Diary Management System

Database Management
CMPT 308-210

Team Sweet Dreams

Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Fall 2023

Project Report of Diary Management System

Team Name
Team Sweet Dreams

Project
Diary Management System

Team Members
1. Evan Spillane ..... Evan.Spillane1@marist.edu (Team Head)
2. Connor Fleischman ..... Connor.Fleischman1@marist.edu (Team Member)
3. Lilli Cartiera ..... Lilliana.Cartiera1@marist.edu (Team Member)
4. Abel Scholl ..... Anna.Scholl1@marist.edu (Team Member)
5. Neo Pi .....Neo.Pi1@marist.edu (Team Member)

Description of Team Members
1. Evan Spillane

I am a dual major in Cybersecurity and Philosophy, and think they balance each other out very well. Computer Science is based on math which is a field based on concrete answers whereas Philosophy is a field that lacks any concrete answers. I am from Valley Forge, PA (which is a little bit outside Philadelphia). Our group consists of the people sitting in the front/front-left of the classroom but we quickly got acclimated with each other. Lilli is my best friend (hence why they were sitting next to me in class) and I also lived with Neo last year, and he is in the Esports program with me. I volunteered to lead the group because I am confident in my project management and communication skills. I think my experience as 2025 Class president as well as the president of Marist Esports, and the Chair of the Rules & Administration Committee will be useful in helping my group thrive and do our best work!

2. Connor Fleischman

I am a Computer Science major from Wingdale, NY. I've got a little brother, Shane, who's currently a sophomore in high school and the captain of the varsity soccer team. My love of coding originated when I was little. Although opening the Command Prompt may not seem like much, as a ten-year-old it made me feel like a spy. Choosing my group was easy as all I did was turn around and ask. But the hard part came when we

tried to think up a name for us. It took a while but after many failed attempts we finally came up with one that fit. Sweet Dreams, brought up by Abel initially, was a perfect fit as by 9:00 p.m. on a Monday, everyone just wants to go home and have sweet dreams. As for the decision of the team head, Evan stepped up to fill the role and the group unanimously agreed. Along with this, we also chose the task of creating a Diary Management system as our project which I, along with the rest of the group, look forward to working on.

3. Lilli Cartiera

I am an Applied Math and Data Science double major from Wethersfield, Connecticut. I enjoy my major because making sense of the world using numbers is my strength. In my free time, I play piano and video games with my friends. On campus, I am the Vice President of the Math Club and I represent Marist's chapter of AWM (American Women in Mathematics). I am best friends with Evan and therefore I know he is a great leader. To me, he was an obvious choice. He is active on campus and passionate about creating tangible positive change. Our team was drawn together because we all sat near each other in class. Now that I know them better I look forward to working with them; so far they have taken great initiative and are enthusiastic about this project.

4. Abel Scholl

I am a Computer Science major from Wilkes-Barre, PA. I enjoy coding, puzzle games, and anything art-related, especially drawing and crocheting. I approach my major like an artist and let the creation aspect of writing code drive me to learn as much as I can and apply it wherever I can. This group is made up of people who were sitting closest to me when it came time to choose groups, but the collective effort to actively include everyone is what stood out to me about everyone here. It is easy to ignore and avoid what is unfamiliar or to prematurely judge based on initial perception, so I am looking forward to working together with the different personalities and perspectives brought to the table. Furthermore, I believe Evan will be a good team head judging by the way he volunteered. To me, this shows confidence in management which is essential for a good leader.

5. Neo Pi

I am a Cybersecurity major born and raised in South Jersey. Growing up, I always wanted to be a doctor and did not think I would learn to code until my freshman year of college. I chose this as my major over medicine because growing up as a kid I was always interested in video games and what a computer can do. The simplest way to choose a group was to team up with people around you, which led me to join this group. I

chose Evan as the team head because I know based on past experience that he is well-organized, intelligent, and capable of being a group leader.

Table of Contents

Project Report of Diary Management System.... 2
Table of Contents.... 5
Table of Figures.... 7
Project Objective.... 8
Related Work.... 9
Project Merits.... 10
GitHub Repository.... 10
Entity Relationship Model.... 11
Enhanced Entity Relationship Model.... 19
Figure 2: Enhanced Entity Relationship Model (EER).... 19
SQL Data Type Slideshow Presentation.... 21
SQL Demos.... 22
Figure 3: integerAndOverflowExample.sql.... 22
Figure 4: integerExample.sql.... 23
Figure 5: Project_Phase_03_DateTimeTypes.sql.... 24
Figure 6: Project_Phase_03_FractionalSeconds.sql.... 25
Figure 7: otherDataTypes.sql.... 26
Database Development.... 27
Figure 7: Creating the Database.... 27
Figure 8: Creating the Colors Table.... 27
Figure 9: Creating the ActivityStatus Table.... 28
Figure 10: Creating the Roles Table.... 28
Figure 11: Creating the Users Table.... 29
Figure 12: Creating the Locations Table.... 29
Figure 13: Creating the EntryTypes and Diaries Tables.... 30
Figure 14: Creating the Organizations Table.... 31
Figure 15: Creating the Entries Table.... 32
Figure 16: Creating the Permissions Table.... 33
Figure 17: Creating the RolePermissions Table.... 33
Figure 18: Creating the OrganizationMembers Table.... 34
Figure 19: Creating the UserDiaries Table.... 34
Figure 20: Creating the UserPermissions Table.... 34
Loading Data and Performance Enhancements.... 35
Inserting Data & Data Optimization.... 39
Normalization Check.... 40
Application Development.... 41

Graphical User Experience Design....41
Connection to DB....44
Login Page....44
Figure 23: Login Page GUI....44
Figure 24: Login Access Granted....44
Main Menu Page....46
Figure 25: Main Menu Page GUI....47
Action Pages....47
i. Search....47
Figure 26: Search GUI....47
ii. Insertion....47
Figure 27: Adding a New Diary....48
iii. Modification....48
Figure 28: Editing a Diary....48
iv. Deletion....48
v. Print All....48
References....63

Table of Figures

1. Figure 1: Entity Relationship Model (ER)……..…11
2. Figure 2: Enhanced Entity Relationship Model (EER)…….……18
3. Figure 3: integerAndOverflowExample.sql…….....21
4. Figure 4: integerExample.sql……22
5. Figure 5: Project_Phase_03_DateTimeTypes.sql…….23
6. Figure 6: Project_Phase_03_FractionalSeconds.sql……..……24
7. Figure 6: otherNumericTypes.sql…….……..……25
8. Figure 7: Creating the Database…….26
9. Figure 8: Creating the Colors Table……...26
10. Figure 9: Creating the ActivityStatus Table……...27
11. Figure 10: Creating the Roles Table ……..…….27
12. Figure 11: Creating the Users Table
……..……28
13. Figure 12: Creating the Locations Table……28
14. Figure 13: Creating the Entry Types and Diaries Tables……...29
15. Figure 14: Creating the Organizations Table…….30
16. Figure 15: Creating the Entries Table……31
17. Figure 16: Creating the Permissions Table……32
18. Figure 17: Creating the RolePermissions Table…….32
19. Figure 18: Creating the OrganizationMembers Table……33
20. Figure 19: Creating the UserDiaries Table
……33
21. Figure 20: Creating the UserPermissionsTable……..33
22. Figure 21 Login Flowchart…….……..40
23. Figure 22 Main Menu, Settings, Edit, Diaries Flowchart……..41
24. Figure

Project Objective

The purpose of the Diary Management System is to create a Diary Management System GUI using Python, the Tkinter library, and SQL. This system should allow a user to record daily events and experiences and store the data in separate SQL tables. It will include separate administrator and normal user logins with different role-based permissions that allow different levels of access to features. Users should be able to add, edit, and delete records as well as search for records by time, place, or duration. Finally, the Diary Management System will also have settings where the admin can moderate users and their information, user-friendly warnings and alerts, a feature menu, and an exit function.

Related Work

Businesses related to our Project

Business 1: EventReference[1]

Positives: Efficient ideas, organized concept

Negatives: UI is too complicated for new users to navigate

Business 2: Setmore[2]

Positives: Extremely friendly UI, many different tools available

Negatives: Does not have multi-user data management features in certain versions

Business 3: Client Diary[3]

Positives: Large scale with many tools (especially in its premium version)

Negatives: Expensive, some tools may be useless to users with less complex data

The diary system can have multiple fields for each employee. These fields can include job title, country, company details, organization type, and various questions that are relevant to the diary system's purpose. For example: what type of events do you organize? What is your annual budget for meetings and events? Which products or services are you interested in for your meetings and events? What locations are you interested in for your meetings and events?

A diary can also be used for staff to message each other and set up meetings. The diary can also be used to match buyers with suppliers based on their individual preferences. Each party can select those they are interested in meeting with. A ranking system can also be utilized.

Lastly, the information each user puts into the diary can be used to create a schedule of meetings throughout a period of time. We can do this by examining each user's rankings of other users. The timing and frequency of these meetings can be planned based on this information.

[1] See References; Item 1
[2] See References; Item 2
[3] See References; Item 3

Project Merits

Adopting our Diary Management System is an ideal choice for any small-scale company that struggles to manage team schedules, plan meetings, or organize events. Whether the entry is a meeting, special event, time away, reminder, or just a simple note, keeping track of these details with your team helps take the hassle out of planning and keeps both work and life running smoothly. Sort your team into organizations, assign roles to members, color code entries, and share diaries to help make teamwork as productive, efficient, and stressless as possible. Our diary management system is also made easy for new users to learn and navigate, featuring a comprehensive menu of options for convenient reference. If you are looking for a free and easy way to manage your team's time, choose Sweet Dreams Diary Management System.

GitHub Repository

https://github.com/CainSpillane/201_DiaryManagementSystem_SweetDreams

Entity Relationship Model

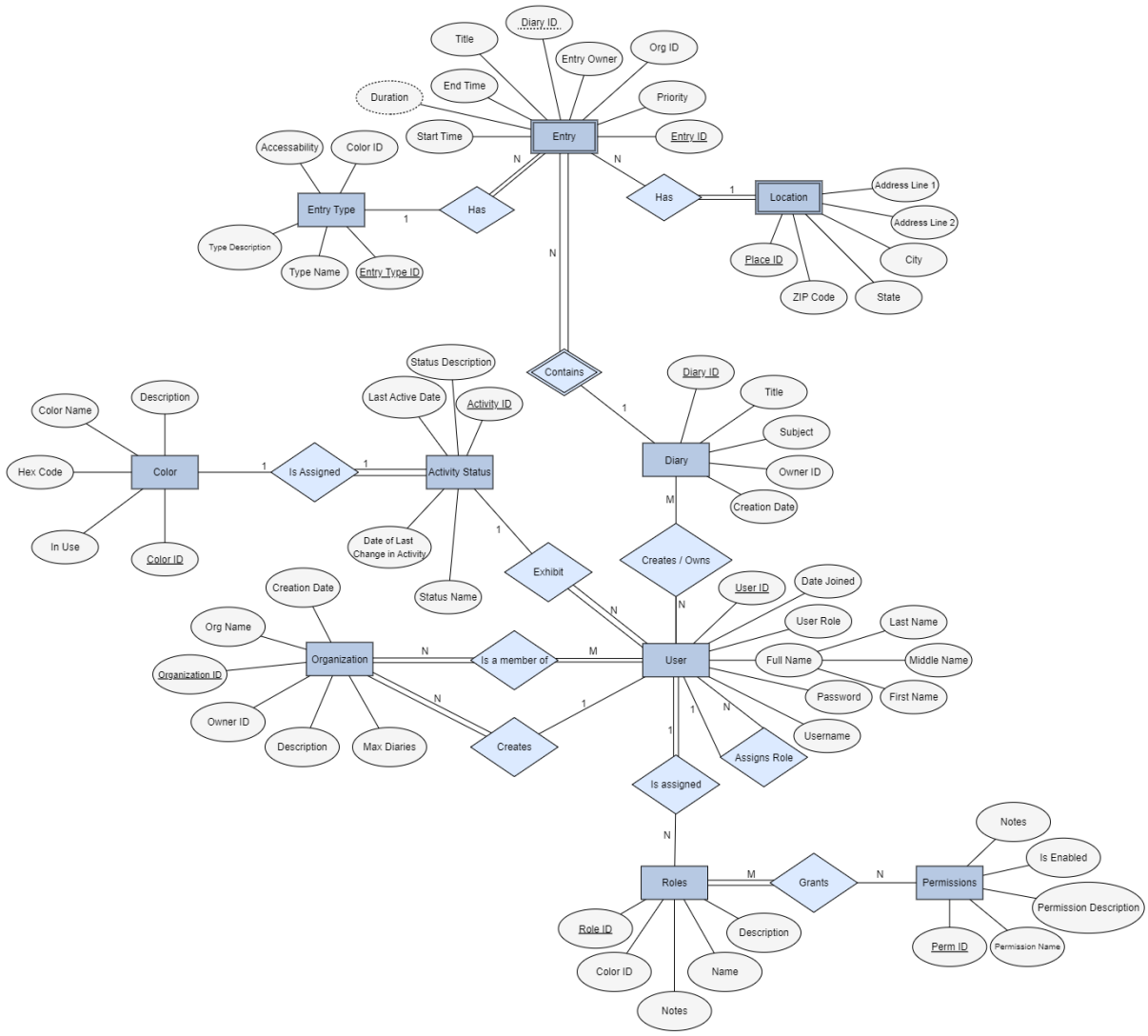


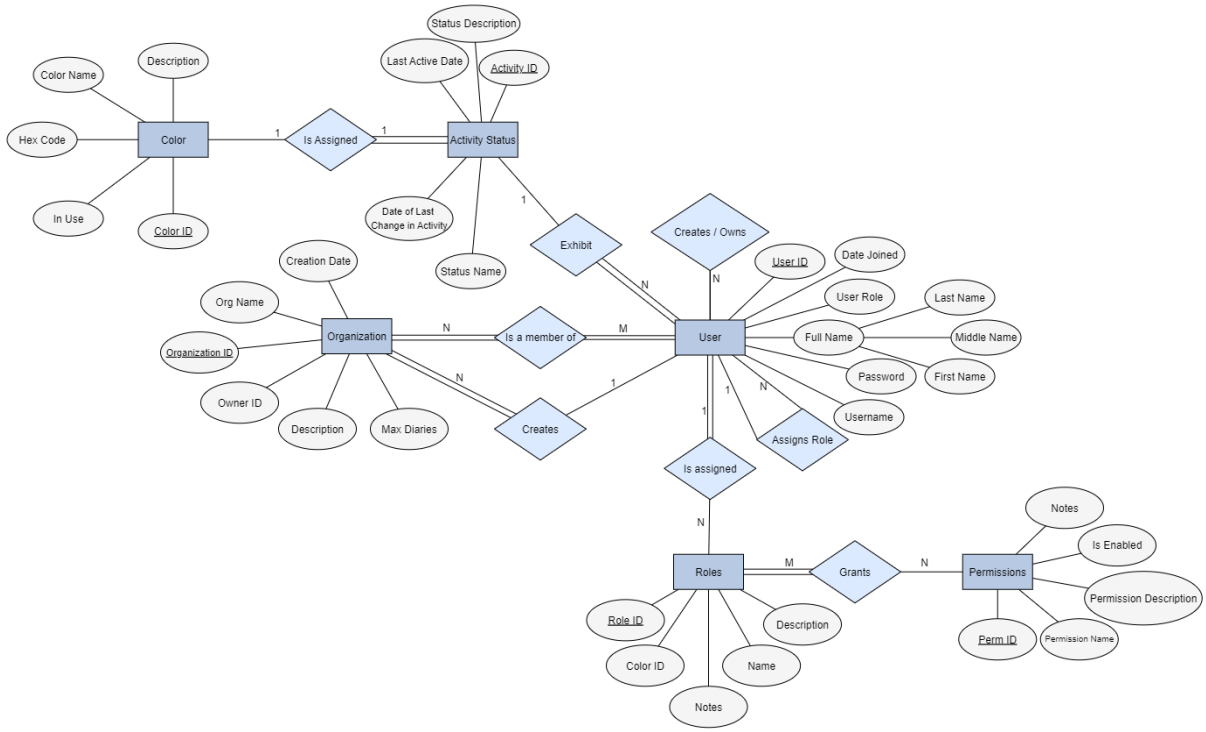
Figure 1: Entity Relationship Model (ER)
Model made using yEd Live (yworks.com)

To display the mini-world we selected entities that relate to each other. First, we selected a user entity that represents individuals who can access the system. This entity is a central part of the mini-world as it interacts with other entities. This was a starting point for us. Then, we added a list of applicable attributes to help lay out what each entity can do. For example, in user, the attributes contained important information: full name, User ID, user role, username, and password. To be more specific, the full name is split into more attributes: first, middle, and last. Then, we represented the relationships of users with a “create/owns action”, “is assigned”, and “is a member of”. With the use of these relationships, we can create further entities by extending

