName, middleName, suffix, addressID, phoneEmailID)

SELECT firstNamein, lastNamein, middlenamein, suffixin, Address.addressID, PhoneEmailIdentifier.phoneEmailID

FROM Address, PhoneEmailIdentifier

WHERE Address.street = streetin

AND Address.city = cityin

AND Address.zip = zipin

AND Address.state = statein

AND PhoneEmailIdentifier.phoneNumber = phonein

AND PhoneEmailIdentifier.emailAddress = emailin

ON DUPLICATE KEY UPDATE Person.addressid = (SELECT addressID FROM Address WHERE Address.street = streetin

AND Address.city = cityin

AND Address.zip = zipIn

AND Address.state = statein);

INSERT INTO EventAttendee(eventID, personID) SELECT eventin, Person.personID
```

```
FROM Event, Person WHERE Person.firstName = firstNamein
AND Person.lastName = lastNamein
AND Person.middleName = middlenamein
AND Person.suffix = suffixin
AND Person.phoneEmailID = (SELECT phoneEmailID FROM PhoneEmailIdentifier WHERE
PhoneEmailIdentifier.phoneNumber = phonein AND PhoneEmailIdentifier.emailAddress = emailin);



END
$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure createAccount
-- -----------------------------------------------------

DELIMITER $$
USE `occ`$$
CREATE PROCEDURE `createAccount` ()
BEGIN

END
$$

DELIMITER ;

-- -----------------------------------------------------
-- View `occ`.`PersonData`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `occ`.`PersonData`;
USE `occ`;
CREATE  OR REPLACE VIEW `PersonData` AS
   SELECT
      Person.lastName,
      Person.firstName,
      Person.middleName,
      Person.suffix,
      Address.street,
      Address.city,
      Address.zip,
      Address.state,
      PhoneEmailIdentifier.phoneNumber,
      PhoneEmailIdentifier.emailAddress
   FROM
      Person
         LEFT JOIN
      Address ON (Person.addressID = Address.addressID)
         LEFT JOIN
      PhoneEmailIdentifier ON (Person.phoneEmailID = PhoneEmailIdentifier.phoneEmailID);


-- -----------------------------------------------------
-- View `occ`.`OrganizationData`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `occ`.`OrganizationData`;
USE `occ`;
CREATE  OR REPLACE VIEW `OrganizationData` AS
```

```
   SELECT
      Organization.officialName,
      Address.street,
      Address.city,
      Address.zip,
      Address.state,
      PhoneEmailIdentifier.phoneNumber,
      PhoneEmailIdentifier.emailAddress,
      Organization.hours,
      Organization.website
   FROM
      Organization
         LEFT JOIN
      Address ON (Address.addressID = Organization.addressID)
         LEFT JOIN
      PhoneEmailIdentifier ON (PhoneEmailIdentifier.phoneEmailID = Organization.phoneEmailID)
;

-- ----------------------------------------------------
-- View `occ`.`EventData`
-- ----------------------------------------------------
DROP TABLE IF EXISTS `occ`.`EventData`;
USE `occ`;
CREATE  OR REPLACE VIEW `EventData` AS
   SELECT
      Event.eventID,
      Event.eventName,
      Event.description,
```

```
    Event.date,
    Event.startTime,
    Event.endTime,
    Event.image,
    Coordinate.latitude,
    Coordinate.longitude,
    Organization.officialName,
    Address.street,
    Address.city,
    Address.zip,
    Address.state
  FROM
    Event
      LEFT JOIN
    Coordinate ON (Event.coordinateID = Coordinate.coordinateID)
      LEFT JOIN
    Organization ON (Event.organizationID = Organization.organizationID)
      LEFT JOIN
    Address ON (Organization.addressID = Address.addressID);
USE `occ`;

DELIMITER $$
USE `occ`$$
CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Address_BEFORE_INSERT` BEFORE
INSERT ON `Address` FOR EACH ROW
BEGIN
if new.street = '' then
signal sqlstate '45000';
ELSEIF new.city = '' then
```

```
signal sqlstate '45000';
ELSEIF new.zip = '' then
signal sqlstate '45000';
ELSEIF new.state = '' then
signal sqlstate '45000';
end if;
END
$$

USE `occ`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`occ`.`PhoneEmailIdentifier_BEFORE_INSERT` BEFORE INSERT ON `PhoneEmailIdentifier` FOR
EACH ROW
BEGIN
if new.phoneNumber = '' AND new.emailAddress = '' then
signal sqlstate '45000';
end if;
END
$$

USE `occ`$$
CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Mailing_BEFORE_INSERT` BEFORE
INSERT ON `Mailing` FOR EACH ROW
BEGIN
if new.addressID = 1 AND new.POBox = '' then
signal sqlstate '45000';
end if;
END
$$
```

```
USE `occ`$$

CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Organization_BEFORE_INSERT`
BEFORE INSERT ON `Organization` FOR EACH ROW

BEGIN

if new.officialName = '' then

signal sqlstate '45000';

end if;

END

$$


USE `occ`$$

CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Person_BEFORE_INSERT` BEFORE
INSERT ON `Person` FOR EACH ROW

BEGIN

if new.firstName = '' then

signal sqlstate '45000';

ELSEIF new.lastName = '' then

signal sqlstate '45000';

end if;

END

$$


USE `occ`$$

CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Coordinate_BEFORE_INSERT`
BEFORE INSERT ON `Coordinate` FOR EACH ROW

BEGIN

if new.latitude = '' then

signal sqlstate '45000';
```

```
ELSEIF new.longitude = '' then
signal sqlstate '45000';
END IF;
END
$$

USE `occ`$$
CREATE DEFINER = CURRENT_USER TRIGGER `occ`.`Volunteer_BEFORE_INSERT` BEFORE INSERT ON `Volunteer` FOR EACH ROW
BEGIN
if new.userName = '' then
signal sqlstate '45000';
ELSEIF new.password = '' then
signal sqlstate '45000';
end if;
END
$$


DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```