Contact Management System

Database Management Systems
MSCS 542L

Marist Database Administrators (DBA's)

Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Fall 2022

Project Report of Contact Management System

Team Name

Marist DBA's

Team Members

1. Noah Carbo - noah.carbo1@gmail.com - (845) 489-8296 - (Team Head)
2. Theresa Gundel - theresa.gundel1@marist.edu - (518) 526-8737 - (Team Member)
3. Ryan Weidling - ryan.weidling1@marist.edu - (215) 630-7789 - (Team Member)
4. Dale Doster - dale.doster1@marist.edu - (845) 518-2405 - (Team Member)
5. Sai sumanth Gurrala - saisumanthreddy.gurrala1@marist.edu - (Team Member)
6. Thomas Diaz-Piedra - thomas.diaz-piedra1@marist.edu - (201) 824-3695 - (Team Member)
7. Banu Pitta Reddy - banuprakash.pittareddy1@marist.edu - (845) 214-3986 - (Team Member)

Description of Team Members

1. Noah Carbo
    a. My name is Noah Carbo, I am working on my master's in Software Development at Marist College. I wanted to work with my group based on their proximity to where I sat on the first day of class. Everyone on the team seems very competent in their fields and we have agreed on communicating through Whatsapp while working on this project. I am looking forward to working with everyone this semester.
2. Theresa Gundel
    a. I graduated from Binghamton University in 2021 with a Bachelor's in Computer Science. Now I am pursuing a Master's in Computer Science at Marist College. I did not know anyone in the class beforehand so I chose to work with this group because we were sitting near each other in class. We chose Noah as Team Head because he volunteered first and seemed like a good fit.
3. Ryan Weidling
    a. My name is Ryan and I am working on my second degree from Marist. I graduated in 2018 with a Bachelor's in Communications and am now pursuing a Master's in Computer Science to pursue a career in software engineering. I wanted to work with the current group members because of two reasons. One, they were in close proximity to me, and two, each person at some point in the first two weeks had demonstrated qualities

that would make them great teammates. Whether that was showing friendliness by introducing themselves when we sat down, or sharing their background and proving their technical prowess. We chose Noah as Team lead because when the project was announced he was already brainstorming ideas for what our group should be working on.

4. Dale Doster
    a. In 2020, I graduated with a degree in Computer Science from Marist College. My focus in undergrad was on Software Development. After having worked as a Software Engineer for the past couple years, I decided to pursue a Master's degree, again with a focus on Software Development. I decided to work with my team because they were sitting near me and were easily approachable. We decided to select Noah as the Team Head because he volunteered and has good communication skills.
5. Sai sumanth Gurrala
    a. I am Sai Sumanth, I graduated with a bachelors in computer science from Bharath University, India. I am currently working with my group members for the Database management project, because we were sitting together in class. And it made more sense once we got connected through a whatsapp group and shared our interests and work. I am confident that my team members are very competitive and together we can bring the most out of ourselves for the project of contact management system. We chose Noah as our team head, because we felt he was more capable of leading our team towards our goal.
6. Thomas Diaz-Piedra
    a. My name is Thomas. I graduated from Marist with a bachelors in computer science and minors in information systems and information technology last year. I wanted to work with my group mates because we all sat near each other on the first couple days of classes and we all seemed to have the same outlook when it came to this project. We chose Noah as the team head/leader as he volunteered to do it and seemed more than capable of doing the job. Really looking forward to working with this team this semester.
7. Banu Pitta Reddy
    a. My name is Banu Prakash, I graduated with a bachelors in Mechanical engineering from Jawaharlal Nehru Technological University, India. I have worked as a Software Quality Engineer for almost 9 years and decided to pursue my masters degree with focus on Cloud Computing. I met only a few mates in the class and found these people enthusiastic about the work. We spent some time sharing our interests and work, where I got to know more about my team members. Everyone in the team looks to be

goal oriented and working towards a common objective. We chose Noah to head the team as he volunteered to do it and we trust his efficiency with the level of confidence that he demonstrates. I am very excited to join the team and looking forward to a great collaboration in completing this project.

Table of Contents

Table of Figures

Project Objective

The Contact Management System (CMS) was conceived to allow for storage of any and all information that a user wants to store. These could include phone numbers, home addresses, email addresses and other information which are stored securely and distinctly from other user information. This system at a minimum supports the following functions:

1. Allows for admin access with a secure credentials and include the following abilities
    a. To add/edit additional admins
    b. Add and delete user profiles
    c. Edit and manage user information
    d. Allows for resetting usernames/passwords in the event the user locks themselves out of their accounts
2. Allows for users to customize their stored information with the following capabilities
    a. Add/Delete multiple contact information
    b. Allows for editing of saved contact information
    c. Contacts can be searched based on their specific details
3. The CMS has a Graphical User Interface (GUI) which allows for interaction between the user and the database
    a. The GUI prevents duplicate contacts to be added
    b. It is user friendly including a welcome screen, provides reports in a tabular form and allows the user to exit the program.
4. Sensitive user information is ciphered so that in the unfortunate event of a hack of our user information the data that is collected would be indecipherable.

Review the Related Work

One example of an existing CMS is Streak. This CMS is unique because it can integrate with Google products [1]. This can be good for people who already use Google accounts, but for people who don't this is not that useful. A positive aspect of Streak is that it can integrate with Gmail to show the contact information, such as the company, of the person you're emailing. It can also show a timeline of all communications with a contact including emails, call logs, and files shared. Streak also has a good system for searching, sorting, filtering, and grouping contact data [1]. The main negative aspect of Streak is that it's focused on Google products so if you don't use a Google account or Google Chrome then this CMS is not ideal [4].

Another example of a CMS is Nimble. Nimble has many positive aspects. For example, you can import contacts into Nimble from your Gmail and Outlook accounts as well as from an uploaded CSV file. Another cool feature is that Nimble can scan email signatures and use that information to keep your contacts up to date. A negative aspect of Nimble is that you're limited to 25,000 contacts unless you want to pay to have more. Also, Nimble attempted to include email access in their tool but it lacks a lot of common functions like accessing folders [5].

A third example of a real-world CMS is Keap. A good feature of Keap is their user-friendly platform. They also have thousands of integration options such as Gmail, Outlook, and Quickbooks [2]. Based on these integrations, Keap automatically keeps all your contact data up to date [1]. A flaw that Keap has is it does not allow users to make calls from their platform. Users can make calls separately and log them after, but Keap does not automatically track calls [3].