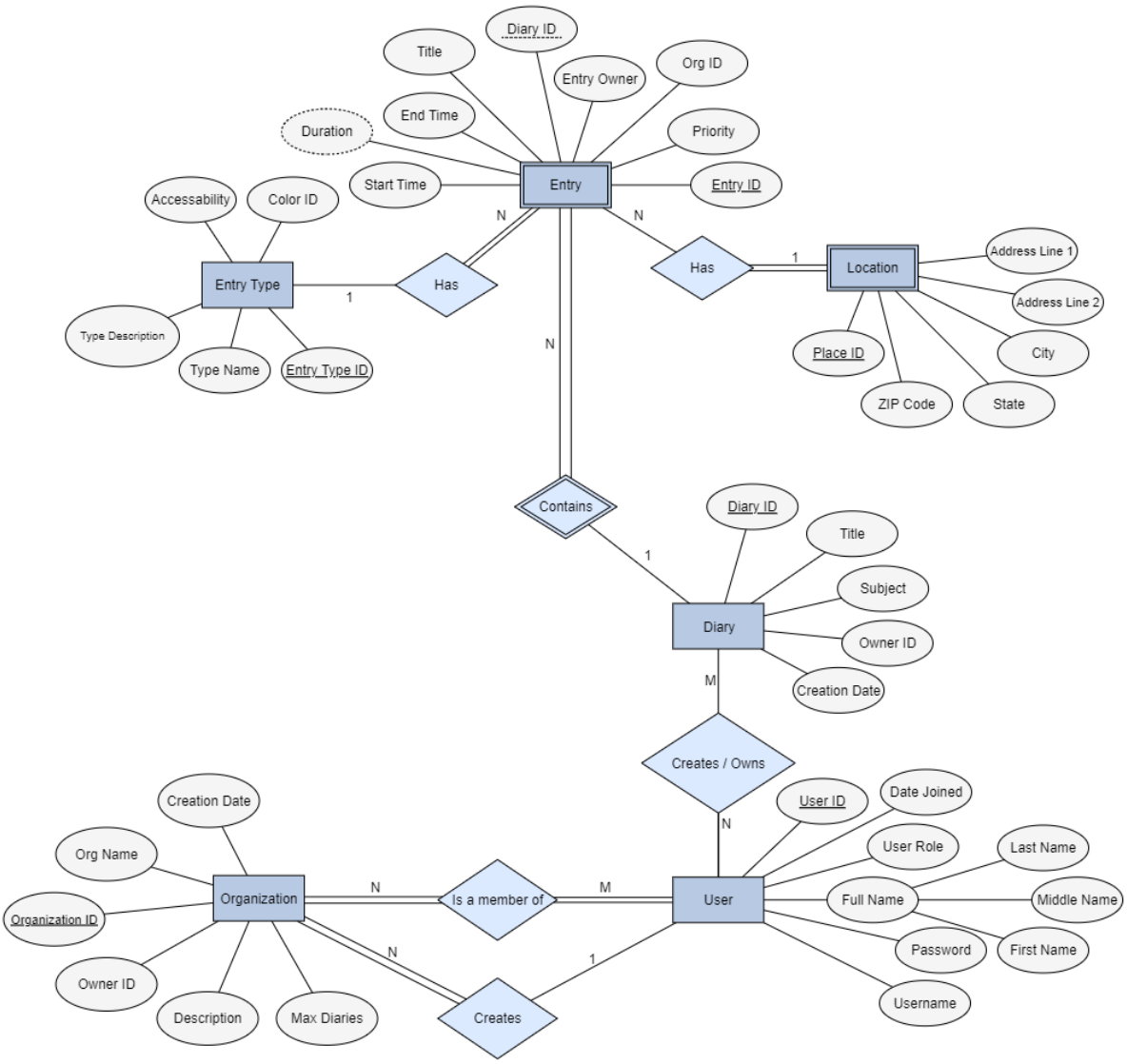
connecting participations. For example, “users” are related to “organizations” by the relationship “is a member of”. Since all users have to be members in order to have access, we get an instance of total participation. Likewise, all members are required to be in an organization. Then we related this organization entity to further entities: shared diary, diary, and entries. Determining cardinalities is based on the relationship between two entities. For example, the cardinality of users to organizations is many to many. Finally, we created another entity that can branch off the first, and input more attributes and relationships. By starting from a user entity, we were able to break down the diary management system.

Listed below are important details and descriptions of all of the entities, as well as their attributes, relationships, and cardinality.

Views that contribute to ER Model



Admin View



User View

User

- Description
  - A representation of a user for the purpose of interacting with the DMS.
- Attributes
  - User ID
  - Full Name (Multivalued Attribute)
    - First Name
    - Middle Name
    - Last Name

    - User Role – determines the permissions a user has
    - Date Joined – the date the user's information was created by the admin
    - Username
    - Password
- Relationships
    - Users can create diaries
    - Users (Administrator) can assign roles to users; This relationship is recursive
    - Users are a member of an organization
    - Users (Administrators) create Organizations
- Cardinality
    - All users are members of an organization, which makes this a many-to-many (M:N) relationship
    - The cardinality for users creating diaries is 1:N
    - Users can create an organization, which is a 1:N Relationship
    - Users assign roles to users in a 1:N relationship

Organization

- Description
    - An organization is an entity that has 1 or more user members and can be used to have shared diaries and shared entries with users that belong to the same organization. For example, this could be used by a company to have their employees have a shared diary space, or by a family who wants to have a shared diary space
- Attributes
    - Organization ID
    - Organization Name
    - Owner ID
    - Creation Date
    - Diary Limit
- Relationships
    - Organizations contain users
    - Organizations are created by users
- Cardinality
    - Organization has an M:N Relationship with Users
    - Organization has a N:1 relationship with user. Any organization is created by only a single user

Diary

- Description
    - The diary entity is used to house and organize multiple user inputs while keeping them separate based on certain criteria. Diaries contain Entries, which contain the

user's textual inputs to the DMS. Diaries can also be sorted and organized based on dates, subjects, and titles

- Attributes
    - Diary ID
    - Owner ID
    - Title
    - Subject
    - Creation Date
- Relationships
    - Diaries must be created by a user
    - Diaries contain Entries
- Cardinality
    - Diary has an M:N relationship with User
    - Diary has a 1:N relationship with Entry

Entry

- Description
    - Entry is a weak entity that contains data about the entries within a given diary
- Attributes
    - Entry ID
    - Diary ID
    - Entry Owner
    - Organization ID
    - Priority
    - Start Time
    - End Time
    - Duration
- Relationships
    - An Entry must have a Diary in which it is contained.
    - Entries must have an Entry Type
    - Entries can have a Location
- Cardinality
    - Entry has an N:1 relationship with Diary
    - Entry has an N:1 relationship with Entry Type
    - Entry has an N:1 relationship with Location

Entry Type

- Description

    - Entry type is an entity that establishes characteristics that are applied to an Entry. All Entries must have an entry type.
- Attributes
    - Entry Type ID
    - Entry Type Name
    - Entry Type Description
    - Accessibility
    - Color ID
- Relationships
    - Entry Type Can be associated with an entry
    - Entry Type is assigned a color
- Cardinality
    - Has a 1:N relationship with Entry
    - Has a 1:1 relationship with Color

Location

- Description
    - Location is a weak entity that contains information about the place of a given entry, should a user wish to input one
- Attributes
    - Location ID
    - Address Line 1
    - Address Line 2
    - City
    - State
    - ZIP Code
- Relationships
    - Location is associated with an Entry
- Cardinality
    - Location has a 1:N relationship with Entry

Role

- Description
    - Role is an entity that determines levels of system privilege for a given user. One example of a role is Administrator, and Admins can assign roles to other users
- Attributes
    - Role Name
    - Role Description
    - Color ID

    - Role Notes
- Relationships
    - A Role is assigned to a user by another user (administrator)
    - Roles grant permissions (from the Permission entity) to the role recipient based on the ID of the Role that was assigned
- Cardinality
    - Has an N:1 relationship with User
    - Has an N:M relationship with Permission

Permission

- Description
    - Permission is an entity that dictates rules regarding what a user is capable of doing on the DMS. Permissions are associated with a role, and roles are granted to users. Users then inherit the permissions from their role, so long as they still have that role.
- Attributes
    - Permission ID
    - Permission Name
    - Permission Description
    - Is Enabled
        - Is Enabled is a boolean value determining whether an individual permission is enabled within a role.
    - Notes
- Relationships
    - Permissions are granted by Roles
- Cardinality
    - Has an N:M relationship with Roles

Activity Status

- Description
    - Activity Status is an entity that displays information about a user's recent activity (or lack thereof) on the DMS.
- Attributes
    - Activity ID
    - Status Description
    - Last Active Date
    - Status Name
    - Date of last change in activity
- Relationships

    - Activity Status is exhibited by a user
    - Activity Status is assigned a color based on which type of status it is.
- Cardinality
    - Has a 1:1 relationship with Color
    - Has a 1:N relationship with User

Color

- Description
    - Color is an entity that contains information about the color and associated properties, which is used to help differentiate between different types of entries, statuses, and roles.
- Attributes
    - Color ID
    - Hex Code
    - Color Name
    - Description
    - In use
- Relationships
    - Color is assigned to the Entry Type entity
    - Color is assigned to the Activity Status entity
    - Color is assigned to the Role entity
- Cardinality
    - Has a 1:1 relationship with Entry Type
    - Has a 1:1 relationship with Activity Status
    - Has a 1:1 relationship with Roles

Enhanced Entity Relationship Model

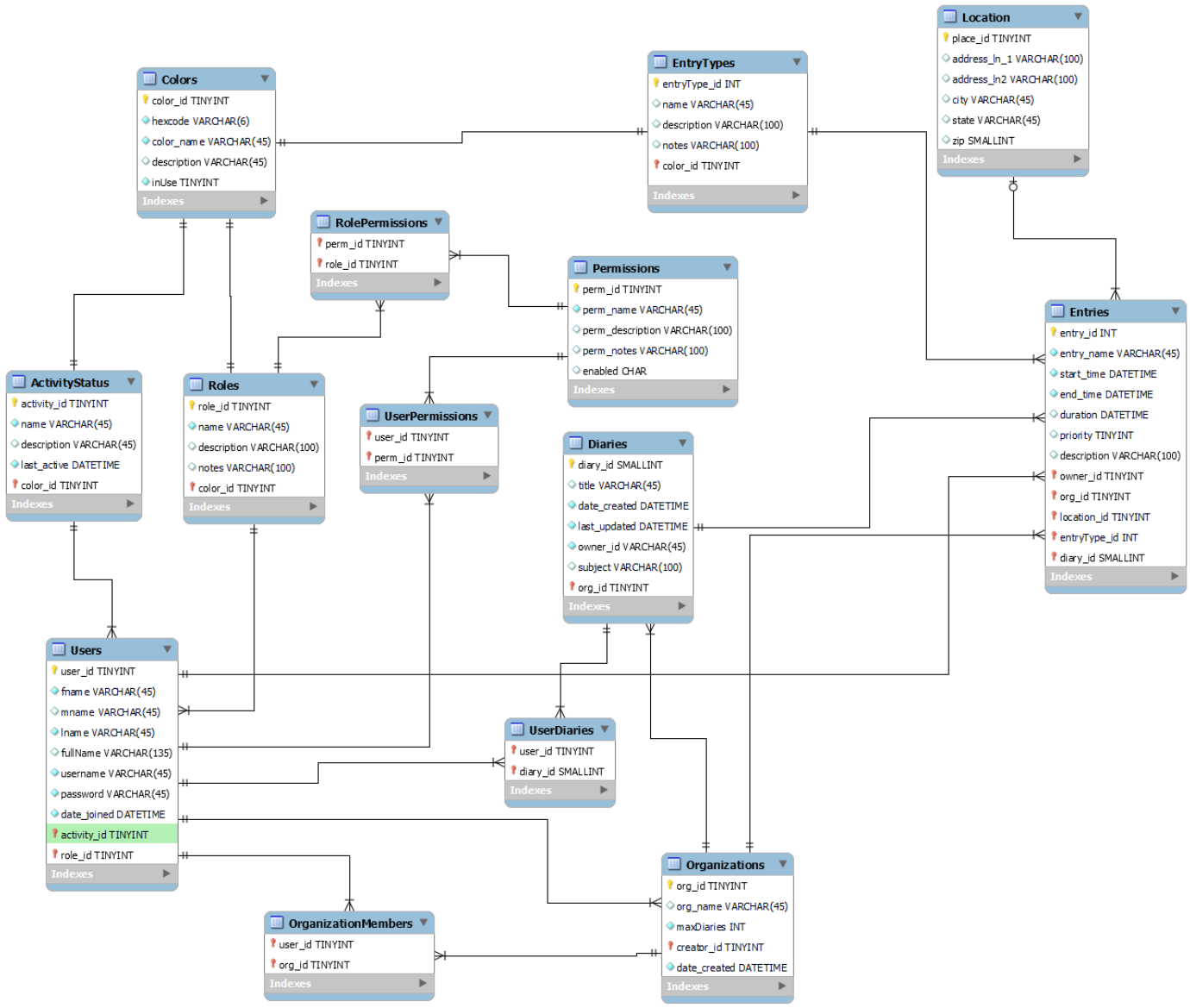


Figure 2: Enhanced Entity Relationship Model (EER)

In an EER model, keys help uniquely identify entities as their information is needed across the database. These keys help keep track of relationships between different entities without duplicating data across tables. In the DMS EER model, the entities Users, ActivityStatus, Roles, Permissions, Organizations, Diaries, Entries, Location, Colors, and EntryTypes all have specific IDs that act as primary keys within their own table and foreign keys within other tables with their data associated. For instance, Users has a primary key user_id that is used as a foreign key within Entries (shown as “owner” in Entries table) in order to establish a one-to-many relationship between Users and Entries where one user can own many entries.

Users also has foreign keys from ActivityStatus, and Roles, because every user has associated information relating to these entities; Every user must have one activity status and one role, but no more at a single time, making the relationships one to many. Though User is related to ActivityStatus, Role, and Entries, Entries does not carry a foreign key of any entity in this list except for Users, because an entry does not need to know anything about a user except which specific user it is. Some relationships, such as Users to Diaries, have many-to-many relationships. In this case, One user can have access to many diaries, and a diary can be accessed by many users. Rather than have Users carry the key of every single diary it has access to, another table is implemented that holds only the primary keys of Users and Diaries and keeps track of every combination as necessary. Finally, other different types of relationships include one-to-one and zero-or-one-to-many. The relationship between ActivityStatus and Colors is one-to-one, because one status can be associated with only one color and vice versa. The relationship between Entries and Location is zero-or-one-to-many, because a location can be attached to more than one Entry at a time, but not every entry has to have a location. If it is associated with a location, it can have only one. The other relationships within the model are all handled in the same ways described above and are as follows:

- ActivityStatus to Users: one to many
- Roles to Users: one to many
- Users to Entries: one to many
- Users to Organizations: many to many
- Users to Diaries: many to many
- Roles to Permissions: many to many
- Colors to ActivityStatus: one to one
- Colors to Roles: one to one
- Colors to EntryTypes: one-to-one
- EntryTypes to Entries: one to many
- Location to Entries: zero or one to many
- Organizations to Entries: one to many
- Diaries to Entries: one to many
- Organizations to Diaries: one to many

SQL Data Type Slideshow Presentation

We chose to present information on the Numeric data types as well as the Date and Time data types. Described first, numeric data types consist of integer types (which include TINYINT, SMALLINT MEDIUMINT, INTEGER, and BIGINT), bit type (BIT), fixed-point type (DECIMAL), and floating-point types (FLOAT, DOUBLE). The presentation highlights the syntax of each type, applicable parameters, maximum and minimum values, as well as its appropriate usage. We also touch on out-of-range values and how MySql responds to these cases depending on its current mode. After Numeric types we introduce Date and Time types, including DATE, DATETIME, TIMESTAMP, TIME, and YEAR. We also cover the SQL calendar and conversions as well as syntax, general format, and behavior determined by MYSQL.

SQL Demos

Figure 3: integerAndOverflowExample.sql

```sql
-- Integer types and overflow example using TINYINT
-- The following code will create a table with two TINYINT fields, one signed and the other unsigned.
-- This example demonstrates the range of values that the TINYINT data type can hold
-- and MYSQL's way of dealing with results of out-of-range values.

CREATE TABLE TinyIntExample (
    tinyIntsSigned TINYINT,
    tinyIntsUnsigned TINYINT UNSIGNED
);
-- Signed TINYINT Example
INSERT INTO TinyIntExample (tinyIntsSigned) VALUES (0), (127), (-128); -- max values within range limits
-- These next two lines will throw errors, becasue the values being inserted are out of range.
INSERT INTO TinyIntExample (tinyIntsSigned) VALUES (128); -- Out of Range Value
INSERT INTO TinyIntExample (tinyIntsSigned) VALUES (-129); -- Out of Range Value

-- show all signed tinyint data in the table
SELECT tinyIntsSigned FROM TinyIntExample WHERE tinyIntsSigned IS NOT NULL;

-- Unsigned TINYINT Example
INSERT INTO TinyIntExample (tinyIntsUnsigned) VALUES (0), (255); -- max values within range limits
-- These next two lines will throw errors, becasue the values being inserted are out of range.
INSERT INTO TinyIntExample (tinyIntsUnsigned) VALUES (-1); -- Out of Range Value
INSERT INTO TinyIntExample (tinyIntsUnsigned) VALUES (256); -- Out of Range Value

-- show all unsigned tinyint data in the table
SELECT tinyIntsUnsigned FROM TinyIntExample WHERE tinyIntsUnsigned IS NOT NULL;

SET sql_mode = 'TRADITIONAL'; -- this line will enable Strict SQL mode
SET sql_mode = ''; -- this line will change the sql mode to unrestricted
SET sql_mode = 'NO_UNSIGNED_SUBTRACTION'; -- this line will change the sql mode to NO_UNSIGNED_SUBTRACTION

-- The result of the following line will be different depending on what mode is enabled.
-- Subtraction between signed and unsigned numbers is unsigned by default.
SELECT CAST(2 AS UNSIGNED) - 5; -- an unsigned value subtracted by a signed value with a negative result

 DROP TABLE TinyIntExample; -- used to delete the table if necessary for starting over
```

Figure 4: integerExample.sql

```sql
-- Comprehensive Example using all integer data types
-- -- The following code creates a table of all integer types, both signed and unsigned
-- -- in order to demonstrate the value ranges of each type.
CREATE TABLE IntegerExample (
    tinyIntsSigned TINYINT,
    tinyIntsUnsigned TINYINT UNSIGNED,
    smallIntsSigned SMALLINT,
    smallIntsUnsigned SMALLINT UNSIGNED,
    mediumIntsSigned MEDIUMINT,
    mediumIntsUnsigned MEDIUMINT UNSIGNED,
    intsSigned INT,
    intsUnsigned INT UNSIGNED,
    bigIntsSigned BIGINT,
    bigIntsUnsigned BIGINT UNSIGNED
);

-- insert upper bound values for each field.
INSERT INTO IntegerExample (tinyIntsSigned, tinyIntsUnsigned, smallIntsSigned, smallIntsUnsigned,
mediumIntsSigned, mediumIntsUnsigned, intsSigned, intsUnsigned, bigIntsSigned, bigIntsUnsigned)
    VALUES (127,255,32767,65535,8388607,16777215,2147483647,4294967295,9223372036854775807,18446744073709551615);
-- insert zero, a lower bound value for unsigned types and middle bound for signed types.
INSERT INTO IntegerExample (tinyIntsSigned, tinyIntsUnsigned, smallIntsSigned, smallIntsUnsigned,
mediumIntsSigned, mediumIntsUnsigned, intsSigned, intsUnsigned, bigIntsSigned, bigIntsUnsigned)
    VALUES (0,0,0,0,0,0,0,0,0,0);
-- insert lower bound values for signed types only (because negative values are out of range for unsigned types)
INSERT INTO IntegerExample (tinyIntsSigned, smallIntsSigned, mediumIntsSigned, intsSigned, bigIntsSigned)
    VALUES (-128,-32768,-8388608,-2147483648,-9223372036854775808);
-- display all data in the IntegerExample table for a full list of value ranges per integer data type
SELECT * FROM IntegerExample;

 DROP TABLE IntegerExample; -- used to delete the table if necessary for starting over
```

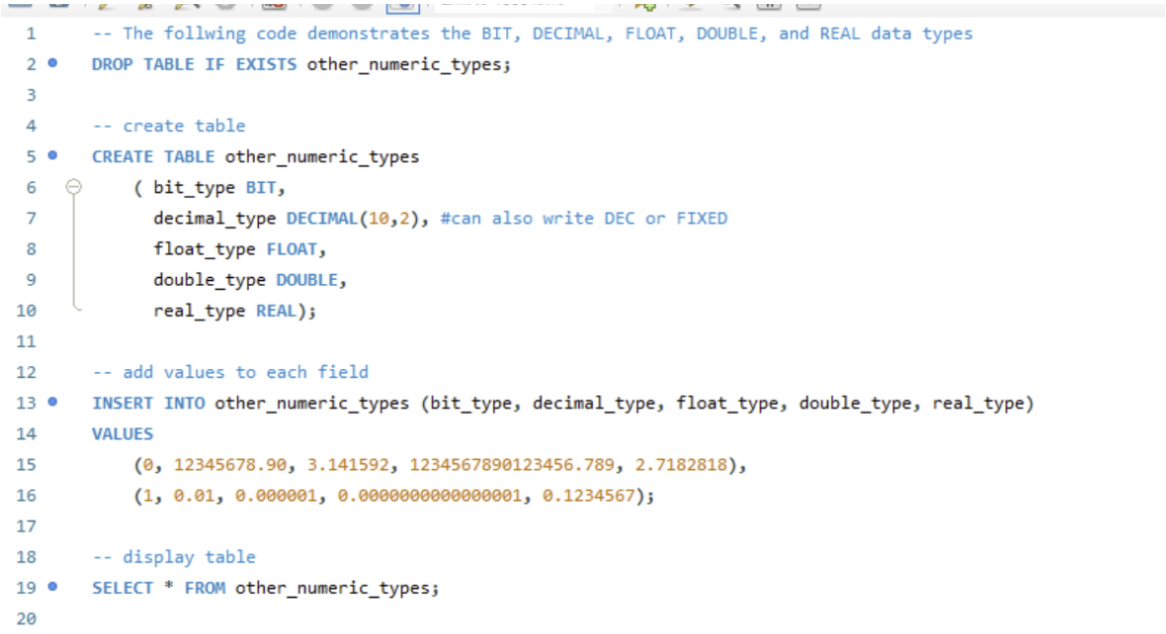
Figure 5: Project_Phase_03_DateTimeTypes.sql

```sql
-- Date and Time types and Auto-initialization/Auto-updating

-- Creates a table showing the given date, time, year, as well as the current DATETIME and TIMESTAMP
use project_phase_03;
create table DATEandTIME (
    CurrentDATE DATE,
    CurrentTIME TIME,
    CurrentYEAR YEAR,
    CurrentDATETIME DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP,
    CurrentTIMESTAMP TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP
);

-- Inserts data into the generated table with each piece of data coresponding to its respective column
insert into DATEandTIME (CurrentDate, CurrentTIME, CurrentYEAR, CurrentDATETIME, CurrentTIMESTAMP)
    values ('2023-10-09', '16:39:55.143665', '2023', NOW(),NOW());

-- Shows the table
select * from DATEandTIME;

-- Removes the table (for reuseability)
-- drop table DATEandTIME;
```

Figure 6: Project_Phase_03_FractionalSeconds.sql

```sql
-- Fractional Seconds

-- This bit of code stops MySQL from truncating the fractional part of time
SET @@sql_mode = sys.list_add(
@@sql_mode, 'TIME_TRUNCATE_FRACTIONAL'
);

-- Creates a table showing a given time and rounding that time to a specified length
use project_phase_03;
Create table FractionalSeconds (
    CurrentTIME TIME,
    CurrentTIME6 TIME(6),
    CurrentTIME5 TIME(5),
    CurrentTIME4 TIME(4),
    CurrentTIME3 TIME(3),
    CurrentTIME2 TIME(2),
    CurrentTIME1 TIME(1),
    CurrentTIME0 TIME(0)
);

-- Inserts data into the generated table with each piece of data coresponding to its respective column
insert into FractionalSeconds (CurrentTIME, CurrentTIME6, CurrentTIME5, CurrentTIME4,
    CurrentTIME3, CurrentTIME2, CurrentTIME1, CurrentTIME0)
    values('16:39.143665', '16:39:55.143665', '16:39:55.143665', '16:39:55.143665',
    '16:39:55.143665', '16:39:55.143665', '16:39:55.143665', '16:39:55.143665');

-- Shows the table
select * from FractionalSeconds;

-- Removes the table (for reuseability)
-- drop table FractionalSeconds;
```

Figure 7: otherDataTypes.sql



```sql
-- The follwing code demonstrates the BIT, DECIMAL, FLOAT, DOUBLE, and REAL data types
DROP TABLE IF EXISTS other_numeric_types;

-- create table
CREATE TABLE other_numeric_types
    ( bit_type BIT,
      decimal_type DECIMAL(10,2), #can also write DEC or FIXED
      float_type FLOAT,
      double_type DOUBLE,
      real_type REAL);

-- add values to each field
INSERT INTO other_numeric_types (bit_type, decimal_type, float_type, double_type, real_type)
VALUES
    (0, 12345678.90, 3.141592, 1234567890123456.789, 2.7182818),
    (1, 0.01, 0.000001, 0.0000000000000001, 0.1234567);

-- display table
SELECT * FROM other_numeric_types;

```

Database Development

```
1   -- -----------------------------------------------------
2   --  teamSweetDreams_DMS
3   -- -----------------------------------------------------
4
5   DROP DATABASE IF EXISTS teamSweetDreams_DMS;
6
7   -- -----------------------------------------------------
8   -- Create teamSweetDreams_DMS Database
9   -- -----------------------------------------------------
10  CREATE DATABASE IF NOT EXISTS teamSweetDreams_DMS;
11  USE teamSweetDreams_DMS;
12
```

Figure 7: Creating the Database

```
Description: Drops / Creates / Selects Schema
```

```
13  -- -----------------------------------------------------
14  -- Create the Colors Table
15  -- -----------------------------------------------------
16  DROP TABLE IF EXISTS teamSweetDreams_DMS.Colors;
17
18  CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.Colors(
19    color_id TINYINT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
20    hexcode VARCHAR(6) NOT NULL,
21    color_name VARCHAR(45) NULL,
22    description VARCHAR(45) NULL,
23    inUse BOOLEAN NOT NULL
24   );
25
```

Figure 8: Creating the Colors Table

```
Colors Table Description: Creates ‘Colors’
Attributes & Type: color_id(TINYINT UNSIGNED), hexcode(VARCHAR), color_name(VARCHAR),
       description(VARCHAR), inUse(BOOLEAN)
Primary Key: color_id
Relationships: ‘Roles’, ‘EntryTypes’
```

```
26      -- -----------------------------------------------------
27      -- Create the ActivityStatus Table
28      -- -----------------------------------------------------
29      DROP TABLE IF EXISTS teamSweetDreams_DMS.ActivityStatus;
30
31      CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.ActivityStatus(
32        activity_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
33        name VARCHAR(45) NOT NULL,
34        description VARCHAR(45) NULL,
35        last_assigned DATETIME NOT NULL,
36        color_id TINYINT  UNSIGNED,
37        PRIMARY KEY (activity_id),
38        FOREIGN KEY (color_id) REFERENCES teamSweetDreams_DMS.Colors (color_id)
39      );
40
```

Figure 9: Creating the ActivityStatus Table

```
Activity Status Table Description: Creates 'ActivityStatus'
Attributes & Type: activity_id(TINYINT UNSIGNED), name(VARCHAR),
       description(VARCHAR), last_assigned(DATETIME), color_id(TINYINT)
Primary Key: activity_id
Foreign Key: color_id
Relationship: 'Colors'
```

```
42      -- -----------------------------------------------------
43      -- Create Roles Table
44      -- -----------------------------------------------------
45      DROP TABLE IF EXISTS  teamSweetDreams_DMS.Roles;
46
47      CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.Roles(
48        role_id TINYINT NOT NULL,
49        name VARCHAR(45) NULL,
50        description VARCHAR(100) NULL,
51        notes VARCHAR(100) NULL,
52        color_id TINYINT NOT NULL,
53        PRIMARY KEY (role_id),
54        FOREIGN KEY (color_id) REFERENCES teamSweetDreams_DMS.Colors (color_id)
55      );
```

Figure 10: Creating the Roles Table

```
Roles Table Description: Creates 'Roles'
Attributes & Type: role_id(TINYINT), name(VARCHAR), description(VARCHAR),
       notes(VARCHAR), color_id(TINYINT)
Primary Key: role_id
Foreign Key: color_id
Relationships: 'Colors'
```

```
56      -- -----------------------------------------------------
57      -- Create Users Table
58      -- -----------------------------------------------------
59      DROP TABLE IF EXISTS teamSweetDreams_DMS.Users;
60
61      CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.Users (
62        user_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
63        fname VARCHAR(45) NOT NULL,
64        lname VARCHAR(45) NOT NULL,
65        fullName VARCHAR(135) GENERATED ALWAYS AS (concat(fname, ' ', lname)) STORED NOT NULL,
66        username VARCHAR(45) NOT NULL,
67        password VARCHAR(45) NOT NULL,
68        date_joined DATETIME NOT NULL,
69        activity_id TINYINT UNSIGNED NOT NULL,
70        role_id TINYINT UNSIGNED NOT NULL,
71        PRIMARY KEY (user_id),
72        FOREIGN KEY (activity_id) REFERENCES teamSweetDreams_DMS.ActivityStatus (activity_id),
73        FOREIGN KEY (role_id) REFERENCES teamSweetDreams_DMS.Roles (role_id)
74        );
```

Figure 11: Creating the Users Table

Users Table Description: Creates 'Users'
Attributes & Type: user_id(TINYINT UNSIGNED), fName(VARCHAR), lName(VARCHAR), fullName(VARCHAR), username(VARCHAR), password(VARCHAR), date_joined(DATETIME), activity_id(TINYINT), role_id(TINYINT)
Primary Key: user_id
Foreign Key: activity_id, role_id
Relationships: 'ActivityStatus', 'Roles'

```
76      -- -----------------------------------------------------
77      -- Create Location Table
78      -- -----------------------------------------------------
79      DROP TABLE IF EXISTS teamSweetDreams_DMS.Location;
80
81      CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.Location (
82        place_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
83        address_ln_1 VARCHAR(100) NULL,
84        address_ln2 VARCHAR(100) NULL,
85        city VARCHAR(45) NULL,
86        state VARCHAR(45) NULL,
87        zip VARCHAR(5) NULL,
88        PRIMARY KEY (place_id)
89        );
90
```

Figure 12: Creating the Locations Table

Location Table Description: Creates 'Location'
Attributes & Types: place_id(TINYINT UNSIGNED), address_ln_1(VARCHAR), address_ln2(VARCHAR), city(VARCHAR), state(VARCHAR), zip(VARCHAR)
Primary Key: place_id
Relationships: 'Entries'

```
106     -- -----------------------------------------------------
107     -- Create EntryTypes Table
108     -- -----------------------------------------------------
109     DROP TABLE IF EXISTS teamSweetDreams_DMS.EntryTypes;
110
111     CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.EntryTypes (
112       entryType_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
113       name VARCHAR(45) NULL,
114       description VARCHAR(100) NULL,
115       notes VARCHAR(100) NULL, -- -------------------
116       color_id TINYINT UNSIGNED NOT NULL,
117       PRIMARY KEY (entryType_id),
118       FOREIGN KEY (color_id) REFERENCES teamSweetDreams_DMS.Colors (color_id)
119       );
120
121     -- -----------------------------------------------------
122     -- Create Diaries Table
123     -- -----------------------------------------------------
124     DROP TABLE IF EXISTS  teamSweetDreams_DMS.Diaries;
125
126     CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.Diaries (
127       diary_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
128       title VARCHAR(45),
129       date_created DATETIME NOT NULL,
130       last_updated DATETIME NOT NULL,
131       owner_id TINYINT UNSIGNED NOT NULL,
132       subject VARCHAR(100) NULL,
133       org_id TINYINT UNSIGNED NOT NULL,
134       PRIMARY KEY (diary_id),
135       FOREIGN KEY (org_id) REFERENCES teamSweetDreams_DMS.Organizations (org_id),
136       FOREIGN KEY (owner_id) REFERENCES teamSweetDreams_DMS.Users (user_id)
137       );
```

Figure 13: Creating the EntryTypes and Diaries Tables

```
Entry Types Table Description: Creates ‘entryTypes’
Attributes & Type: entryType_id(INT UNSIGNED), name(VARCHAR), description(VARCHAR),
      notes(VARCHAR), color_id(TINYINT UNSIGNED)
Primary Key: entryType_id
Foreign Key: color_id
Relationships: ‘Colors’

Diaries Table Description: Creates ‘Diaries’
Attributes & Type: diary_id(SMALLINT UNSIGNED), title(VARCHAR),
      date_created(DATETIME), last_updated(DATETIME), owner_id(TINYINT UNSIGNED)
      subject(VARCHAR), org_id(TINYINT)
Primary Key: diary_id
Foreign Key: TINYINT
Relationships: ‘Organizations’, ‘Users’
```

```
91      -- -----------------------------------------------------
92      -- Table  teamSweetDreams_DMS.Organizations
93      -- -----------------------------------------------------
94      DROP TABLE IF EXISTS teamSweetDreams_DMS.Organizations;
95
96      CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.Organizations (
97        org_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
98        org_name VARCHAR(45) NULL,
99        maxDiaries INT NOT NULL,
100       creator_id TINYINT UNSIGNED NOT NULL,
101       date_created DATETIME NOT NULL,
102       PRIMARY KEY (org_id),
103       FOREIGN KEY (creator_id) REFERENCES teamSweetDreams_DMS.Users (user_id)
104       );
105
```

Figure 14: Creating the Organizations Table

```
Organizations Table Description: Creates 'Organizations'
Attributes & Type: org_id(TINYINT UNSIGNED), org_name(VARCHAR), maxDiaries(INT),
       creator_id(TINYINT UNSIGNED), date_created(DATETIME)
Primary Key: org_id
Foreign Key: creator_id
Relationships: 'Users'
```

```
139     -- -----------------------------------------------------
140     -- Create Entries Table
141     -- -----------------------------------------------------
142     DROP TABLE IF EXISTS  teamSweetDreams_DMS.Entries;
143
144     CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.Entries (
145       entry_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
146       entry_title VARCHAR(45) NOT NULL,
147       start_time DATETIME NOT NULL,
148       end_time DATETIME NOT NULL,
149       duration DATETIME,
150       priority TINYINT NULL,
151       description VARCHAR(100) NULL,
152       owner_id TINYINT UNSIGNED NOT NULL,
153       org_id TINYINT UNSIGNED NOT NULL,
154       location_id TINYINT UNSIGNED NULL,
155       entryType_id INT UNSIGNED NOT NULL,
156       diary_id SMALLINT UNSIGNED NOT NULL,
157       PRIMARY KEY (entry_id),
158       FOREIGN KEY (owner_id) REFERENCES teamSweetDreams_DMS.Users (user_id),
159       FOREIGN KEY (org_id) REFERENCES teamSweetDreams_DMS.Organizations (org_id),
160       FOREIGN KEY (location_id) REFERENCES teamSweetDreams_DMS.Location (place_id),
161       FOREIGN KEY (entryType_id) REFERENCES teamSweetDreams_DMS.EntryTypes (entryType_id),
162       FOREIGN KEY (diary_id) REFERENCES teamSweetDreams_DMS.Diaries (diary_id)
163       );
```