The Merits of our Project

Our CMS will have a number of key capabilities provided for the end user that will entice them to utilize our CMS over a competitor. Our CMS will allow for both admin and standard user login, with admin access allowing for full control over additional admin processes, adding/deleting user profiles, editing/managing user information, and allowing for the resetting of usernames/passwords in the event the user locks themselves out of their accounts. Additionally, users will be able to customize their stored information with the following capabilities: add/delete multiple pieces of contact data, edit saved contact information, and search within the CMS. Our CMS will have a GUI which will allow for users to interact with the database. One of the main draws to our CMS will be the idea that the GUI will help prevent duplicate contacts from being added. On top of a functional GUI, our CMS will cipher sensitive user information so that in the unfortunate event of a hack, our users' information will be indecipherable and safe from bad actors.

GitHub Repository Address

https://github.com/NoahCarboMarist/MSCS542_ContactManagementSystem_ContactManagers

Entity Relationship Model (ER Model)

Our CMS is intended for a business to use. Based on this design decision, the entities are focused on workplace things, such as employees, groups (like teams), supervisors, and departments. We chose attributes that would typically describe each entity. For example, the Employee entity has the following attributes: first name, last name, email, phone number, etc. Some of these attributes are foreign keys like Employees have Department_ID and Supervisor_ID. The design mainly centers around employees' relationships to things such as logging into users, belonging to departments, and being supervisors. Our ER Model can be seen in Figure 1.

Descriptions:

The **Employee** entity describes a person employed at a company using our CMS. This person has a first name, last name, email address, phone number, work phone number, gender, and age. They also have an integer ID which serves as the primary key for this entity. There are foreign keys relating the Employee entity to its corresponding Department and Supervisor.

An Employee has a many to one relationship, "belong", with the Department entity. A **Department** has an integer primary key called ID. It has a foreign key corresponding to the ID of its Supervisor. It also has a name.

A Supervisor entity has a one to one relationship, "supervise", with the Department entity. It also has a one to one relationship, "are", with the Employee entity. The **Supervisor** entity has an integer ID as its primary key. It also has foreign keys corresponding to the Supervisor's Employee record and the Department they supervise.

An Employee entity has a one to many relationship, "are", with the **Certified** entity. This entity has an integer ID as the primary key as well as foreign keys corresponding to the Employee who received the certification and which Certification was received. There is also the date the employee was certified.

The Certified entity has a one to one relationship, "are", with the **Certification** entity. This entity has an integer ID as the primary key. It also has a name and type.

The Employee has a one to many relationship, "have", with the Relationship entity. The **Relationship** entity has an integer ID as its primary key. It also has two IDs for each employee in the relationship, a status, and a date.

The Employee entity has a one to one relationship, "login", with the User entity. The **User** entity has an integer as its primary key. It also has a password, email, type, and key. There is a foreign key to the ID in the corresponding Employee record.

The Employee entity has a one to many relationship, "send", with the **Message** entity. The Message entity has a primary key which is an integer called ID. It also has the sender ID, receiver ID, and group ID, as well as the date it was sent.

The Group entity has a one to many relationship, "has", with the Message entity. It also has a many to one relationship, "join", with the Employee entity. The **Group** entity has an integer ID for its primary key. It also has the ID of the leader of the group, the group name, and the date it was created. It also has a foreign key to the GroupMembers entity.

The **GroupMembers** entity has a many to one relationship, "contains", with the Group entity. The GroupMembers entity has a primary key which is an integer ID. It also has the group ID, user ID, and the date the user joined.

The Employee entity has a one to many relationship, "have", with the **Email History** entity. This entity has the IDs of the sender and receiver, as well as the date it was sent and the body of the email. The Email History has a unique integer ID as its primary key.

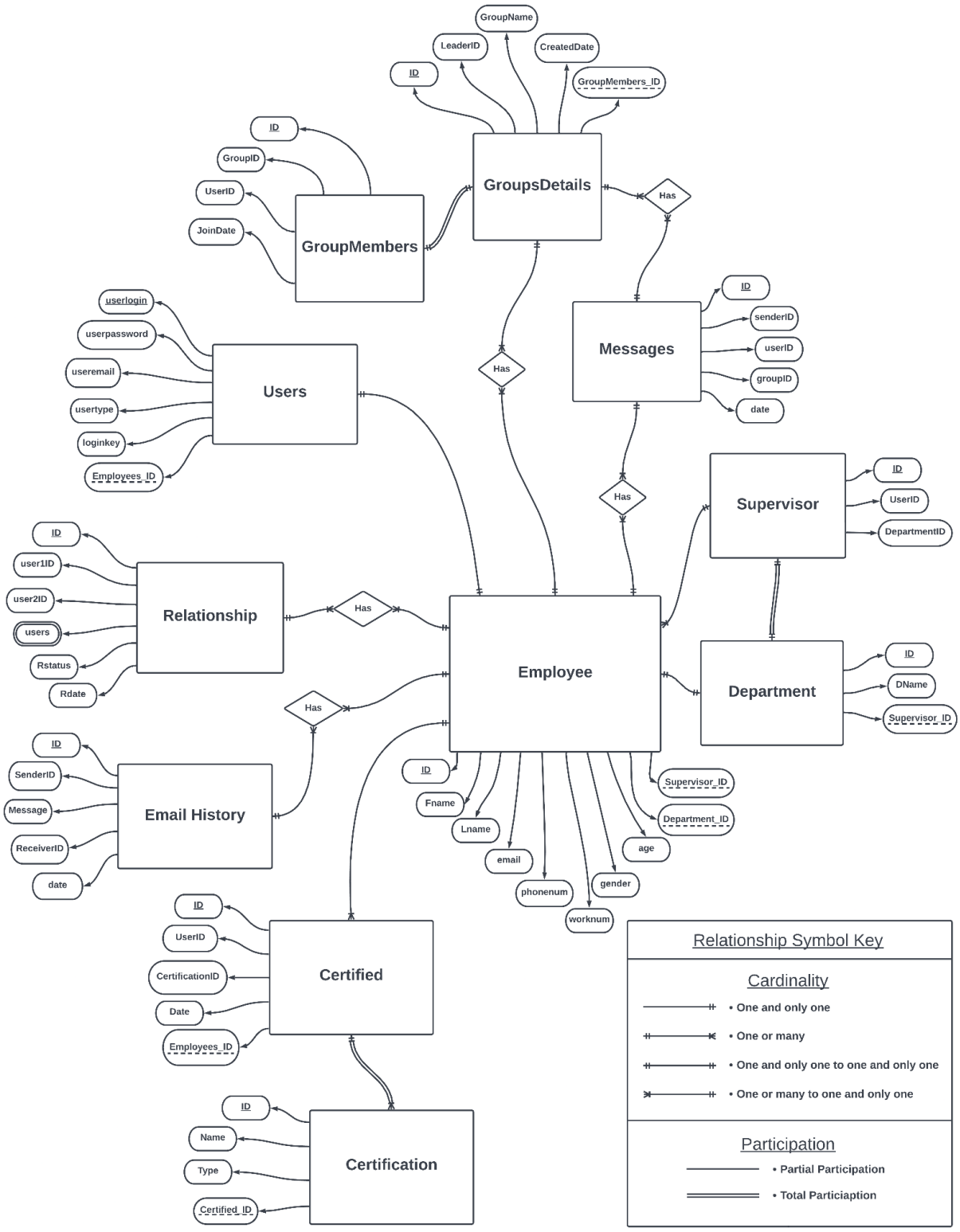


Figure 1: ER Model designed by Lucidchart [9]