Figure 15: Creating the Entries Table

```
Entries Table Description: Creates ‘Entries’
Attributes & Type: entry_id(INT), entry_tytle(VARCHAR), start_time(VARCHAR),
       end_time(DATETIME), duration(DATETIME), priority(TINYINT),
       description(VARCHAR), owner_id(TINYINT), org_id(TINYINT),
       location_id(TINYINT), entryType_id(INT), diary_id(SMALLINT)
Primary Key: entry_id
Foreign Key: owner_id, org_id, location_id, entryType_id, diary_id
Relationships: ‘Users’, ‘Organizations’, ‘Location’, ‘EntryTypes’, ‘Diaries’
```

```
165     -- -----------------------------------------------------
166     -- Create Permissions Table
167     -- -----------------------------------------------------
168     DROP TABLE IF EXISTS  teamSweetDreams_DMS.Permissions;
169
170     CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.Permissions (
171       perm_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
172       perm_name VARCHAR(45) NOT NULL,
173       perm_description VARCHAR(100) NULL,
174       perm_notes VARCHAR(100) NULL,
175       enabled BOOLEAN NULL DEFAULT 0,
176       PRIMARY KEY (perm_id)
177     );
178
```

Figure 16: Creating the Permissions Table

Permissions Table Description: Creates ‘Permissions’
Attributes & Type: perm_id(TINYINT), perm_name(VARCHAR), perm_description(VARCHAR),
perm_notes(VARCHAR), enabled(CHAR)
Primary Key: perm_id
Relationships: Roles

```
179     -- -----------------------------------------------------
180     -- Create RolePermissions Table
181     -- -----------------------------------------------------
182     DROP TABLE IF EXISTS  teamSweetDreams_DMS.RolePermissions;
183
184     CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.RolePermissions (
185       perm_id TINYINT UNSIGNED NOT NULL,
186       role_id TINYINT UNSIGNED NOT NULL,
187       PRIMARY KEY (perm_id, role_id),
188       FOREIGN KEY (perm_id) REFERENCES teamSweetDreams_DMS.Permissions (perm_id) ON DELETE CASCADE, -- if id is deleted, will apply to related tables as well
189       FOREIGN KEY (role_id) REFERENCES teamSweetDreams_DMS.Roles (role_id) ON DELETE CASCADE
190       );
```

Figure 17: Creating the RolePermissions Table

Role Permissions Table Description: Creates ‘rolePermissions’
Attributes & Type: perm_id(TINYINT), role_id(TINYINT)
Foreign Keys: perm_id, role_id
Relationships: ‘Permissions’, ‘Roles’

```
192     -- -----------------------------------------------------
193     -- Create OrganizationMembers Table
194     -- -----------------------------------------------------
195 •   DROP TABLE IF EXISTS  teamSweetDreams_DMS.OrganizationMembers;
196
197 •   CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.OrganizationMembers (
198       user_id TINYINT UNSIGNED NOT NULL,
199       org_id TINYINT UNSIGNED NOT NULL,
200       PRIMARY KEY (user_id, org_id),
201       FOREIGN KEY (user_id) REFERENCES teamSweetDreams_DMS.Users (user_id) ON DELETE CASCADE,
202       FOREIGN KEY (org_id) REFERENCES teamSweetDreams_DMS.Organizations (org_id) ON DELETE CASCADE
203       );
```

Figure 18: Creating the OrganizationMembers Table

Organization Members Table Description: Creates ‘organizationMembers’
Attributes & Type: user_id(TINYINT), org_id(TINYINT)
Foreign Keys: user_id, org_id
Relationships: ‘Users’, ‘Organizations’

```
205     -- -----------------------------------------------------
206     -- Create UserDiaries Table
207     -- -----------------------------------------------------
208 •   DROP TABLE IF EXISTS  teamSweetDreams_DMS.UserDiaries;
209
210 •   CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.UserDiaries (
211       user_id TINYINT UNSIGNED NOT NULL,
212       diary_id SMALLINT UNSIGNED NOT NULL,
213       PRIMARY KEY (user_id, diary_id),
214       FOREIGN KEY (user_id) REFERENCES teamSweetDreams_DMS.Users (user_id) ON DELETE CASCADE,
215       FOREIGN KEY (diary_id) REFERENCES teamSweetDreams_DMS.Diaries (diary_id) ON DELETE CASCADE
216       );
```

Figure 19: Creating the UserDiaries Table

User Diaries Table Description: Creates ‘userDiaries’
Attributes & Type: user_id(TINYINT), diary_id(SMALLINT)
Foreign Keys: user_id, diary_id
Relationships: ‘Users’, ‘Diaries’

```
218     -- -----------------------------------------------------
219     -- Create UserPermissions Table
220     -- -----------------------------------------------------
221 •   DROP TABLE IF EXISTS  teamSweetDreams_DMS.userPermissions;
222
223 •   CREATE TABLE IF NOT EXISTS  teamSweetDreams_DMS.userPermissions (
224       user_id TINYINT UNSIGNED NOT NULL,
225       perm_id TINYINT UNSIGNED NOT NULL,
226       PRIMARY KEY (user_id, perm_id),
227       FOREIGN KEY (user_id) REFERENCES teamSweetDreams_DMS.Users (user_id) ON DELETE CASCADE,
228       FOREIGN KEY (perm_id) REFERENCES teamSweetDreams_DMS.Permissions (perm_id) ON DELETE CASCADE
229       );
```

Figure 20: Creating the UserPermissions Table

User Permissions Table Description: Creates ‘userPermissions’
Attributes & Type: user_id(TINYINT), perm_id(TINYINT)
Foreign Keys: user_id, perm_id
Relationships: ‘Users’, ‘Permissions’

Loading Data and Performance Enhancements

The following SQL demonstrates how we imported sample data into the tables we created in the previous section.

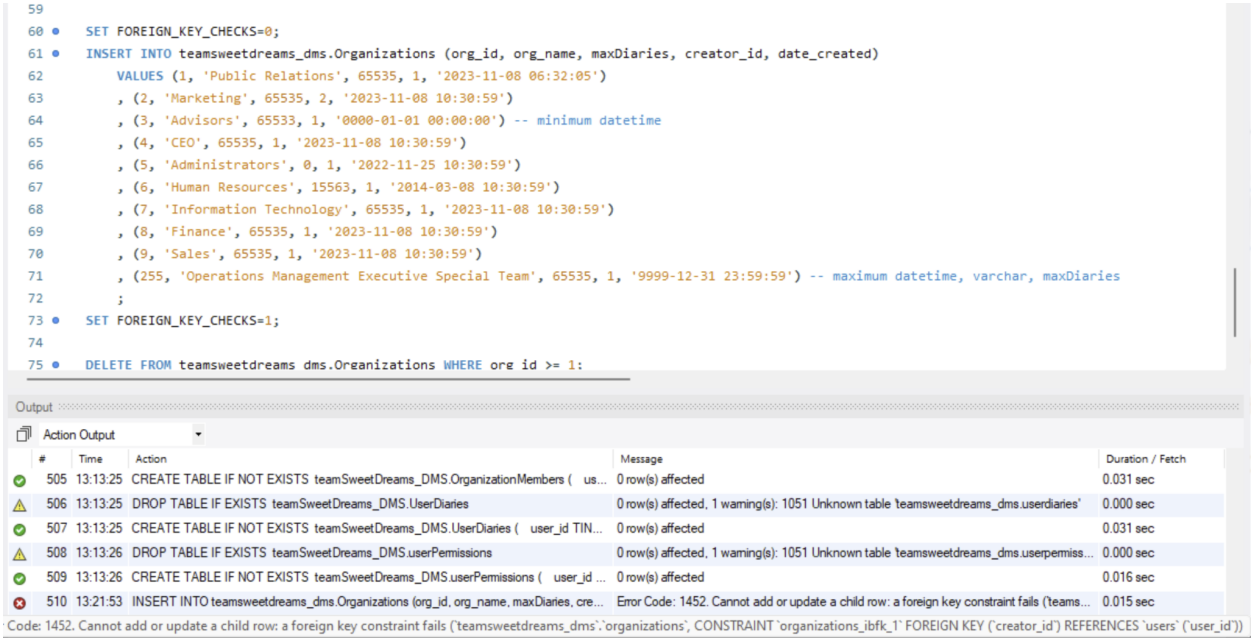
```
--select the database to use
USE teamsweetdreams_dms;

-- populate the Colors table
INSERT INTO teamsweetdreams_dms.Colors(color_id, hexcode, color_name, description, inUse)
    VALUES (1, 'ed5050', 'Red1', 'Used for upper management.', 1)
, (2, 'c18007', 'Orange1', 'Used for New Hires.', 1)
, (3, 'ead748', 'Yellow1', 'Used to denote Meetings.', 0)
, (4, 'c18007', 'Green1', NULL, 1)
, (5, '4285d1', 'Blueeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee', 'Used for New Hires.', 1) -- a color name with 45 characters
, (6, '6514b7', 'Purple1', 'Used for notes.', 1)
, (7, 'b2b2b2', 'Marist Gray', 'Official Marist Gray.', 1)
, (8, '2d9999', 'Jeff', 'Used for Jeff.', 1)
, (10, '2d9943', 'Poughkeepsie', 'Used for events in Poughkeepsie. weeeeeeeeee!', 1) -- a description with 45 characters
, (255, 'c8102e', 'Marist', 'Official Marist Red', 1);

-- populate the ActivityStatus table
INSERT INTO teamsweetdreams_dms.ActivityStatus(activity_id, name, description, last_assigned, color_id)
    VALUES (1, 'Online', 'User is Online', '2023-11-08 06:32:05', 4)
, (2, 'Offline', 'User is Offline', '9999-12-31 11:13:00', 255) -- maximum date
, (3, 'Idle', 'User is idle', '2010-02-28 21:04:00', 3)
, (4, 'Meeting', 'User is in a meeting', '2020-02-29 23:12:31', 4) -- leap year date
, (5, 'Away', 'User is away', '2023-11-08 23:59:59', 5) -- maximum time
, (6, 'lunch', 'User is on lunch', '2022-12-30 00:00:00', 6) -- minimum time
, (7, 'user has 45 characters 1 1 1 1 1 1 1 1 ! ! !!', 'This user has 45 characters 1 1 1 1 1 1 1 1 !', '2023-11-08 00:00:00', 7) -- maximum
varchar length
, (20, 'Jeff', 'User is Jeff', '0000-01-01 01:40:02', 8) -- minimum date
, (10, 'Poughkeepsie', 'User is in Poughkeepsie', '2023-05-08 23:01:0', 4)
, (255, 'Time-Out', 'This user is in timeout.', '2023-04-07 00:00:00', 10);

-- Populate the Location table
INSERT INTO teamsweetdreams_dms.Location(place_id, address_ln_1, address_ln2, city, state, zip)
    VALUES (1, '3399 North Road', 'Marist College MSC 11111', 'Poughkeepsie','New York', '12601') -- minimum ID
, (2, '47 Essex Ave', NULL, 'Westford', 'MA', '01886')
, (3, NULL, 'C. Library', 'Poughkeepsie', 'New York', '12601')
, (4, 'Greystone', NULL, NULL, NULL, '54701')
, (5, 'The weird room outside the new guys office name starts with a J', 'Someone please confirm', 'Glendale', '', NULL) -- Maximum varchar
length
, (6, '8724 Grant St.', NULL, 'Woodstock', 'Georgia', '01886')
, (7, 'Dining Hall', '', '', '', '')
, (8, '1600 Pennsylvania Ave', NULL, 'N.W.', 'Washington DC', '20500')
, (10, 'Arlington Street', 'Red Building Blue Fence', 'Los Banos', NULL, NULL)
, (255, 'Large Conference Room', NULL, NULL, NULL, NULL); -- maximum ID, NULL values

 -- Populate the Permissions table:
INSERT INTO teamsweetdreams_dms.permissions (perm_id, perm_name, perm_description, perm_notes, enabled) VALUES
 ('1', 'All', 'All permissions', 'Granted to role: HEAD ADMIN', 0)
, ('2', 'Alter', 'Grants the ability to alter tables', 'Granted to role: ADMIN', 0)
, ('3', 'Create', 'Grants the ability to create tables', 'Granted to role: ADMIN', 1)
, ('4', 'Drop', 'Grants the ability to drop tables', 'Granted to role: ADMIN, DELETER', 1)
, ('5', 'Grant', 'Grants the ability to grant other users permissions', 'Granted to role: SUPERVISOR', 1)
, ('6', 'Insert', 'Grants the ability to insert data into tables', 'Granted to role: ADMIN, INSERTER', 0)
, ('7', 'Spectate', 'Grants the ability to view only', 'Granted to role: VIEWER', 1)
, ('8', 'Recrute', 'Grants the ability to recrute', 'Granted to role: RECRUITER', 0)
, ('9', 'Godly', 'Grants the ability to delete the entire workspace', 'Granted to role: GOD', 0)
, ('10', 'Visitor', 'Grants the ability to have access to a workspace for a short time', 'Granted to role: TEMPORARY', 1);

-- Populate the Roles table:
INSERT INTO teamsweetdreams_dms.roles (role_id, name, description, color_id, notes) VALUES
 ('1', 'Head Admin', 'Gives all permissions to the user', 1, 'Used for the president of the company')
, ('2', 'Admin', 'Gives the roles of creating, deleting, changing tables and can edit the roles of others', 2, 'Used for higher level
personnel ')
, ('3', 'Viewer', 'Only allows the user to view data in and the columns of tables', 3, 'Used for personnel who should only be able to view
data')
, ('4', 'Inserter', 'Only allows the user to insert data into tables', 4, 'Used for personnel who should only be able to insert data')
, ('5', 'Deleter', 'Only allows the user to delete data in tables', 5, 'Used for personnel who should only be able to delete data')
, ('6', 'Supervisor', 'Allows for the user to change the permissions of users', 6, 'Used for managerial employees')
, ('7', 'Recruiter', 'Gives the user the ability to add new users to the workspace', 7, 'Used for recruiters')
, ('8', 'Restricted', 'Does not allow ANY permissions', 8, 'Used to restrict users from the database')
```

```
, ('9', 'God', 'Grants the God-like power to create and destroy the entirety of a workspace', 10, 'Used for God-like users')
, ('10', 'Temporary', 'Gives the user \'Viewer\' permissions for one day', 255, 'Used for users who should only see the data for a short
time');


-- Populate the Users table:
INSERT INTO teamsweetdreams_dms.users (user_id, fname, lname, username, password, activity_id, role_id, date_joined) VALUES
 ('1', 'Connor', 'Fleischman', 'waytofaded', 'Toothfairy', 5, 3, '2023-11-08 00:00:00')
, ('2', 'Evan', 'Spillane', 'EvanSpillane', 'Password', 2, 4, '2023-11-08 00:00:00')
, ('3', 'Lilli', 'Cartiera', 'LilliCartiera', 'Google123', 255, 7, '2023-11-08 00:00:00')
, ('4', 'Abel', 'Scholl', 'AbelScholl', 'BehindU', 10, 10, '2023-11-08 00:00:00')
, ('5', 'Neo', 'Pi', 'NeoPi', 'ABCDEFG123', 6, 1, '2023-11-08 00:00:00')
, ('6', 'Saul', 'Goodman', 'BetterCallSaul', 'LyrsAreUs', 1, 1, '2023-11-08 00:00:00')
, ('7', 'Anakin', 'Skywalker', 'theChosenOne', 'Jedi4Life', 1, 3, '2023-11-08 00:00:00')
, ('8', 'Neo', 'Anderson', 'MAtrixHakr', '123CantHakME', 20, 4, '2023-11-08 00:00:00')
, ('9', 'Jesus', 'Christ', 'SonofGod', 'Heaven', 2, 7, '2023-11-08 00:00:00')
, ('10', 'Finn', 'TheHuman', 'FinnTheHuman', 'ILoveJake', 1, 6, '2023-11-08 00:00:00');

--Populate Organizations table
INSERT INTO teamsweetdreams_dms.Organizations (org_id, org_name, maxDiaries, creator_id, date_created)
    VALUES (1, 'Public Relations', 65535, 1, '2023-11-08 06:32:05')
, (2, 'Marketing', 65535, 2, '2023-11-08 10:30:59')
, (3, 'Advisors', 65533, 1, '0000-01-01 00:00:00') -- minimum datetime
, (4, 'CEO', 65535, 1, '2023-11-08 10:30:59')
, (5, 'Administrators', 0, 1, '2022-11-25 10:30:59')
, (6, 'Human Resources', 15563, 1, '2014-03-08 10:30:59')
, (7, 'Information Technology', 65535, 1, '2023-11-08 10:30:59')
, (8, 'Finance', 65535, 1, '2023-11-08 10:30:59')
, (9, 'Sales', 65535, 1, '2023-11-08 10:30:59')
, (255, 'Operations Management Executive Special Team', 65535, 1, '9999-12-31 23:59:59') -- maximum datetime, varchar, maxDiaries
 ;

-- Populate the EntryTypes table:
INSERT INTO teamsweetdreams_dms.entrytypes (entryType_id, name, description, notes, color_id) VALUES
 ('1', 'Required', 'Everyone is required to attend', NULL, 1)
, ('2', 'High priority', 'Attendance is highly recommend but not required', NULL, 2)
, ('3', 'Priority', 'Attendance is recommended but not required', 'Use high/low priority for more pressing/ less important events', 3)
, ('4', 'Optional', 'Attendance is optional', NULL, 4)
, ('5', 'Special Event', 'Used for the least important entries', NULL, 5)
, ('6', 'Away', 'Denote predetermined times away from the office', 'This type will not notify the user', 3)
, ('7', 'Note', NULL, 'Leave a note to your team', 4)
, ('8', 'Put-off', 'Used for entries whose event will be put-off until later', NULL, 5)
, ('9', 'Heads-up', 'Used for reminding users about the entry ', NULL, 8)
, ('10', 'Delete later', 'Used for entries which will be deleted later', NULL, 6);

-- Populate the Diaries table:
INSERT INTO teamSweetDreams_DMS.Diaries (diary_id, title, date_created, last_updated, subject, org_id, owner_id) VALUES
('32766', 'Meetings', '2022-01-22', '2022-07-15', 'Manages departmental meeting times', 255, 3),
('1', 'Partners', '2022-07-04', '2022-08-03', 'Keeps track of business partners events', 9, 5),
('32767', 'Catalog', '2020-11-11', '2022-09-21', 'Deals with current products on the market', 3, 4),
('34', 'Inventory', '2021-12-31', '2022-10-10', 'Tracks what items are currently in stock and/or need restocking', 8, 7),
('2347', 'Employees', '2021-10-10', '2022-11-29', 'Manages current employees, contains personal info', 3, 8),
('1237', 'Feedback', '2021-10-10', '2022-12-14', 'Contains customer and employee survey feedback', 1, 1),
('3498', 'Trends', '2020-02-29', '2023-01-07', 'Tracks crucial trends in company dealings', 2, 3),
('2348', 'Events', '2022-06-15', '2023-02-18', 'Calendar for event planning', 5, 10),
('2312', 'Campaigns', '2021-08-08', '2023-03-22', 'Holds info about various company campaigns and descriptions', 7, 2),
('65535', 'Team', '2020-11-11', '2023-04-05', 'For teams to communicate with one another', 1, 3);

-- Populate the Entries table:
INSERT INTO teamsweetdreams_dms.entries (entry_id, entry_title, start_time, end_time, priority, description, entryType_id, owner_id, org_id,
diary_id, location_id) VALUES
 ('1', 'Meeting with boss', '2023-11-10 08:23:14', '2023-11-11 01:23:45', '1', 'Have statistics ready', 2, 2, 2, 34, 2)
, ('2', 'Presentation on Q1', '2023-11-10 12:34:56', '2023-11-11 05:45:23', '2', 'Black tie', 1, 2, 3,1237, 5)
, ('3', 'Meeting with board', '2023-11-11 09:12:34', '2023-11-12 02:34:56', '1', 'With Brian Gormanly', 2, 4, 1, 1, 1)
, ('4', 'New York show', '2023-11-12 06:45:23', '2023-11-13 03:12:34', '3', NULL, 6, 3, 6, 2312, NULL)
, ('5', 'Meeting with Greg', '2023-11-12 11:23:45', '2023-11-13 08:23:14', '1', 'About Forged Signature', 10, 8, 4, 32766, NULL)
, ('6', 'Call with Washington', '2023-11-12 12:34:56', '2023-11-13 09:12:34', '4', NULL, 5, 4, 3, 32766, NULL)
, ('7', 'Presentation on Q2', '2023-11-13 05:45:23', '2023-11-14 06:45:23', '2', 'Providing an overview of this quarter\'s progress.', 8, 10,
255, 65535, 4)
, ('8', 'Meeting with Connor', '2023-11-14 01:23:45', '2023-11-14 11:23:45', '1', 'Connor\'s office', 3, 1, 7, 2312, NULL)
, ('9', 'Hong Kong show', '2023-11-14 08:23:14', '2023-11-15 02:34:56', '3', NULL, 4, 8, 1, 2348, 1)
, ('10', 'Great Job, Guys!', '2023-11-11 02:34:56', '2023-11-12 11:23:45', '1', 'Im so proud of everyone\'s work on Thursday. We nailed it!
Lunch on me tomorrow!', 7, 9, 2, 2347, NULL)
;

-- Populate the table RolePermissions
INSERT INTO teamsweetdreams_dms.RolePermissions (role_id, perm_id)
VALUES (10, 10), (1, 1), (2, 8), (3, 2), (4, 7), (5, 8), (6, 5), (7, 5), (8, 4), (9, 7);

-- Populate the the OrganizationMembers table
```

```
INSERT INTO teamsweetdreams_dms.organizationmembers (user_id, org_id)
VALUES (1, 2), (1, 3), (3, 1), (1, 4), (2, 2), (3, 2), (4, 2), (4, 6), (3, 7), (5, 7);

-- Populate the UserDiaries Table
INSERT INTO teamsweetdreams_dms.UserDiaries (user_id, diary_id)
VALUES (10, 34), (1, 1237), (2, 2312), (3, 2312), (4, 2347), (5, 2348), (6, 65535), (7, 65535), (8, 34), (9, 32767);

-- Populate the UserPermissions table
INSERT INTO teamsweetdreams_dms.userPermissions (user_id, perm_id)
VALUES (10, 10), (1, 1), (2, 8), (3, 2), (4, 7), (5, 8), (6, 5), (7, 5), (8, 4), (9, 7);
```

Handling Foreign Key Constraints

With so many foreign key references, inserting data into a table is unsuccessful when done out of order. The execution of the following SQL on the table Organizations gives an example of this with the error “Cannot add or update a child row: foreign key constraint fails.” This is thrown because the table takes a foreign key “creator_id” from the Users table which in this instance has not been created yet, so the reference to creator_id does not exist. To bypass this error for the sake of loading example data, the query “SET FOREIGN_KEY_CHECKS=0” (line 60) is executed before the INSERT INTO statement in order to disable foreign key constraints and allow the insertion. After the INSERT INTO statement is executed, “SET FOREIGN_KEY_CHECKS=1” (line 73) is run to enable foreign key constraints again.



```
59
60 •  SET FOREIGN_KEY_CHECKS=0;
61 •  INSERT INTO teamsweetdreams_dms.Organizations (org_id, org_name, maxDiaries, creator_id, date_created)
62        VALUES (1, 'Public Relations', 65535, 1, '2023-11-08 06:32:05')
63        , (2, 'Marketing', 65535, 2, '2023-11-08 10:30:59')
64        , (3, 'Advisors', 65533, 1, '0000-01-01 00:00:00') -- minimum datetime
65        , (4, 'CEO', 65535, 1, '2023-11-08 10:30:59')
66        , (5, 'Administrators', 0, 1, '2022-11-25 10:30:59')
67        , (6, 'Human Resources', 15563, 1, '2014-03-08 10:30:59')
68        , (7, 'Information Technology', 65535, 1, '2023-11-08 10:30:59')
69        , (8, 'Finance', 65535, 1, '2023-11-08 10:30:59')
70        , (9, 'Sales', 65535, 1, '2023-11-08 10:30:59')
71        , (255, 'Operations Management Executive Special Team', 65535, 1, '9999-12-31 23:59:59') -- maximum datetime, varchar, maxDiaries
72        ;
73 •  SET FOREIGN_KEY_CHECKS=1;
74
75 •  DELETE FROM teamsweetdreams_dms.Organizations WHERE org_id >= 1;
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
505	13:13:25	CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.OrganizationMembers (us...	0 row(s) affected	0.031 sec
506	13:13:25	DROP TABLE IF EXISTS teamSweetDreams_DMS.UserDiaries	0 row(s) affected, 1 warning(s): 1051 Unknown table 'teamsweetdreams_dms.userdiaries'	0.000 sec
507	13:13:25	CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.UserDiaries (user_id TIN...	0 row(s) affected	0.031 sec
508	13:13:26	DROP TABLE IF EXISTS teamSweetDreams_DMS.userPermissions	0 row(s) affected, 1 warning(s): 1051 Unknown table 'teamsweetdreams_dms.userpermiss...	0.000 sec
509	13:13:26	CREATE TABLE IF NOT EXISTS teamSweetDreams_DMS.userPermissions (user_id ...	0 row(s) affected	0.016 sec
510	13:21:53	INSERT INTO teamsweetdreams_dms.Organizations (org_id, org_name, maxDiaries, cre...	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`teams...	0.015 sec

Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`teamsweetdreams_dms`.`organizations`, CONSTRAINT `organizations_ibfk_1` FOREIGN KEY (`creator_id`) REFERENCES `users` (`user_id`))

The constraints we have implemented help protect our database from insertion of invalid instances. For example, our primary keys are defined as NOT NULL and UNSIGNED so that the unique identifier for each table must have a value and also be a positive integer data type. Additionally, we used DATETIME data types to handle our date variables to avoid issues with inconsistent formatting, which is a common issue when dealing with dates. Another example of our constraints is shown in our tables Locations and Entries, where the foreign key reference to Locations table in Entries table is Null as well as are the non-key attributes of Locations; this

allows for null values to exist and thus its optional definition and inclusion as part of an entry. Finally, each entity/table has its own primary key made up of a single value that uniquely identifies that specific field across all tables. This way, it is simple to update only a part of the database rather than filling the database with extraneous junk data in order to add one simple instance. Referencing these keys as foreign keys in other tables also reduces data redundancy and assures all data is relevant and related in some way in order to be inserted.

We implemented the following queries to ensure our tables have proper structures:

```
-- The following queries are used to test the database's structure.

-- show all colors' information whose hexcode has a 9 in it.
SELECT * FROM teamsweetdreams_dms.Colors wHERE hexcode LIKE '%9%';

-- show the name of each color and what activity status it is assigned to.
SELECT C.color_name, activity_id
FROM Colors C
LEFT JOIN ActivityStatus A ON A.color_id=C.color_id
ORDER BY C.color_id;

-- Show the address of all places with a specified city
SELECT address_ln_1, address_ln_2  FROM teamsweetdreams_dms.Location WHERE city IS NOT NULL OR '';

-- show the person who was online most recently.
SELECT U.fullname, A.last_assigned
FROM Users U
LEFT JOIN ActivityStatus A ON A.activity_id=U.activity_id WHERE A.name='Online'
ORDER BY A.last_assigned DESC LIMIT 1;

-- show how many organizations each user has created
SELECT COUNT(org_id), fullname FROM teamsweetdreams_dms.Organizations O
RIGHT JOIN Users U ON U.user_id=O.creator_id
GROUP BY fullname;

-- show the name of any person whose password is 8 characters or less
SELECT fullname FROM teamsweetdreams_dms.users WHERE LENGTH(password) <= 8 ;

-- Select all entries that are of type 'Note'
SELECT entry_title, entryType_id FROM Entries WHERE entryType_id = (SELECT entryType_id FROM
teamsweetdreams_dms.entrytypes WHERE name = 'Note');

-- Retrieve the names of people in the company and the permissions they have assigned to them.
SELECT name, perm_name FROM Roles R
RIGHT JOIN RolePermissions RP ON RP.role_id = R.role_id
RIGHT JOIN Permissions P ON P.perm_id = RP.perm_id
ORDER BY R.role_id;

-- Retrieve the names of people in the company and the organizations they are a part of.
SELECT fullname, org_name FROM Users U
INNER JOIN organizationmembers OM ON OM.user_id = U.user_id
INNER JOIN Organizations O ON O.org_id = OM.org_id
ORDER BY U.user_id;

-- Retrieve the names of people in the company and the diaries they have access to.
SELECT fullname, title FROM Users U
JOIN UserDiaries UD ON UD.user_id = U.user_id
JOIN Diaries D ON D.diary_id = UD.diary_id
ORDER BY D.diary_id;

-- Retrieve the names of people in the company and the permissions they have assigned to them.
SELECT fullname, perm_name FROM Users U
JOIN UserPermissions UP ON UP.user_id = U.user_id
```

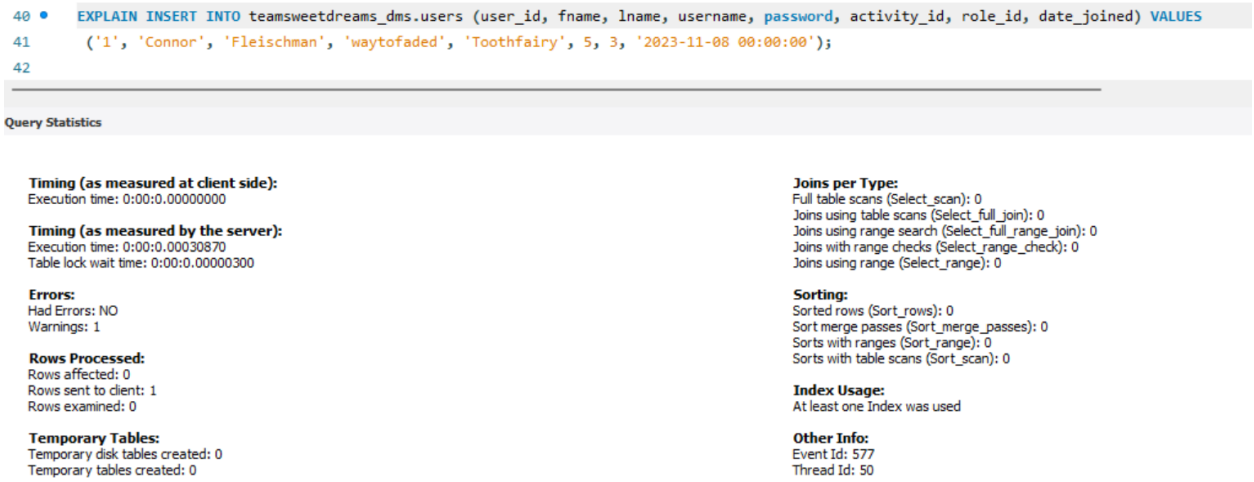
```
JOIN Permissions P ON P.perm_id = UP.perm_id
ORDER BY U.user_id;
```

Inserting Data & Data Optimization

Using the INSERT INTO statement, we were able to insert 10 unique instances of information into each table. Certain instances tested the maximum and minimum requirements in each field. With the data inserted, we began to optimize our queries to ensure that our database runs smoothly and efficiently. We condensed multiple INSERT INTO statements into a single statement with multiple values. This allowed us to cut back on space as well as processing time.

Furthermore, we ensured that our statements ran efficiently. We compared running multiple smaller INSERT INTO statements with one large INSERT INTO statement. Using the EXPLAIN and SHOW STATUS statements we compared the number of times the query ran and the number of round-trips needed to complete the query. We found that when multiple INSERT INTO statements were used the number of queries ran and the number of round-trips needed was much higher and much less efficient, than using a single large INSERT INTO statement. This demonstrated to us that using one large statement not only was easier to write, but also it cut down on the number of queries ran, and the number of server round-trips needed to complete the query.

In the end, our group found that inserting multiple instances at once using a single INSERT statement is generally more efficient than inserting instances one by one. However, the performance benefits are directly proportional to the size of the data being inserted as the more data, the more time it will take to complete the query, and vice versa.



The above images describes the use of the EXPLAIN query on a non-optimized INSERT INTO statement. As shown, the time to complete this one singular query (measured by the server) is 0:00:0.00030870. The image below shows the use of the same EXPLAIN query, however, now it is used on an INSERT INTO statement which has been correctly optimized. The query statistics

tab shows that from the optimized query the time to complete the query (measured by the server) is only 0:00:0.00050990. This means that the difference in time when running one unoptimized INSERT INTO statement is only 0:00:0.0002012 faster than if the INSERT INTO statement was optimized. This gap in time only exists if one INSERT INTO statement is needed. If the user needed to use multiple, the time to complete the query is much faster when using an optimized statement than with an unoptimized statement. This hypothesis is further confirmed when using the SHOW STATUS statement as it confirms that when multiple INSERT INTO statements are used the time to complete the queries is much greater than the time to complete one large, optimized INSERT INTO statement.

```
22 •  EXPLAIN INSERT INTO teamsweetdreams_dms.users (user_id, fname, lname, username, password, activity_id, role_id, date_joined) VALUES
23     ('1', 'Connor', 'Fleischman', 'waytofaded', 'Toothfairy', 5, 3, '2023-11-08 00:00:00')
24    , ('2', 'Evan', 'Spillane', 'EvanSpillane', 'Password', 2, 4, '2023-11-08 00:00:00')
25    , ('3', 'Lilli', 'Cartiera', 'LilliCartiera', 'Google123', 255, 7, '2023-11-08 00:00:00')
26    , ('4', 'Abel', 'Scholl', 'AbelScholl', 'BehindU', 10, 10, '2023-11-08 00:00:00')
27    , ('5', 'Neo', 'Pi', 'NeoPi', 'ABCDEFG123', 6, 1, '2023-11-08 00:00:00')
28    , ('6', 'Saul', 'Goodman', 'BetterCallSaul', 'LyrsAreUs', 1, 1, '2023-11-08 00:00:00')
29    , ('7', 'Anakin', 'Skywalker', 'theChosenOne', 'Jedi4Life', 1, 3, '2023-11-08 00:00:00')
30    , ('8', 'Neo', 'Anderson', 'MAtrixHakr', '123CantHakME', 20, 4, '2023-11-08 00:00:00')
31    , ('9', 'Jesus', 'Christ', 'SonofGod', 'Heaven', 2, 7, '2023-11-08 00:00:00')
32    , ('10', 'Finn', 'TheHuman', 'FinnTheHuman', 'ILoveJake', 1, 6, '2023-11-08 00:00:00');
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00050990
Table lock wait time: 0:00:0.00000400

Errors:
Had Errors: NO
Warnings: 1

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 0

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
At least one Index was used

Other Info:
Event Id: 581
Thread Id: 50

Normalization Check

In order to normalize our data up to the third normal form, there are a few requirements we need to fulfill. First normal form (1NF) requires that each column has a single value and that each row is unique. We achieved this by ensuring there were no multi-valued attributes as well as using candidate keys to uniquely identify each row in each table. Therefore, our data is in 1NF. Second normal form (2NF) requires that the data be in 1NF, there are no partial dependencies, every table has a primary key, and relationships between tables are formed using foreign keys. Our use of primary and foreign keys have already been established, and the only tables in our database with candidate keys made of a set of foreign keys have no other non-key columns and therefore no partial dependencies; our data is in 2NF. Finally, the third normal form (3NF) requires that the data must be in 2NF and have no transitive dependencies. All of our tables are

split so that every non-key attribute is dependent only on the primary key of that table, eliminating any transitive dependencies and ensuring our data fulfills 3NF.