Enhanced Entity Relationship Model (EER Model)

Keys and relationships play an integral part in relational databases. Keys are attributes (or a set of attributes) that help to identify a row uniquely in a table. Keys are used when you want to show that there is a relationship between different columns and tables of a relational database. There are several different types of keys: Primary Key, Candidate Key, Super Key, Foreign Key, Alternate Key, Composite Key, and Artificial Key. [6] A primary key contains values in a column that uniquely identify rows of data in a table. "A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table." [7] "A relationship, in the context of databases, is a situation that exists between two relational database tables when one table has a foreign key that references the primary key of the other table. Relationships allow relational databases to split and store data in different tables, while linking disparate data items." [8]

Our team was able to make use of numerous keys and relationships while designing our CMS, as seen in our EER Model in Figure 2. The employee entity has an integer ID which serves as its primary key. The employee entity has corresponding foreign keys for its relationships with Department and Supervisor. The Employee has a many to one relationship with the Department entity. The Department entity has an integer primary key called ID. It has a foreign key corresponding to the ID of its Supervisor. The Supervisor entity has a one to one relationship with the Department entity. It also has a one to one relationship, "are", with the Employee entity. The Supervisor entity has an integer ID as its primary key. It also has foreign keys corresponding to the Supervisor's Employee record and the Department they supervise. An Employee entity has a one to many relationship with the Certified entity. The Certified entity has an integer ID as its primary key as well as foreign keys corresponding to the Employee who received the certification and which Certification was received. The Certified entity has a one to one relationship with the Certification entity. The Relationship entity has an integer ID as its primary key. The Employee has a one to many relationship with the Relationship entity. The User entity has an integer as its primary key. There is a foreign key to the ID in the corresponding Employee record. The Employee entity has a one to one relationship with the User entity. The Message entity has a primary key which is an integer called ID. The Employee entity has a one to many relationship with the Message entity. The Group entity has an integer ID for its primary key. It also has a foreign key to the GroupMembers entity. The Group entity has a one to many relationship with the Message entity. The GroupMembers entity has a primary key which is an integer ID. The GroupMembers entity has a many to one relationship with the Group entity. The Email History has a unique integer ID as its

primary key. The Employee entity has a one to many relationship with the Email History entity.

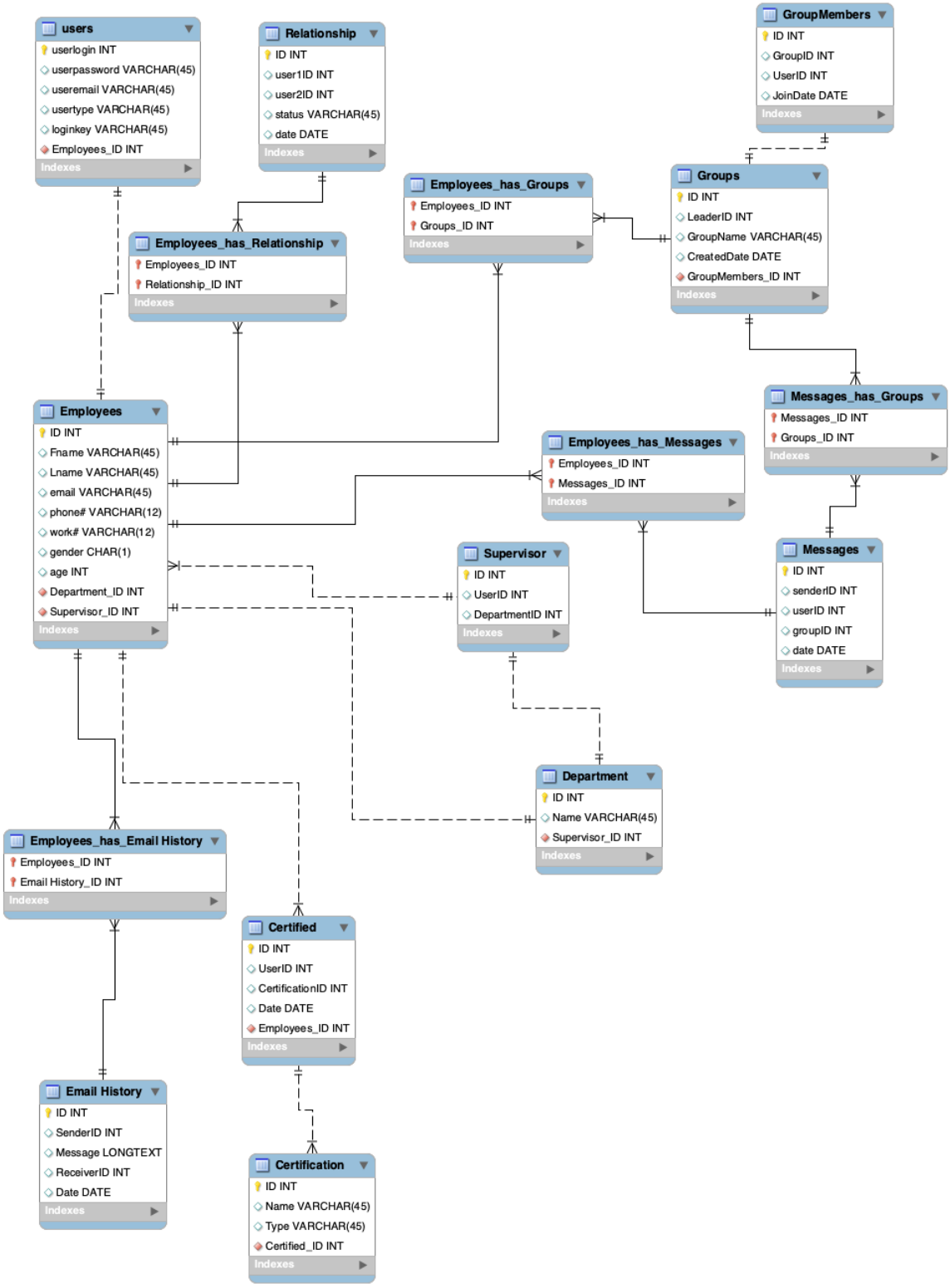
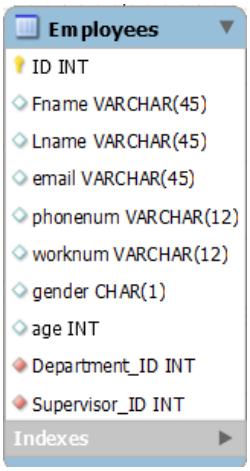
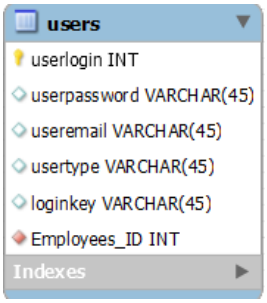
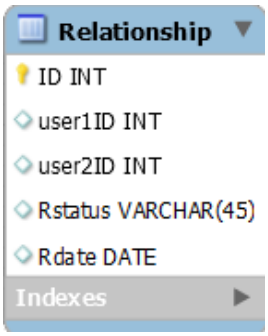
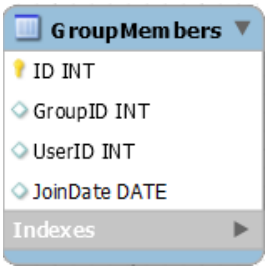
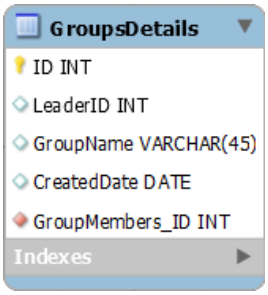
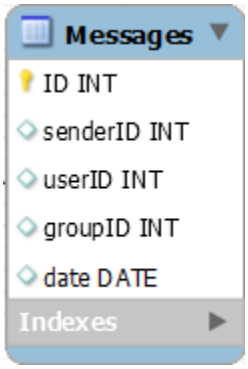
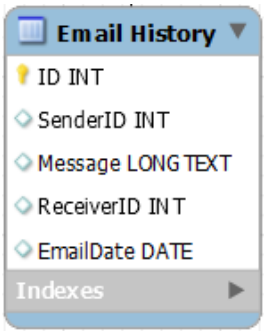
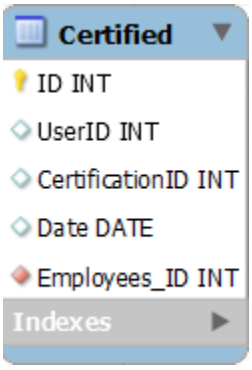
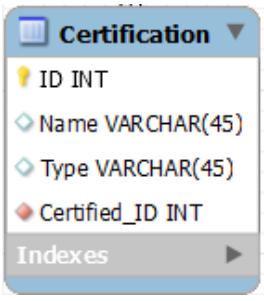