Application Development

Graphical User Experience Design

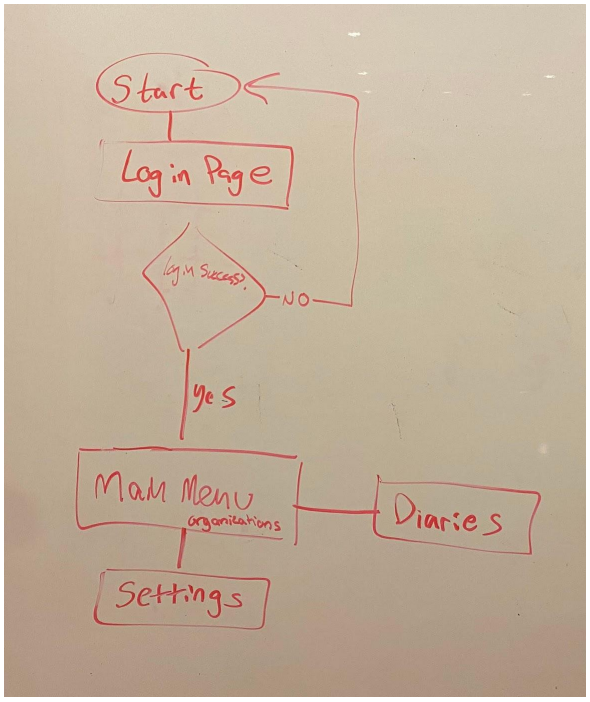


Figure 21: Login Flowchart

We used a flowchart to more easily visualize what our User Experience would look like. In the following images you will notice that upon STARTing the user is brought to a LOGIN page. In this page the user is prompted to give their respective *username* and *password* in order to gain access to the remainder of the application. Once the user has input their credentials, the system checks whether or not they match the information in the database. If their information does not match that in the database, the system will then return the user to START and prompt them for their *username* and *password* again. If the user's information does match that in the database, the user is then forwarded to the MAIN MENU. From this page the user is able to access the SETTINGS and DIARIES pages. The MAIN MENU functions as the central landing page for all the users needs.

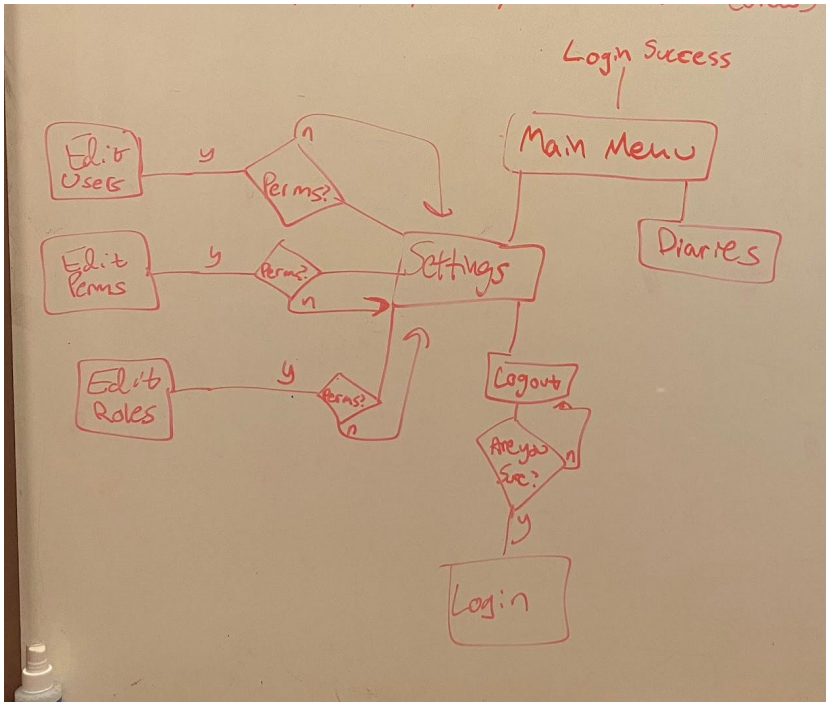


Figure 22: Main Menu, Settings, Edit, Diaries Flowchart

From there the user has access to the different organizations they are members of. Within these organizations are DIARIES which hold entries which users are then able to edit directly. The user is able to click DIARIES to be brought to the different DIARIES within the given organization. The user is then able to see the distinct entries within the different DIARIES. Upon returning to the MAIN MENU the user is able to access the SETTINGS page. This page is a hub for any and all settings the user might need access to. Upon selecting SETTINGS the user is able to see and change their personal information. Their *username*, *password*, and *email*. These settings are basic and given to all users regardless of their role or permissions. The application then checks whether or not the user has the appropriate permissions to access certain aspects of the settings. The permission being checked for is the administrative privilege. These aspects are EDIT USERS, EDIT PERMS, and/or EDIT ROLES. The EDIT USERS settings controls what roles a specific user has. It also manages what users are part of a given organization. The EDIT PERMS tab manages the different permissions corresponding to a given role. In this tab the user can tweak what specific permissions a user with a given role will have. In the EDIT ROLES section the user is able to edit the distinct roles which are given to the different users within an organization. Within the EDIT ROLES section the user can create, delete, and edit the roles for their organization. Within these aspects of the SETTINGS page the user can also navigate back to the LOG OUT page. If the user selects LOG OUT they will be prompted with a pop-up asking them if they “are sure” if they want to log out. If the user selects **YES** then they will be returned

to START and prompted to LOGIN again. If the user selects **NO** the user will then be returned to the SETTINGS page.

Views Implementation

```
-- The following document creates the required views for our pages

-- creates a view merging all diaries, all associated entries, the name of the associated organization, and a user_id
with access to it.
CREATE OR REPLACE VIEW DiaryInfoPgVW AS (
SELECT D.*, E.*, O.org_name, UD.user_id FROM Diaries D
LEFT JOIN Entries E ON E.entryDiary_id = D.diary_id
LEFT JOIN UserDiaries UD ON UD.diary_id = D.diary_id
LEFT JOIN Organizations O ON O.org_id = D.diaryOrg_id
ORDER BY D.diary_id
);

-- creates a view merging all entries, their locations, their associated diary titles, and the fullname of the entry
owner.
CREATE OR REPLACE VIEW EntryInfoPgVW AS (
SELECT E.*, L.*, fullname, D.title AS diary_title FROM Entries E
LEFT JOIN Diaries D ON D.diary_id = E.EntryDiary_id
LEFT JOIN Users U ON U.user_id = E.entryOwner_id
LEFT JOIN Location L ON L.place_id = E.location_id
ORDER BY E.entry_id
);

-- Create few for Settings Page
-- get: full name, user role, user permissions, activity status, Organization names, and organization members.
DROP VIEW SettingsPageVW;
CREATE VIEW SettingsPageVW AS (
SELECT fullname, A.name AS 'Activity Status', perm_name, R.name AS 'Role', O.org_name FROM Users U
JOIN UserPermissions UP ON UP.user_id = U.user_id
JOIN Permissions P ON P.perm_id = UP.perm_id
JOIN RolePermissions RP ON RP.perm_id = P.perm_id
JOIN Roles R ON R.role_id=RP.role_id
JOIN organizationMembers OM ON OM.user_id=U.user_id
JOIN Organizations O ON O.org_id=OM.org_id
JOIN ActivityStatus A ON A.activity_id=U.activity_id
ORDER BY U.user_id
);
SELECT * from SettingsPageVW;

-- Create few for EditRoles Page
-- get: full name, user role, user permissions
DROP VIEW EditRolesVW;
CREATE VIEW EditRolesVW AS (
SELECT fullname, perm_name, R.name AS 'Role' FROM Users U
JOIN UserPermissions UP ON UP.user_id = U.user_id
JOIN Permissions P ON P.perm_id = UP.perm_id
JOIN RolePermissions RP ON RP.perm_id = P.perm_id
JOIN Roles R ON R.role_id=RP.role_id
ORDER BY R.role_id
);
SELECT * from EditRolesVW;
```

Graphical User Interface Design

Connection to DB

Our connection to our database is internal and used by the other pages to execute sql queries. The code for this connection can be found inside the RootWindow module for our RootWindow class which is used to reference the main window itself.

```
##the following field is used to reference the database connection.
        self.connection = mysql.connector.connect(host="localhost", user="root", password='root', database="teamsweetdreams_dms")
        self.cursor = self.connection.cursor()
```

Login Page

Our login page is relatively simple with a welcome screen and 2 input prompts for the user to enter their username and password. The entered fields are then used to run an SQL query that verifies the existence and match of the username and password, and if they are correct, it will welcome the user with their name, and then take them to the main menu page.

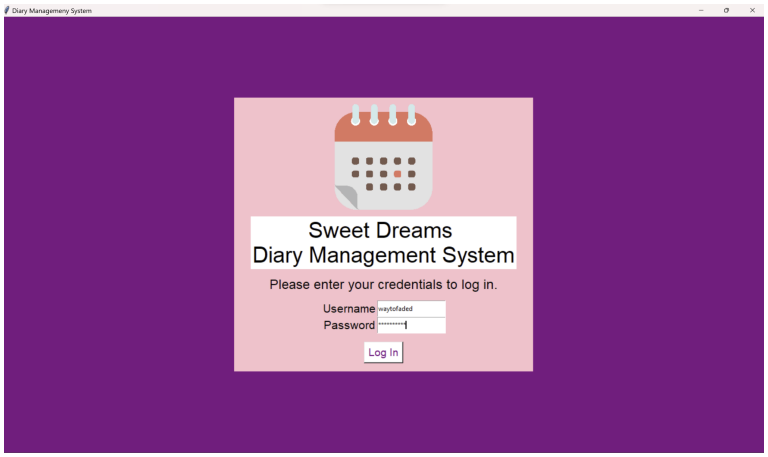


Figure 23: Login Page GUI

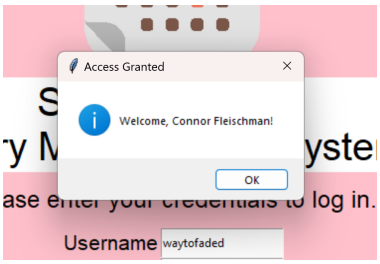


Figure 24: Login Access Granted

Source Code:

```
##Team Sweet Dreams Diary Management System
##Created on: 11/14/2023
##Last Updated: 11/19/2023

##The following code defines the Login page of the Diary Management System
##If a user is not logged in, this screen will appear.


##Team Sweet Dreams Diary Management System
##Created on: 11/14/2023
##Last Updated: 11/27/2023

##The following code defines the Login page of the Diary Management System
##If a user is not logged in, this screen will appear.


import tkinter as tk
from tkinter import Canvas, PhotoImage
from PIL import Image, ImageTk
import DMS_MainPage

def Action(root, username, password, master=None):
        ##first verifies the username and password of the user.
        ##if verified, displays a confirmation message and sends the user to the main page.
        ##otherwise, shows a warning and allows the user to retry.

        user_id=verifyCredentials(root, username, password)
        if user_id:
        root.cursor.execute(f"""SELECT fullname FROM Users WHERE user_id={user_id};""")
        fullname = root.cursor.fetchone()[0]
        tk.messagebox.showinfo('Access Granted', f'Welcome, {fullname}!')
        root.setCurrentUser(user_id)
        root.removeWidgets(master)
        DMS_MainPage.MainPage(root)
        else:
        tk.messagebox.showwarning('Access Denied', f'The username or password is incorrect.')

def verifyCredentials(root, username, password):
        ##this might not be how this will be done in the final product,
        ##but mysql workbench is acting up Big Time and I cant troubleshoot it

        root.cursor.execute("""SELECT username, password, user_id FROM Users;""")
        data = root.cursor.fetchall()
        for item in data:
        if item[0]==username and item[1]==password:
        return item[2] ##return user_id
        return None

def Login(root):
        ##takes the RootWindow object as a parameter
```

```
calendarImage = root.getImage('calendar_clipArt.png', int(1777/10), int(1920/10))

root.setCurrentPage("Login")
outerFrame = tk.Frame(root.root, bg='Purple')
innerFrame = tk.Frame(outerFrame, bg = "Pink", width=root.screen_width/2, height=root.screen_height/2)

imageLabel=tk.Label(innerFrame, image=calendarImage, bg='Pink')
imageLabel.image=calendarImage
label1 = tk.Label(innerFrame, text='Sweet Dreams \nDiary Management System', font=('Helvetica', 30), bg="White")
label2 = tk.Label(innerFrame, text='Please enter your credentials to log in.', font=('Helvetica', 18), bg='Pink')

entryFrame = tk.Frame(innerFrame)

usernameLabel = tk.Label(entryFrame, text='Username', font=('Helvetica', 16), bg='Pink')
passwordLabel = tk.Label(entryFrame, text='Password', font=('Helvetica', 16), bg='Pink')

userEntry = tk.Entry(entryFrame)
passEntry = tk.Entry(entryFrame, show="*")

loginButton = tk.Button(innerFrame, text="Log In", font=('Helvetica', 14), bg='White', fg="Purple",
        command=lambda:Action(root, userEntry.get(), passEntry.get(), master=root.root))

##add widgets to screen
outerFrame.pack(fill=tk.BOTH, expand=True)
innerFrame.pack(expand=True, padx=root.screen_width/10, pady=root.screen_height/10)
imageLabel.pack(padx=10, pady=10)
label1.pack(fill=tk.BOTH, padx=30)
label2.pack(fill=tk.BOTH, padx=10, pady=12)
entryFrame.pack()
usernameLabel.grid(column=1,row=1, sticky='nsew')
userEntry.grid(column=2,row=1, sticky='nsew')
passwordLabel.grid(column=1,row=2, sticky='nsew')
passEntry.grid(column=2,row=2, sticky='nsew')
loginButton.pack(padx=10, pady=15)
```

Main Menu Page

Our main menu page consists of a dropdown menu where a user can select one of the organizations they are a part of, which controls a set of buttons that each link to a diary. Clicking on one of these buttons will change the canter frame of the screen to focus on that diary's information by showing the title and a responsive calendar for that diary. This will also control the other side of the screen where a diary's entries are shown based on the criteria. By default, it shows all entries for the chosen diary.

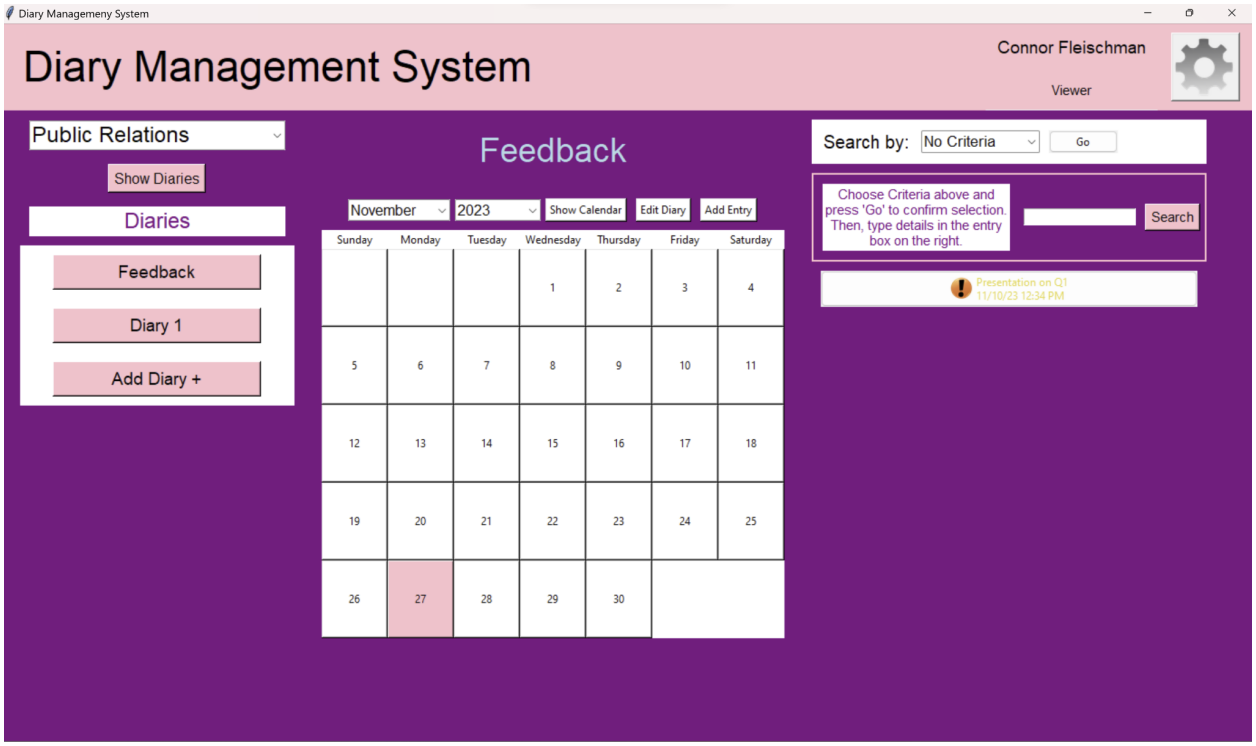


Figure 25: Main Menu Page GUI

Action Pages

i. Search

Our search function is built into our menu page and allows users to fill out an input field, which upon hitting search will query and return any accessible diaries that contain the user input criteria, either place, time, date, or duration.