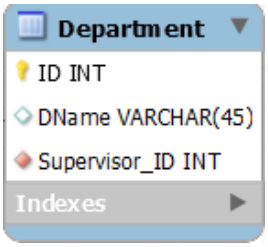
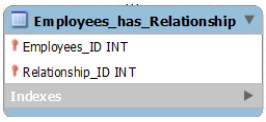
Figure 2: EER Model

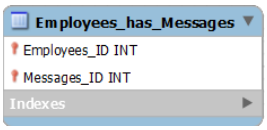
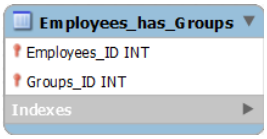
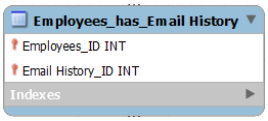
Database Development

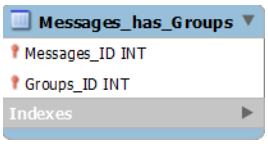
Table Name	Code	Table in EER	Description
Employee	-- Creates Employees Table CREATE TABLE Employees (ID INT NOT NULL PRIMARY KEY, Fname VARCHAR(45), Lname VARCHAR(45), email VARCHAR(45), phoneNum VARCHAR(12), WorkNum VARCHAR(12), gender CHAR(1), age INT, Department_ID INT NOT NULL, Supervisor_ID INT NOT NULL, FOREIGN KEY (Department_ID) REFERENCES Department (ID), FOREIGN KEY (Supervisor_ID) REFERENCES Supervisor (ID));		The Employee table has a primary key ID. It has foreign keys Department_ID and Supervisor_ID. The Department_ID shows the "belongs to" relationship between Employee and Department. The Supervisor_ID represents the "supervises" relationship between Employee and Supervisor. We chose the VARCHAR type to represent Fname, Name, email, phoneNum, and WorkNum. The gender attribute is represented by a single CHAR. Age is represented by an INT.
Users	-- Creates User Table CREATE TABLE users (userlogin INT NOT NULL PRIMARY KEY, userpassword VARCHAR(45) , useremail VARCHAR(45) , usertype VARCHAR(45) , loginkey VARCHAR(45) , Employees_ID INT NOT NULL, FOREIGN KEY (Employees_ID) REFERENCES Employees (ID));		The User table has userlogin, an INT, as its primary key. It has one foreign key, Employees_ID, which represents the "login" relationship between Employee and User. The VARCHAR type is used to represent the userpassword, useremail, usertype and loginkey.

Relationship	-- Creates Relationship Table CREATE TABLE Relationship (ID INT NOT NULL PRIMARY KEY, user1ID INT , user2ID INT , Rstatus VARCHAR(45) , Rdate DATE);		The Relationship table has a primary key called ID which is an INT. It has two more INT type attributes: user1ID and user2ID. The status of the relationship, Rstatus, is a VARCHAR. The date, Rdate, is a DATE type.
Group Members	-- Creates GroupMembers Table CREATE TABLE GroupMembers (ID INT NOT NULL PRIMARY KEY, GroupID INT , UserID INT , JoinDate DATE);		The GroupMembers table has a primary key called ID which is an INT. It has two more INT type attributes: GroupID and UserID. It also has a JoinDate attribute of type DATE.
Group Details	-- Creates GroupDetails Table CREATE TABLE GroupDetails (ID INT NOT NULL PRIMARY KEY , LeaderID INT , GroupName VARCHAR(45) , CreatedDate DATE , GroupMembers_ID INT NOT NULL, FOREIGN KEY (GroupMembers_ID) REFERENCES GroupMembers (ID));		The GroupsDetails table has a primary key called ID which is an INT. There is another INT type attribute called LeaderID. The GroupDetails has a name which is represented by a VARCHAR. The creation date is DATE type. There is a foreign key called GroupMembers_ID which represents the "contains" relationship between GroupDetails and GroupMembers.