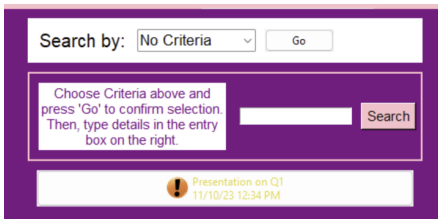


Figure 26: Search GUI

ii. Insertion

The main diary page has ways to insert. This is done so that users can create new diaries and new entries which did not previously exist. Clicking on the “Add Diary” button prompts this action.

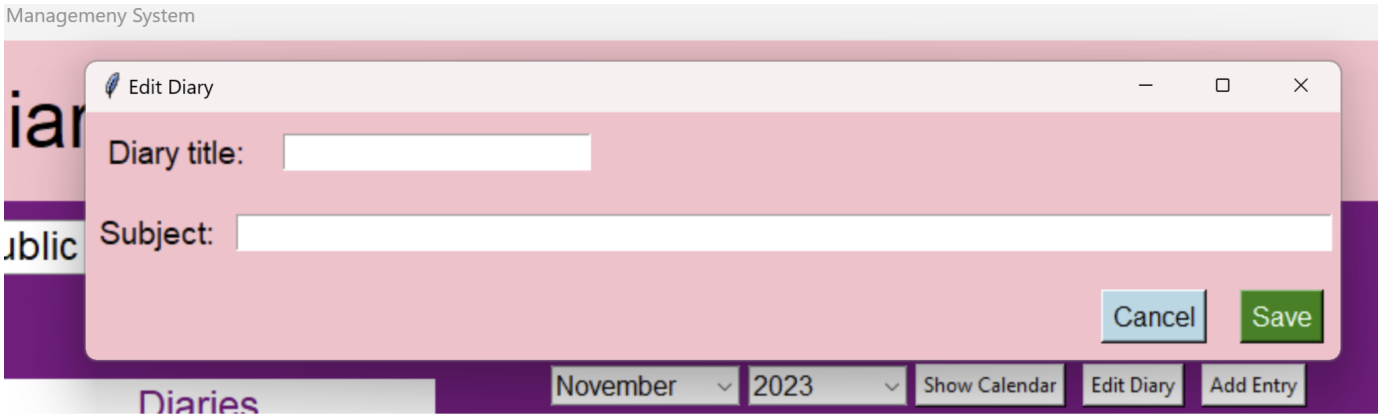


Figure 27: Adding a New Diary

iii. Modification

Admin users can edit user settings using the User Settings page in the top right of the main menu page. The users can view their user information such as Name, Username, Organizations etc. There is then an edit information button which changes the displayed data into input fields for the user to enter in what they want their data modified to. This is also true for diaries, organizations, and entries.

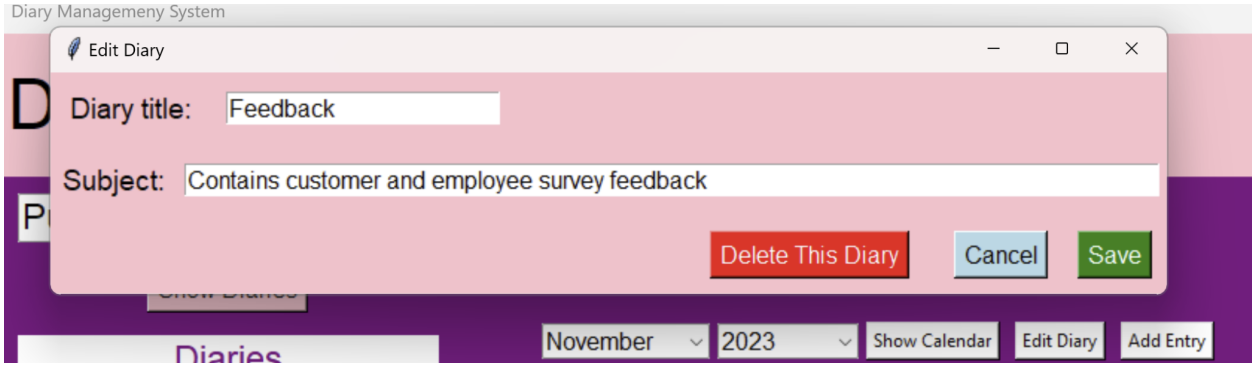


Figure 28: Editing a Diary

iv. Deletion

Users on the diaries and entries pages have the ability to delete these fields if they have permission to do so.

v. Print All

Our Print All implementation is meant to be through the settings page fit with information about all information about the data maintained by the application.