Messages	-- Creates Messages Table CREATE TABLE Messages (ID INT NOT NULL PRIMARY KEY, senderID INT , userID INT , groupID INT , Messagedate DATE);		The Messages table has a primary key called ID of type INT. It also has senderID, userID, and groupID which are all INT types. It also has a date attribute of type DATE.
Email History	-- Creates EmailHistory Table CREATE TABLE EmailHistory (ID INT NOT NULL PRIMARY KEY, SenderID INT , Message LONGTEXT , ReceiverID INT , EmailDate DATE);		The EmailHistory table has a primary key called ID of type INT. It has a SenderID and ReceiverID which are also INT types. The message is LONG TEXT type so that the size limit is very large. The EmailDate is a DATE type.
Certified	-- Creates Certified Table CREATE TABLE Certified (ID INT NOT NULL PRIMARY KEY, UserID INT , CertificationID INT , Date DATE , Employees_ID INT NOT NULL, FOREIGN KEY (Employees_ID) REFERENCES Employees (ID));		The Certified table has a primary key called ID of type INT. It has a UserID and CertificationID which are INT types. It also has a Date which is a DATE type. There is a foreign key called Employees_ID which represents the "are" relationship between Certified and Employee.
Certification	-- Creates Certification Table CREATE TABLE Certification (ID INT NOT NULL PRIMARY KEY, Name VARCHAR(45) , Type VARCHAR(45) , Certified_ID INT NOT NULL, FOREIGN KEY (Certified_ID) REFERENCES Certified(ID));		The Certification table has a primary key called ID of type INT. The VARCHAR type is used to represent the Name and Type attributes. There is a foreign key called Certified_ID which represents the "are" relationship between Certified and Certification.

Supervisor	-- Creates Supervisor Table CREATE TABLE Supervisor (ID INT NOT NULL PRIMARY KEY, UserID INT, DepartmentID INT);		The Supervisor table has a primary key called ID of type INT. It has a UserID and DepartmentID which are both INTs.
Department	-- Creates Department Table CREATE TABLE Department (ID INT NOT NULL PRIMARY KEY, DName VARCHAR(45), Supervisor_ID INT NOT NULL, FOREIGN KEY (Supervisor_ID) REFERENCES Supervisor (ID));		The Department table has a primary key called ID which is an INT. It has a name which is represented by a VARCHAR. It has a foreign key called Supervisor_ID.
Employees_has_Relationship	-- Creates Employees_has_Relationship Table CREATE TABLE Employees_has_Relationship (Employees_ID INT NOT NULL , Relationship_ID INT NOT NULL , PRIMARY KEY (Employees_ID, Relationship_ID), FOREIGN KEY (Employees_ID) REFERENCES Employees (ID), FOREIGN KEY (Relationship_ID) REFERENCES Relationship(ID));		The Employees_has_Relationship table is used to represent the many to many relationship between the Employee and Relationship entities. Therefore it has two foreign keys: Employees_ID and Relationship_ID. These foreign keys together make up the primary key for the table.

Employees_has_Messages	-- Creates Employees_has_Messages Table CREATE TABLE Employees_has_Messages (Employees_ID INT NOT NULL, Messages_ID INT NOT NULL, PRIMARY KEY (Employees_ID, Messages_ID), FOREIGN KEY (Employees_ID) REFERENCES Employees (ID), FOREIGN KEY (Messages_ID) REFERENCES Messages (ID));		The Employees_has_Messages table is used to represent the many to many relationship between the Employee and Message entities. Therefore it has two foreign keys: Employees_ID and Messages_ID. These foreign keys together make up the primary key for the table.
Employees_has_Groups	-- Creates Employees_has_Groups Table CREATE TABLE Employees_has_Groups (Employees_ID INT NOT NULL, Groups_ID INT NOT NULL, PRIMARY KEY (Employees_ID, Groups_ID), FOREIGN KEY (Employees_ID) REFERENCES Employees(ID), FOREIGN KEY (Groups_ID) REFERENCES GroupDetails(ID));		The Employees_has_Groups table is used to represent the many to many relationship between the Employee and GroupDetails entities. Therefore it has two foreign keys: Employees_ID and Groups_ID. These foreign keys together make up the primary key for the table.
Employees_has_Email_History	-- Creates Employees_has_Email_History Table CREATE TABLE Employees_has_Email_History (Employees_ID INT NOT NULL, Email_History_ID INT NOT NULL, PRIMARY KEY (Employees_ID, Email_History_ID), FOREIGN KEY (Employees_ID)		The Employees_has_EmailHistory table is used to represent the many to many relationship between the Employee and EmailHistory entities. Therefore it has two foreign keys: Employees_ID and Email_History_ID. These foreign keys together

	REFERENCES Employees(ID), FOREIGN KEY (Email_History_ID) REFERENCES EmailHistory (ID));		make up the primary key for the table.
Messages_has_Groups	-- Creates Messages_has_Groups Table CREATE TABLE Messages_has_Groups (Messages_ID INT NOT NULL, Groups_ID INT NOT NULL, PRIMARY KEY (Messages_ID, Groups_ID), FOREIGN KEY (Messages_ID) REFERENCES Messages (ID), FOREIGN KEY (Groups_ID) REFERENCES GroupDetails (ID));		The Messages_has_Groups table is used to represent the many to many relationship between the Message and GroupDetails entities. Therefore it has two foreign keys: Messages_ID and Groups_ID. These foreign keys together make up the primary key for the table.

Loading data and performance enhancements

Loading data

When data is inserted into a table there is a possibility of conflicts between tables due to foreign key restraints. This issue can be overcome by turning off the FOREIGN_KEY_CHECK which turns off the checking between tables for foreign key restraints. Set Foreign_Key_Checks controls whether or not the foreign keys are checked by setting it to 0. By turning it off no errors will be present when the insertion code is run. The foreign key check is turned back on by setting it to 1 which allows for the check to be turned on for other insertions into the tables. This is shown in the following code:

```
SET FOREIGN_KEY_CHECKS=0;
-- Code for insertion into tables
SET FOREIGN_KEY_CHECKS=1;
```

Table Name	Code	Description
Employee	Insert into Employees(ID,FName,LName,email,phonenum,worknum,gender,age,department_ID,Supervisor_ID)values (0,'Ariana', 'Grande', 'agran@cms.com',0000000000,0000000001,'F',30,0,0), (1, 'Dwayne', 'TheRockJohnson', 'dther@cms.com',0000000002,0000000003,'M',50,1,0), (2, 'Barak', 'Obama', 'bobam@cms.com',0000000004,0000000005,'M',56,2,0), (3, 'Abraham', 'Lincoln', 'alinc@cms.com',0000000006,0000000007,'M',200,3,0), (4, 'Michael', 'Jackson', 'mjack@cms.com',0000000008,0000000009,'M',43,0,0), (5, 'Magic', 'Johnson', 'mjohn@cms.com',0000000010,0000000011,'M',54,0,1), (6, 'Jennifer', 'Lawrence', 'jlawr@cms.com',0000000012,0000000013,'F',36,6,2), (7, 'Taylor', 'Swift', 'tswif@cms.com',0000000014,0000000015,'F',27,7,3), (8, 'King', 'George III', 'kgeor@cms.com',0000000016,0000000017,'M',300,8,0), (9, 'Queen', 'Elizabeth', 'qeliz@cms.com',0000000018,0000000019,'F',96,1,0), (10,'Megan', 'Markel', 'mmark@cms.com',0000000020,0000000021,'F',29,5,1), (11,'Derek', 'Jeter', 'djete@cms.com',0000000022,0000000023,'M',42,4,3), (12,'Will', 'Smith',	Inserts user data into table which includes employee information

	'wsmit@cms.com',0000000024,0000000025,'M',55,3,3), (13,'Meryl', 'Streep', 'mstre@cms.com',0000000026,0000000027,'F',64,2,2), (14,'Lizzo', Null, 'lizzo@cms.com',0000000028,0000000029,'F',28,9,2);	
Users	Insert into users(userlogin,userpassword,useremail,usertype,loginkey, employees_ID)values (100,'admin','agran@cms.com','admin','12345',0), (101,'admin1','dther@cms.com','admin','13345',1), (102,'admin2','bobam@cms.com','admin','14345',2), (103,'admin3','alinc@cms.com','admin','15345',3), (104,'employee0','mjack@cms.com','employee','16345',4), (105,'employee1','mjohn@cms.com','employee','17345',5), (106,'employee2','jlawr@cms.com','employee','18345',6), (107,'employee3','tswif@cms.com','employee','19345',7), (108,'employee4','kgeor@cms.com','employee','20345',8), (109,'employee5','qeliz@cms.com','employee','21345',9), (110,'employee6','mmark@cms.com','employee','22345',10), (111,'employee7','djete@cms.com','employee','23345',11), (112,'employee8','wsmit@cms.com','employee','24345',12), (113,'employee9','mstre@cms.com','employee','25345',13), (114,'employee10','lizzo@cms.com','employee','26345',14);	Inserts user data used for credentials and access level
Certification	Insert into Certification(ID,name,type,certified_ID)values (0,'Customer Service','Admin',null), (1,'Networking','IT',null), (2,'Internal Transfers','HR',null), (3,'Organizational Excellence','Employee Retention',null), (4,'Math','Accounting',null), (5,'New Hire Screening','Employee Transfers',null), (6,'Lab Equipment','Research',null), (7,'Word','Office365',null), (8,'Executive Assistance','Admin',null), (9,'Attendance','Admin',null);	Inserts certifications for employees in table
Certified	Insert into certified(ID,userID,certificationID, certdate,employees_ID)values (0,0,0,'2022-05-20',0), (1,0,1,'2022-05-20',0), (2,0,2,'2022-05-20',0), (3,0,3,'2022-05-20',0), (4,1,4,'2022-05-20',0), (5,1,5,'2022-05-20',0), (6,2,6,'2022-05-20',0), (7,2,7,'2022-05-20',0),	Inserts employees who have specific certifications

	(8,3,8,'2022-05-20',0), (9,3,9,'2022-05-20',0);	
Department	Insert into Department(ID, DName, Supervisor_ID)values (0, 'Administration', 0), (1, 'Human Resources', 0), (2, 'Information Technology', 1), (3, 'Quality Control',2), (4, 'Marketing',2), (5, 'Telecommunications',1), (6, 'Programming', 1), (7, 'Customer Service',2), (8, 'General Services',3), (9, 'Maintenance',3);	Inserts department data to store department names and supervisor IDs
Supervisor	Insert into Supervisor(ID, UserID, DepartmentID) values (0,0,0), (1,1,2), (2,2,3), (3,3,8);	Inserts employees who are supervisors
Messages	Insert into Messages(ID,senderID,userID,groupID,message,message date)values (0,0,1,0,'We are going to play a company game of telephone pass along my message the next day "Pancakes"','2022-5-20'), (1,1,2,0,'Passing along the CEOs message "Pancakes"','2022-5-21'), (2,2,3,0,'Passing along the CEOs message "Pancakes"','2022-5-22'), (3,3,4,0,'Passing along the CEOs message "Pancakes"','2022-5-23'), (4,4,5,0,'Passing along the CEOs message "Paincakes"','2022-5-24'), (5,5,6,0,'Passing along the CEOs message "Paincakes"','2022-5-25'), (6,6,7,0,'Passing along the CEOs message "Paincakes"','2022-5-26'), (7,7,8,0,'Passing along the CEOs message "coffeecakes"','2022-5-27'), (8,8,9,0,'Passing along the CEOs message "coffeecakes"','2022-5-28'), (9,9,10,0,'Passing along the CEOs message "coffeecorn"','2022-5-29'), (10,10,11,0,'Passing along the CEOs message "cornkernals"','2022-5-30'), (11,11,12,0,'Passing along the CEOs message "cornkernals"','2022-5-31'), (12,12,13,0,'Passing along the CEOs message "cornmeal"','2022-6-01'), (13,13,14,0,'Passing along the CEOs message "cornmeal"','2022-6-02'), (14,14,0,0,'Your message to the company said cornbread','2022-6-03'), (15,0,1,0,'That is not the word I gave you and as such	Inserts messages into table used to track messages between users

	you are now terminated at this company','2022-6-04'), (16,1,2,0,'They were basically the same thing :(','2022-6-05');	
Group Members	Insert into groupmembers(ID,groupid,userid,joindate)values (0,0,0,'2022-05-20'), (1,0,1,'2022-05-20'), (2,0,2,'2022-05-20'), (3,0,3,'2022-05-20'), (4,0,4,'2022-05-20'), (5,1,8,'2022-05-22'), (6,1,9,'2022-05-22'), (7,1,7,'2022-05-22'), (8,2,11,'2022-05-24'), (9,2,10,'2022-05-24'), (10,3,3,'2022-05-26'), (11,3,14,'2022-05-26'), (12,4,6,'2022-05-28'), (13,4,5,'2022-05-28');	Inserts group members to track what users are part of which groups
Group Details	Insert into GroupDetails(ID,leaderID,groupname,createddate,groupmembers_ID)values (0,0,'I fired The Rock','2022-5-20',0), (1,1,'Im fired','2022-6-10',1), (2,2,'Wow','2022-5-20',1), (3,2,'Team Meetings','2022-5-25',2), (4,4,'Employee Retention','2022-5-16',4), (5,2,'TGIF','2022-5-30',1), (6,5,'Hello World','2022-5-21',3), (7,3,'CMS542','2022-5-15',1), (8,10,'Project Assignment','2022-5-28',0), (9,5,'Project Report Phase 4','2022-6-21',0);	Insert group details which include the name and the group member ID
Email History	Insert into emailhistory(ID,SenderID,Message,ReceiverID,EmailDate) values (0,0,'I will give you one last chance pass along my message "telephone"',1,'2022-06-04'), (1,1,'Passing along the CEOs message "telephone"',2,'2022-06-05'), (2,2,'Passing along the CEOs message "telephone"',3,'2022-06-06'), (3,3,'Passing along the CEOs message "telephone"',4,'2022-06-07'), (4,4,'Passing along the CEOs message "telephone"',5,'2022-06-08'), (5,5,'Passing along the CEOs message "telephone"',6,'2022-06-09'), (6,6,'Passing along the CEOs message "telephone"',7,'2022-06-10'), (7,7,'Passing along the CEOs message "telephone"',8,'2022-06-11'), (8,8,'Your message was "telephone"',0,'2022-06-12'), (9,0,'Congrats it looks like you will no longer be terminated',1,'2022-06-13');	Inserts email history into table to store email history between users