Our code implementation for our Main Page is shown in the following:

```
##Team Sweet Dreams Diary Management System
##created on: 11/15/23
##last updated: 11/27/2023

##The following code defines the main screen for the Diary Management System.
##If a user is logged in successfully, this screen will appear.

import tkinter as tk
from RootWindow import RootWindow
import calendar
import datetime
from tkinter import messagebox, PhotoImage
from tkinter.ttk import Combobox, Button, Style
from PIL import Image, ImageTk

def banner(root):
        ##creates the banner at the top of the screen and the spacers on the side
        ##returns the space left on the screen

        aW, aH = root.screen_width, root.screen_height ##available width and height of the screen
        topW, topH = root.screen_width, 75 ##width and height of top banner
        spacerW, spacerH = 20, aW-topH ##width and height of side spacers

        aW-=spacerW*2
        aH-=topH

        ##rewrite the width and height to variables to reduce hardcoded numbers
        topBanner = tk.Frame(root.root, bg="Pink", width=topW, height=topH)
        leftSpacer = tk.Frame(root.root, bg="Purple",width=spacerW, height=spacerH)
        rightSpacer = tk.Frame(root.root, bg="Purple",width=spacerW, height=spacerH)
        mainFrame = tk.Frame(root.root, bg="Purple", width=aW, height=aH)

        frame11 = tk.Frame(topBanner, height=100, width=300)
        frame112 = tk.Frame(frame11, bg='Pink', height=50, width=200)
        frame113 = tk.Frame(frame11, bg='Pink', height=50, width=200)
        frame114 = tk.Frame(frame11, bg='Pink', height=topH, width=topH)

        nameLabel = tk.Label(frame112, text=f'{root.getCurrentUserDetails()[3]}', font=('Helvetica', 14), bg='Pink')
        roleLabel = tk.Label(frame113, text=f'{root.getCurrentUserRoleDetails()[1]}', font=('Helvetica', 11), bg='Pink')

        img=root.getImage("gear.png", topH-5, topH-5)
        settingsButton = tk.Button(frame114, image=img, command=lambda:openSettings(root))
        settingsButton.image=img

        topLabel = tk.Label(topBanner, text="Diary Management System", font=("Helvetica", 35), bg="Pink", fg="Black")

        topBanner.pack(side='top', fill="both")
        leftSpacer.pack(side='left', fill="both")
        rightSpacer.pack(side='right', fill="both")
        frame11.pack(side='right', fill="both")
        topLabel.pack(side='left', padx=20, pady=5)
```

```
        frame114.pack(side='right', fill="both")
        frame112.pack(fill="both")
        frame113.pack(fill="both")
        nameLabel.pack(padx=10, pady=12, fill='both')
        roleLabel.pack(padx=10, fill='both', pady=10)
        settingsButton.pack(padx=15, pady=10)


        return mainFrame, aW, aH

def openSettings(root):
        ##opens a new window that allows a user
        ##to edit their account information
        ##if id=None, it prompts to login.

        ##create the popup
        userSettingsWindow = RootWindow(title="User Settings")
        userSettingsWindow.root.geometry("700x700")

        ##define the contents
        frame1 = tk.Frame(userSettingsWindow.root, bg="")

        nameLabel=tk.Label(frame1, text="Name:")
        nameInfo=tk.Label(frame1, text=f'{root.getCurrentUserDetails()[3]}') #Displays Name

        usernameLabel=tk.Label(frame1, text="Username:")
        usernameInfo=tk.Label(frame1, text=f'{root.getCurrentUserDetails()[4]}') # Displays Username

        passLabel=tk.Label(frame1, text="Password:")
        passInfo=tk.Label(frame1, text="*********")

        orgsLabel=tk.Label(frame1, text="Organizations:")


        nameEntry=tk.Entry(frame1)

        ##only passing the window to Save,
        ##so master of .removeWidgets(master) is None, which will close the whole window.
        backButton=tk.Button(frame1, text='<- Back', command=lambda:userSettingsWindow.removeWidgets())
        #editButton=tk.Button(frame1, text='Edit Information', command=lambda:editUserSettings(userSettingsWindow),
command=lambda:closeWindow(userSettingsWindow))
        editButton=tk.Button(frame1, text='Edit Information', command=lambda: [editUserSettings(userSettingsWindow),
closeWindow(userSettingsWindow)])


        ##Add the contents to the window
        frame1.pack()
        backButton.grid(column=1, row=1)
        editButton.grid(column=3, row=1)
        nameLabel.grid(column=1, row=3)

        usernameLabel.grid(column=1, row=4)
        passLabel.grid(column=1, row=5)
        orgsLabel.grid(column=1, row=7)
```

```
        nameInfo.grid(column=2, row=3)
        usernameInfo.grid(column=2, row=4)

        userSettingsWindow.run() ##open the window

def editUserSettings(userSettingsWindow):
        pass

def Save(root, master=None):
        ##this function will update the database with the new information.

        ##close the popup window (placeholder action for now)
        root.removeWidgets(master)


def EditOrg(org_name=None):
        ##opens a new window that allows a user
        ##to edit an organization's information
        ##if org_name=None, it prompts to create a new organization.

        ##create the popup
        editWindow = RootWindow(title="Edit Organization")
        editWindow.root.geometry("500x500")

        ##define the contents
        frame1 = tk.Frame(editWindow.root, bg="Blue")
        nameLabel=tk.Label(frame1, text="Organization name:")
        nameEntry=tk.Entry(frame1)

        saveButton=tk.Button(frame1, text='Save Changes', command=lambda:Save(editWindow))

        if org_name:
        if nameEntry.get():
        nameEntry.delete(0, tk.END)
        nameEntry.insert(0, org_name)

        ##Add the contents to the window
        frame1.pack()
        nameLabel.grid(column=1, row=1)
        nameEntry.grid(column=2, row=1)
        saveButton.grid(column=3, row=2)

        editWindow.run() ##open the window



def populateOptionsFrame(root, framesList):
        ##populates the options frame with a way to choose an organization and show its associated diaries

        opf = framesList[0]
        ##Add Frames
        headingFrame = tk.Frame(opf, bg="Purple")
        headingFrame.pack(fill='both')
```

```
        comboFrame=tk.Frame(opf, bg="Purple")
        comboFrame.pack(fill='both')
        headingFrame2 = tk.Frame(opf, bg="Purple")
        headingFrame2.pack(fill='both')
        diaryListFrame = tk.Frame(opf, bg="White")
        diaryListFrame.pack(fill='both')

        if diaryListFrame not in framesList:
        framesList.append(diaryListFrame)

        ##Add heading labels
        heading=tk.Label(headingFrame, text="Organization", font=('Helvetica', 20), bg="Purple", fg="Light Blue")
        #heading.pack(padx=10, pady=10)
        heading2=tk.Label(headingFrame2, text="Diaries", font=('Helvetica', 18), bg="White", fg="Purple")
        heading2.pack(padx=10, pady=10, fill=tk.X)

        userOrgData = root.getUserOrgData()

        orgs=[]
        for item in userOrgData:
        orgs.append(item[-1]) ##append organization name

        orgsCombo = Combobox(comboFrame, font=('Helvetica', 18), state="readonly")
        orgsCombo['values']=orgs
        orgsCombo.current(0)
        orgsCombo.pack(padx=10, pady=10)


        ##get all diaries a user has access to and the organization it is associated with(title and id)
        userDiaryData=root.getUserDiaryData()

        showDiariesButton = tk.Button(comboFrame, text='Show Diaries', font=('Helvetica', 12),
                bg="Pink", command=lambda:showOrgDiaries(root, framesList, orgsCombo.get()))
        editButton = tk.Button(comboFrame, text='Edit', font=('Helvetica', 12),
                bg="Pink", command=lambda:EditOrg(orgsCombo.get()))
        #editButton.configure(command=lambda editButton=editButton:EditOrg())

        showDiariesButton.pack(padx=5, pady=5)
        #editButton.pack(padx=5)

        showOrgDiaries(root, framesList, orgsCombo.get())


def EditOrg(id=None):
        ##opens a new window that allows a user
        ##to edit an organization's information
        ##if id=None, it prompts to create a new organization.

        ##create the popup
        editWindow = RootWindow(title="Edit Organization")
        editWindow.root.geometry("500x500")

        ##define the contents
        frame1 = tk.Frame(editWindow.root, bg="Blue")
```

```
        nameLabel=tk.Label(frame1, text="Organization name:")
        nameEntry=tk.Entry(frame1)

        ##only passing the window to Save,
        ##so master of .removeWidgets(master) is None, which will close the whole window.
        saveButton=tk.Button(frame1, text='Save Changes', command=lambda:Save(editWindow))

        ##Add the contents to the window
        frame1.pack()
        nameLabel.grid(column=1, row=1)
        nameEntry.grid(column=2, row=1)
        saveButton.grid(column=3, row=2)

        editWindow.run() ##open the window



def showOrgDiaries(root, framesList, currentOrg, returnTitle=None):
        dlf = framesList[5]
        root.removeWidgets(dlf)

        ##Show a button for each diary associated with an organization
        data=root.getUserDiaryData()
        for item in data:
        if item[1]==currentOrg: ##if the organization name is the same as the one in the combobox
        diaryButton = tk.Button(dlf, text=item[0], font=('Helvetica', 14),
                bg="Pink", width=20)
        diaryButton.config(command=lambda diaryTitle=item[0], diaryButton = diaryButton: createDiary(root, framesList,
currentOrg, diaryTitle=diaryTitle))
        diaryButton.pack(padx=5, pady=10)

        ##if permission allows:
        addDiaryButton = tk.Button(dlf, text="Add Diary +", font=('Helvetica', 14),
        bg="Pink", width=20, command=lambda:editDiary(root, currentOrg, framesList)) ##pass with no third parameter
prompts to add new
        addDiaryButton.pack(padx=5, pady=10)

        createDiary(root, framesList, currentOrg, diaryTitle=returnTitle)


def saveDiary(root, window, frame, title, subject, currentOrg, framesList, action, oldTitle=None):
        if not title:
        errorLabel = tk.Label(frame, text="Please enter a title.")
        errorLabel.grid(column=2, row=4, padx=10, pady=15)
        return
        now = datetime.datetime.now().strftime('%y-%m-%d %H:%M:%S')

        if action == 'new':
        try:
        root.cursor.execute(f"SELECT org_id FROM Organizations WHERE org_name = '{currentOrg}';")
        org_id = root.cursor.fetchall()[0][0]

        root.cursor.execute(f"""INSERT INTO Diaries (title, date_created, last_updated, owner_id, subject, diaryOrg_id)
        VALUES("{title}", "{now}", "{now}", {root.currentUser_id}, "{subject}", {org_id});""")
```

```
            root.connection.commit()

            root.cursor.execute(f"""INSERT INTO UserDiaries (user_id, diary_id) VALUES ({root.currentUser_id}, (SELECT
MAX(LAST_INSERT_ID()) FROM Diaries));""")
            root.connection.commit()
            except Exception as e:
            print("save new Diary: ", e)

            elif action == 'edit':
            try:
            root.cursor.execute(f"SELECT diary_id FROM Diaries WHERE title='{oldTitle}';")
            diary_id = root.cursor.fetchall()[0][0]
            root.cursor.execute(f"""UPDATE Diaries
SET title="{title}", last_updated="{now}", subject="{subject}" WHERE diary_id='{int(diary_id)}';""")
            root.connection.commit()
            except Exception as e:
            print("save edit diary: ", e)

            elif action == 'delete':
            try:
            root.cursor.execute(f"SELECT diary_id FROM Diaries WHERE title='{oldTitle}';")
            diary_id = root.cursor.fetchall()[0][0]
            root.cursor.execute(f"""DELETE FROM Diaries WHERE diary_id='{int(diary_id)}';""")
            root.connection.commit()
            except Exception as e:
            print("save delete diary: ", e)

            window.removeWidgets(master=None) ##close the window
            showOrgDiaries(root, framesList, currentOrg, title)


def editDiary(root, currentOrg, framesList, diary=None):
            ##opens a new window that allows a user
            ##to edit a diary's information
            ##if diary=None, it prompts to create a new diary.

            ##create the popup
            window = RootWindow(title="Edit Diary")
            window.root.geometry("750x150")

            ##define the contents
            frame1 = tk.Frame(window.root, bg="Pink")
            frame2 = tk.Frame(window.root, bg="Pink")
            frame3 = tk.Frame(window.root, bg="Pink")
            titleLabel=tk.Label(frame1, text="Diary title:", font=('Helvetica',14), bg="Pink")
            titleEntry=tk.Entry(frame1, font=('Helvetica',12))
            subjectLabel=tk.Label(frame2, text="Subject:", font=('Helvetica',14), bg="Pink")
            subjectEntry=tk.Entry(frame2, width=100, font=('Helvetica',12))
            cancelButton=tk.Button(frame3, text='Cancel', command=lambda:window.root.destroy(), font=('Helvetica',12),
bg="Light Blue")
            deleteButton=tk.Button(frame3, text='Delete This Diary', font=('Helvetica',12), bg="Red", fg="White")

            ##only passing the window to Save,
            ##so master of .removeWidgets(master) is None, which will close the whole window.
```

```
        saveButton=tk.Button(frame3, text='Save',  font=('Helvetica',12), bg="Green", fg="White")
        saveButton.config(command=lambda cO=currentOrg,
                saveButton=saveButton:saveDiary(root, window, frame1, titleEntry.get(), subjectEntry.get(), cO, framesList))

        ##Add the contents to the window
        frame1.pack(fill='both')
        frame2.pack(fill='both')
        frame3.pack(fill='both')
        titleLabel.pack(side='left', padx=10, pady=10)
        titleEntry.pack(side='left', padx=10, pady=10)
        subjectLabel.pack(side='left', padx=5, pady=10)
        subjectEntry.pack(side='left', padx=5, pady=10)
        saveButton.pack(side='right', padx=10, pady=10)
        cancelButton.pack(side='right', padx=10, pady=10)

        if diary:
        try:
        deleteButton.pack(side='right', padx=20, pady=10)
        root.cursor.execute(f"""SELECT diary_id, subject FROM diaryinfopgvw WHERE title='{diary}' AND
org_name='{currentOrg}';""")
        data = root.cursor.fetchall()[0]
        diary_id, subject = data[0], data[1]

        if titleEntry.get():
        titleEntry.delete(0, tk.END)
        titleEntry.insert(0, diary)
        if subjectEntry.get():
        subjectEntry.delete(0, tk.END)
        subjectEntry.insert(0, subject)

        saveButton.config(command=lambda cO=currentOrg,
        saveButton=saveButton:saveDiary(root, window, frame1, titleEntry.get(),
                        subjectEntry.get(), cO, framesList, action='edit', oldTitle=diary))

        deleteButton.config(command=lambda cO=currentOrg,
                deleteButton=deleteButton:saveDiary(root, window, frame1, titleEntry.get(),
                        subjectEntry.get(), cO, framesList, action='delete', oldTitle=diary))

        except Exception as e:
        print("first Edit: ", e)

        else:
        saveButton.config(command=lambda cO=currentOrg,
        saveButton=saveButton:saveDiary(root, window, frame1, titleEntry.get(), subjectEntry.get(),
                        cO, framesList, action='new'))


        window.run() ##open the window


def createDiary(root, framesList, currentOrg, diaryTitle=None):
        ##this function populates the middle calendar frame.
        ##with the diary name and a functional calendar.
```

```
cdf = framesList[1]

##Retrieve a specified user's diaries
root.cursor.execute(f"SELECT title, org_name FROM diaryinfopgvw WHERE user_id = '{root.currentUser_id}'")
diaries = root.cursor.fetchall()

##clear the screen
root.removeWidgets(cdf)

##create title headers
if diaryTitle == None: ##if no title is given, choose the first in the list to display.
for item in diaries:
if item[1]==currentOrg:
diaryTitle = item[0]
break

diaryLabel=tk.Label(cdf, text=diaryTitle, font=("Helvetica", 28), bg="Purple", fg="Light Blue")
diaryLabel.pack(side='top', pady=20)

monthYearFrame = tk.Frame(cdf, bg="Purple")
monthYearFrame.pack(side='top')

##create the calendar
calendarFrame=tk.Frame(cdf, bg="White") ##frame for the actual calendar
calendarFrame.pack(side='top', padx=20)

##initialize important veriables
today=datetime.date.today()
months=[]
years=[]

for i in calendar.month_name[1:]: ## get a list of all months
months.append(i)
for i in range(2023, today.year+10): ##get a list of ten years from this year, starting from 2023
years.append(i)


##create comboboxes
monthsCombo = Combobox(monthYearFrame, width=10, state="readonly", font=("Helvetica", 12))
monthsCombo['values']=months
monthsCombo.current((datetime.date.today().month)-1) ##index starts at 0
monthsCombo.pack(side='left', padx=5)

yearsCombo = Combobox(monthYearFrame,width=8, state="readonly", font=("Helvetica", 12))
yearsCombo['values']=years
yearsCombo.current((datetime.date.today().year)-2023)##first index is this year (2023)
yearsCombo.pack(side='left', pady=10)

showCalendarButton = tk.Button(monthYearFrame, text="Show Calendar", bg="White",
        command=lambda:showCalendar(root, calendarFrame, framesList,
                months.index(monthsCombo.get())+1,int(yearsCombo.get()), today, diaryTitle))
showCalendarButton.pack(side='left', padx=5)

editDiaryButton = tk.Button(monthYearFrame, text="Edit Diary", bg="White",
```

```
                command=lambda:editDiary(root, currentOrg, framesList, diary=diaryTitle))
        editDiaryButton.pack(side='left', padx=5)

        addEntryButton = tk.Button(monthYearFrame, text="Add Entry", bg="White",
                command=lambda:addEntry(root, currentOrg, framesList, diaryTitle))
        addEntryButton.pack(side='left', padx=5)

        showCalendar(root, calendarFrame, framesList, months.index(monthsCombo.get())+1, int(yearsCombo.get()), today,
diaryTitle)

        iterateEntries(root, framesList[4], diaryTitle)

        populateSearchFrame(root, framesList, diaryTitle, currentOrg)


def showCalendar(root, calendarFrame, framesList, month, year, today, diaryTitle):
        cdf=framesList[1]
        ##clear calendar
        root.removeWidgets(calendarFrame)

        command=iterateEntries(root, framesList[4], diaryTitle)
        weekdays=[]
        ##have to pull out sunday and do it separately because it absolutely refuses all attempts to be first in the iteration
        weekdays.append(calendar.day_name[-1:][0])
        dayLabel = tk.Label(calendarFrame, text=weekdays[0], bg="White")
        dayLabel.grid(column=1, row=1)
        ##iterate days
        for day in calendar.day_name[:-1]: ##get a list of all weekdays
        weekdays.append(day)
        dayLabel = tk.Label(calendarFrame, text=f'{day}', bg="White")
        dayLabel.grid(column=weekdays.index(day)+1, row=1)

        ##get the first weekday of the month and the number of days in the month, respectively
        firstWeekday, daysInMonth = calendar.monthrange(year, month)
        dayNum=1
        blankCount=0

        for week in range(1,7): ##up to 6 weeks (rows) in one month
        for day in range(1,8): ##seven days(columns) in one week
        if dayNum <= daysInMonth:
        if blankCount < firstWeekday+1: ##add a blank label for every day before the month's first day
                blankButton = tk.Button(calendarFrame, width=9, height=5, bg="White",)
                blankButton.grid(column=day, row=week+1, sticky='nsew')
                blankCount+=1
        else:
                if dayNum==today.day and today.month==month and today.year==year: ##if it is today's date
                dayButton = tk.Button(calendarFrame,text=dayNum, width=9, height=5, bg="Pink") ##indicate in pink
                else:
                dayButton = tk.Button(calendarFrame, text=dayNum, width=9, height=5, bg="White",) ##otherwise, no
indication
                dayButton.grid(column=day, row=week+1, sticky='nsew')
                dayNum+=1
```

```
def addEntry(root, currentOrg, framesList, diaryTitle):
        pass



def iterateEntries(root, entryFrame, diaryTitle, query=None):

        def x(entryFrame, title, color):
        lis=entryFrame.winfo_children()
        for i in lis:
        if isinstance(i, Button):
        if i['text']==title:
                style = Style().configure(f"{role[1]}.TButton",
                        foreground=f"#{color}")##color is the role color
                i['style'] = f"{role[1]}.TButton"

        #reset the frame
        root.removeWidgets(entryFrame)

        if not query:
        query = f"""SELECT entry_title, start_time, priority, entryOwner_id
FROM EntryInfoPgVW WHERE diary_title = '{diaryTitle}';"""

        root.cursor.execute(query)
        entries = root.cursor.fetchall()

        role = root.getCurrentUserRoleDetails()
        root.cursor.execute(f"""SELECT hexcode FROM Colors WHERE color_id={role[-1]}""")
        color = root.cursor.fetchall()[0][0]

        for item in entries:
        startTime=''
        if item[1]:
        startTime=datetime.datetime.strftime(item[1], "%m/%d/%y %I:%M %p")
        txt=f"{item[0]}\n{startTime}"
        img=getPriorityImage(priority=item[2])
        entryButton = Button(entryFrame, text=txt, width=60, image=img, compound=tk.LEFT,
                command=lambda:showEntryDetails(entryFrame, entryId=None))
        entryButton.pack(fill='both', padx=10, pady=10)
        entryButton.image=img
        if item[3] == root.currentUser_id:
        x(entryFrame, txt, color)



def getPriorityImage(priority=None):
        ##returns an image to display next to an entry based on the entry's priority
        if priority:
        img = Image.open("exclamation.png") # load image
        resized_image = img.resize((25,25), Image.Resampling.LANCZOS) # resize, remove structural padding
        new_image = ImageTk.PhotoImage(resized_image)# convert to photoimage
        return new_image
```

```
def showEntryDetails(entryFrame, entryId=None):
        ##creates a popup that shows the details for a specified entry

        ##create the popup
        window = RootWindow(title=f"Entry {entryId}")
        window.root.geometry("500x500")

        ##define the contents
        frame1 = tk.Frame(window.root)
        nameLabel=tk.Label(frame1, text="This is what will display all of the details of the entry.")


        ##only passing the window to Save,
        ##so master of .removeWidgets(master) is None, which will close the whole window.
        editButton=tk.Button(frame1, text='Edit Entry', command=lambda:editEntry(window, frame1))
        closeButton=tk.Button(frame1, text='Close', command=lambda:window.removeWidgets(master=None))##this
command will be changed later

        ##Add the contents to the window
        frame1.pack()
        nameLabel.grid(column=1, row=1)
        editButton.grid(column=3, row=2, padx=5)
        closeButton.grid(column=4, row=2, padx=5)

        window.run() ##open the window



def editEntry(window, frame):
        window.removeWidgets(master=frame) ##remove all contents from the frame
        nameLabel=tk.Label(frame, text="Title:")
        nameEntry=tk.Entry(frame)

        ##only passing the window to Save,
        ##so master of .removeWidgets(master) is None, which will close the whole window.
        saveButton=tk.Button(frame, text='Save Changes', command=lambda:Save(window))

        ##Add the contents to the window
        frame.pack()
        nameLabel.grid(column=1, row=1)
        nameEntry.grid(column=2, row=1)
        saveButton.grid(column=3, row=2)

        window.run() ##open the window


def searchEntries(root, entryFrame, diaryTitle, criteria, details):
        query= ''
        match criteria:
        case "Place":
```

```
            query=f"""SELECT location_id, address_ln_1, address_ln2, city, state, zip FROM EntryInfoPgVW WHERE
(address_ln_1 LIKE '%{details}%' OR address_ln2 LIKE '%{details}%' OR city LIKE '%{details}%' OR state LIKE
'%{details}%' OR zip LIKE '%{details}%')AND title = '{diaryTitle}';"""
            case "Date":
            query=f"""SELECT entry_title, start_time FROM EntryInfoPgVW WHERE start_time LIKE '{details}%';"""

            case "Time":
            query=f"""SELECT entry_title, start_time FROM EntryInfoPgVW WHERE start_time LIKE '%{details}';"""

            case "Duration":
            query=f"""SELECT entry_title, start_time FROM EntryInfoPgVW WHERE duration LIKE '{details}';"""
            iterateEntries(root, entryFrame, diaryTitle, query=query)


def searchHelper(root, searchFrame, entryFrame, diaryTitle, criteria):
            root.removeWidgets(searchFrame)

            t=""
            match criteria:
            case "Date":
            t="Enter a Date (format: yyyy-mm-dd):"
            case "Time":
            t="Enter a Time (HH:MM:SS):"
            case "Duration":
            t="Enter Duration (hours): "
            case "Place":
            t="Enter a Place: "
            case "No Criteria":
            t ="'Choose Criteria above and
press 'Go' to confirm selection.
Then, type details in the entry
box on the right.'"

            searchLabel2=tk.Label(searchFrame, text=t, font=("Helvetica", 11), bg='White', fg="Purple")
            searchLabel2.pack(side='left', pady=10, padx=10)


            searchEntry = tk.Entry(searchFrame)
            searchButton=tk.Button(searchFrame, text="Search", font=("Helvetica", 11), bg='Pink',
                        command=lambda:searchEntries(root, entryFrame, diaryTitle, criteria, searchEntry.get()))

            searchButton.pack(side='right', padx=5)
            searchEntry.pack(side='right', padx=5)


def populateSearchFrame(root, framesList, diaryTitle, currentOrg):
            srf=framesList[3]
            root.removeWidgets(srf)
            searchFrame1 = tk.Frame(srf, bg='White')
            searchLabel1=tk.Label(searchFrame1, text="Search by:", font=("Helvetica", 15), bg='White')
            searchLabel1.pack(side='left', padx=10, pady=10)

            searchByCombo = Combobox(searchFrame1, width=12, state="readonly", font=('Helvetica', 13))
            searchByCombo['values']= ("No Criteria", "Date", "Time", "Duration", "Place")
```

```
        searchByCombo.set("No Criteria Set")
        searchByCombo.current(0)

        searchFrame2 = tk.Frame(srf, bg="Purple", highlightbackground="Pink", highlightthickness=2)
        searchHelper(root, searchFrame2, framesList[4], diaryTitle, searchByCombo.get())
        goButtonSearch = Button(searchFrame1, text="Go",
                command=lambda:searchHelper(root, searchFrame2, framesList[4], diaryTitle, searchByCombo.get()))

        searchByCombo.pack(side='left', pady=10)
        goButtonSearch.pack(side='left', padx=10)


        searchFrame1.pack(side='top', fill='both', pady=10)
        searchFrame2.pack(side='top', fill='both')



def MainPage(root):
        root.root.config(bg='Purple') ##set the main window's background to purple
        ##takes the RootWindow object as a parameter
        ##controls the layout for the whole main window

        availableSpace, aW, aH = banner(root) ## display the banner and borders of the main page

        optionsFrameWidth, optionsFrameHeight = (int(aW*0.2), aH)
        calendarFrameWidth, calendarFrameHeight = (int(aW*0.6), aH)
        entryFrameWidth, entryFrameHeight=(int(aW*0.2), aH)

        ##main page frames
        optionsFrame = tk.Frame(availableSpace, bg="Purple", width=optionsFrameWidth, height=optionsFrameHeight)
        calendarFrame = tk.Frame(availableSpace, bg="Purple", width=calendarFrameWidth, height=calendarFrameHeight)
        searchEntryFrame = tk.Frame(availableSpace, bg="Purple", width=entryFrameWidth, height=entryFrameHeight)
        searchFrame=tk.Frame(searchEntryFrame, bg="Purple", width=entryFrameWidth, height=100)
        entryFrame=tk.Frame(searchEntryFrame, bg="Purple", width=entryFrameWidth, height=entryFrameHeight)

        availableSpace.pack(fill='both')

        optionsFrame.grid(column=1, row=1, sticky='nsew')
        calendarFrame.grid(column=2, row=1, sticky='nsew', padx=10)
        searchEntryFrame.grid(column=3, row=1, sticky='nsew')
        searchFrame.grid(column=1, row=1, sticky='nsew')
        entryFrame.grid(column=1, row=2, sticky='nsew')

        framesList = [optionsFrame, calendarFrame, searchEntryFrame, searchFrame, entryFrame]

        populateOptionsFrame(root, framesList)
```

Conclusion and Future Work

Throughout the past semester, our group has vigorously worked on implementing a Diary Management System using a SQL database. Each phase of the project brought with it its own unique challenges which were divided up between the different members of the group. Using MySQL, and the queries and commands within the application, we built our database and its subsequent tables, records, and fields. Following this, we created multiple entity-relationship models describing the different relationships between the entities we would use in our database. Another aspect of importance regarding this project was understanding how to implement insertion and retrieval queries. The in-class lectures gave an apt description of how to create, update, and delete data in tables and views which allowed us to access our data from our python application. Possibly the most important feature of our database was the data types used; once the data insertion started, we immediately learned the benefits of strong, consistent data types. Our final challenge as a group came in the form of understanding the relationship between the user's experience and the database design. Through the implementation of a UX diagram, our group planned out what a user would encounter when proceeding through our application. With that, we were able to get a clear understanding of what our code should look like. In the end, our project was successful but by no means is this representative of one member, but rather of the group as a whole. It was utterly a team effort which allowed us to overcome these challenges and create a functional Diary Management System.

Features to Add for a Better Application:

In order to improve our application, there are many features we would like to implement. First, we want to truly flesh out the the roles and permissions functionality we originally planned. Tailoring access and editing rights based on user roles ensures data integrity as well as management and security efficiency. Visually distinguishing entries based on their type is also a goal of ours in order to improve the user experience. In addition to this, Utilizing the buttons on the calendar to reference entries specific to particular days can enhance navigation and usability by adding a more instinctual interactivity experience for users. Finally, making the search feature more user-friendly is crucial. This can be achieved by tweaking the design of the search entries, making input as simple as possible for the user. This enhancement would restrict the possible entry options, controlling the format of the input and reducing lookup issues stemming from bad input, thus making the search function much more efficient.

References

1. https://www.eventreference.com/diary-system.php
2. https://www.setmore.com/diary-management
3. https://clientdiary.com/
4. https://www.yworks.com/yed-live/
5. https://dev.mysql.com/doc/refman/8.0/en/numeric-types.html
6. https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html