Manipulating data

Data before using alter to add a column:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 3: Data before using alter to add column

SQL command: *alter table employees add address VARCHAR(45);*

Data after using alter to add a column:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID	address
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0	NULL
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0	NULL
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0	NULL
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0	NULL
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0	NULL
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1	NULL
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2	NULL
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3	NULL
8	King	George III	kgeor@cms.com	16	17	M	300	8	0	NULL
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0	NULL
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1	NULL
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3	NULL
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3	NULL
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2	NULL
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2	NULL

Figure 4: Data after using alter to add a column

Data before using alter to drop a column:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID	address
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0	NULL
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0	NULL
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0	NULL
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0	NULL
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1	NULL
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2	NULL
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3	NULL
8	King	George III	kgeor@cms.com	16	17	M	300	8	0	NULL
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0	NULL
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1	NULL
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3	NULL
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3	NULL
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2	NULL
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2	NULL

Figure 5: Data before using alter to drop a column

SQL command: *alter table employees drop address;*

Data after using alter to drop a column:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 6: Data after using alter to drop a column

Data before using alter to change the type:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 7: Data before using alter to change the type

SQL command: *alter table employees modify age float;*

After using alter to change the type of age to float, it was not obvious so a new record was added with age as a float:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2
15	Theresa	Gundel	tgund@cms.com	0	1	F	23.5	0	0

Figure 8: Data after using alter to change the type

Data before using alter to rename a column:

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 9: Data before using alter to rename a column

SQL command: *alter table employees rename column phoneNum to phoneNumber;*

Data after using alter to rename a column:

ID	Fname	Lname	email	phoneNumber	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 10: Data after using alter to rename a column

Data before reordering 5 columns:

ID	Fname	Lname	email	phoneNumber	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2

Figure 11: Data before reordering 5 columns

SQL commands:

alter table employees modify Fname VARCHAR(45) after Lname;
alter table employees modify phoneNumber VARCHAR(12) after Lname;
alter table employees modify gender CHAR(1) after ID;
alter table employees modify age int after phoneNumber;
alter table employees modify WorkNum VARCHAR(12) after Fname;

Data after reordering 5 columns:

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	30	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	5	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	7	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	11	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	17	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	19	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	25	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	27	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 12: Data after reordering 5 columns

Data before using update to change single numerical record:

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	30	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	5	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	7	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	11	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	17	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	19	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	25	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	27	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 13: Data before using update to change a single numerical record

SQL command: *update employees set age = 25 where id = 0;*

Data after using update to change single numerical record:

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	5	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	7	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	11	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	17	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	19	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	25	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	27	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 14: Data after using update to change a single numerical column

Data before using update to change multiple string records:

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	5	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	7	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	11	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	17	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	19	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	25	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	27	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 15: Data before using update to change multiple string records

SQL command: *update employees set workNum = '1234567890' where age > 50;*

Data after using update to change multiple string records:

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	1234567890	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	1234567890	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	1234567890	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	1234567890	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	1234567890	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	1234567890	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	1234567890	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 16: Data after using update to change multiple string records

Data before using pattern matching to update records (first):a

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	1	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	1234567890	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	1234567890	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	1234567890	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	1234567890	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	1234567890	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	1234567890	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	1234567890	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 17: Data before using pattern matching to update records (first)

SQL command: *update employees set workNum = '0987654321' where Fname like 'A%';*

Data after using pattern matching to update records (first):

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	0987654321	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	1234567890	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	0987654321	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	1234567890	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	1234567890	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	1234567890	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	1234567890	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	1234567890	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 18: Data after using pattern matching to update records (first)

Data before using pattern matching to update records (second):

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	0987654321	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	1234567890	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	0987654321	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	1234567890	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	14	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	1234567890	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	1234567890	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	24	55	Will	1234567890	wsmit@cms.com	3	3
13	F	Streep	26	64	Meryl	1234567890	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 19: Data before using pattern matching to update records (second)

SQL command: *update employees set phoneNumber = '0001112222' where email like '_s%@cms.com';*

Data after using pattern matching to update records (second):

ID	gender	Lname	phoneNumber	age	Fname	WorkNum	email	Department_ID	Supervisor_ID
0	F	Grande	0	25	Ariana	0987654321	agran@cms.com	0	0
1	M	TheRockJohnson	2	50	Dwayne	3	dther@cms.com	1	0
2	M	Obama	4	56	Barak	1234567890	bobam@cms.com	2	0
3	M	Lincoln	6	200	Abraham	0987654321	alinc@cms.com	3	0
4	M	Jackson	8	43	Michael	9	mjack@cms.com	0	0
5	M	Johnson	10	54	Magic	1234567890	mjohn@cms.com	0	1
6	F	Lawrence	12	36	Jennifer	13	jlawr@cms.com	6	2
7	F	Swift	0001112222	27	Taylor	15	tswif@cms.com	7	3
8	M	George III	16	300	King	1234567890	kgeor@cms.com	8	0
9	F	Elizabeth	18	96	Queen	1234567890	qeliz@cms.com	1	0
10	F	Markel	20	29	Megan	21	mmark@cms.com	5	1
11	M	Jeter	22	42	Derek	23	djete@cms.com	4	3
12	M	Smith	0001112222	55	Will	1234567890	wsmit@cms.com	3	3
13	F	Streep	0001112222	64	Meryl	1234567890	mstre@cms.com	2	2
14	F	NULL	28	28	Lizzo	29	lizzo@cms.com	9	2

Figure 20: Data after using pattern matching to update records (second)

Optimizing database

In this section of database optimization, we are using **insert query** and representing the process of database optimization. So, when we write a query for the insertion of single or multiple set of data into the database, the time required for insertion is determined by following factors; Connecting, Sending query to server, Parsing query, Inserting row: (1 × size of row), Inserting indexes: (1 × number of indexes), Closing.

In the insertion of records into the employee table, we are using the insert query multiple times for each insertion of record. So, this type of insertion will consume additional time in inserting the records since this query has to go through all of the factors like connecting, sending query to server, parsing query, inserting row, inserting indexes and closing.

For the insertion of multiple instances using a single insert query, this reduces the time for the execution since we only have to go through the process of insertion into the database all at once.

Query statistics:

Checking the optimization by inserting the single instance and multiple instances.

Query:

```
Insert into
Employees(ID,FName,LName,email,phonenum,worknum,gender,age,department_ID,S
upervisor_ID)values
(16,'Sadik','Hamucru','sadik@cms.com',9893193191,0000000036,'M',24,7,3);
```

Execution statistics for single set of values:

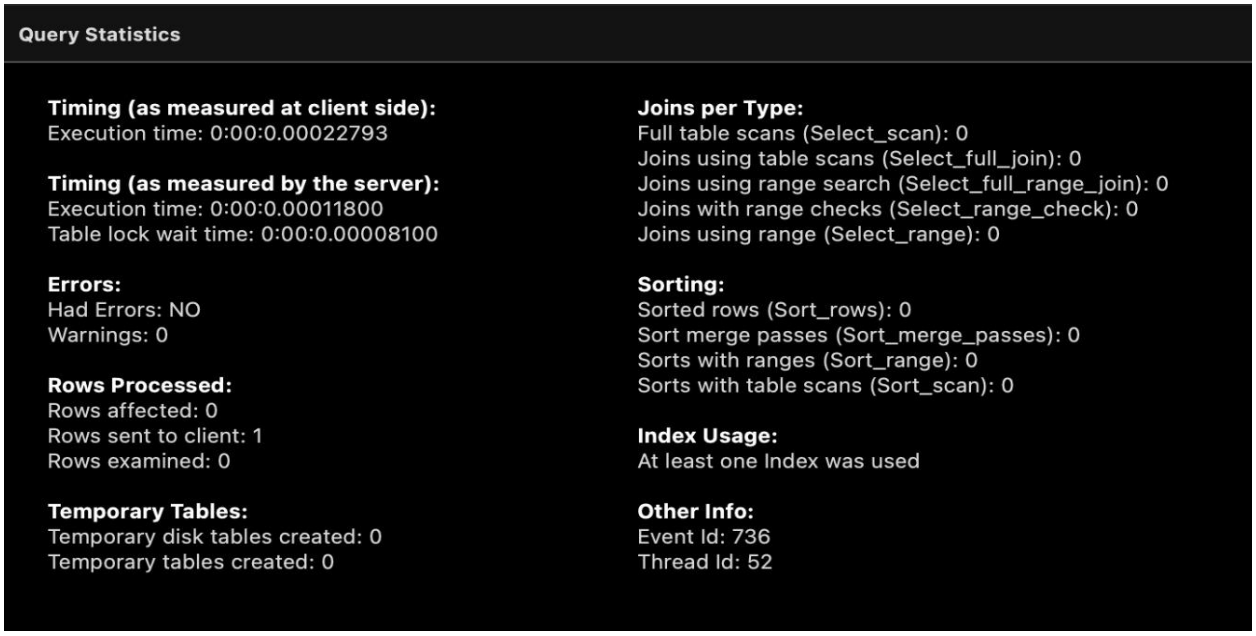


Figure 21: Query stats for Insert query for single set of values

Execution statistics for multiple set of values:

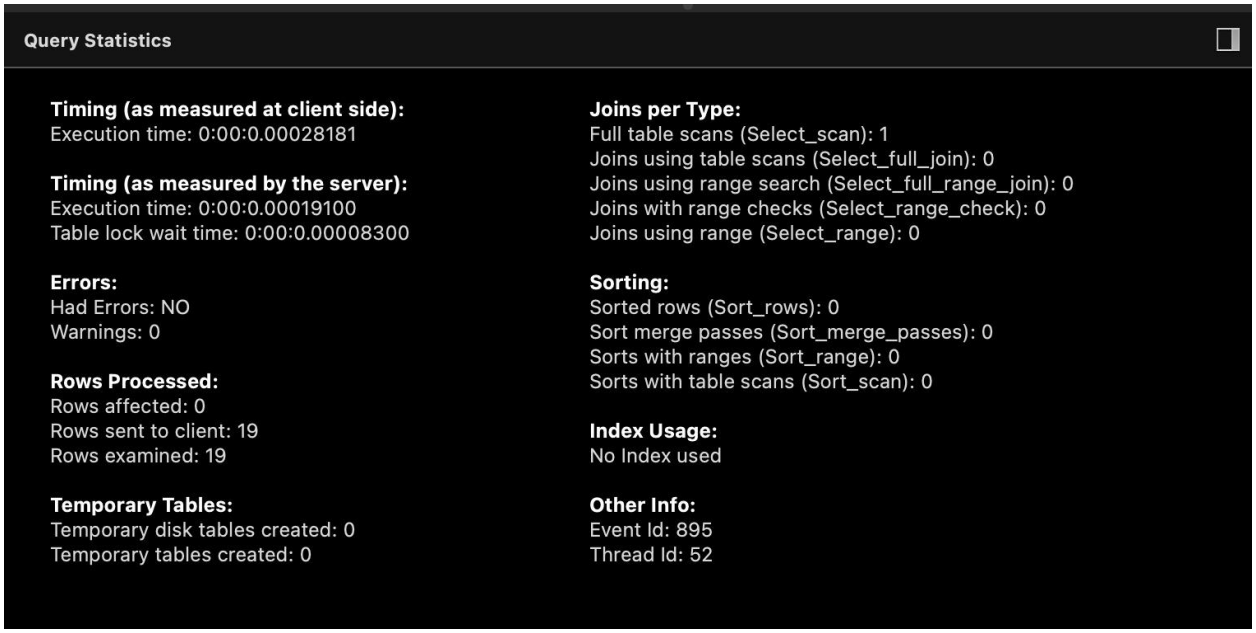


Figure 22: Query stats for Insert query for multiple set of values

Select Query:

In this section of database optimization, we are evaluating the **Select query** and for that purpose we use the employees table to return the last name and first names with a specific condition.

Employee table:
The below table represents the columns in the employee table with the data.

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 23: Data before using select query in the table of Employee

Query:

Here we are returning the LastName and FirstName of the male employees who have an age greater than 30.

```
SELECT Lname,Fname from Employees where age>=30 and gender='M';
```

Result for Select query:

Lname	Fname
TheRockJohnson	Dwayne
Obama	Barak
Lincoln	Abraham
Jackson	Michael
Johnson	Magic
George III	King
Jeter	Derek
Smith	Will

Figure 24: Data returned after using select query on the table of Employee

Execution Plan and Query Stats for the select query are shown below.

Execution Plan:

The below diagram represents the way the select query is used on the employees table.

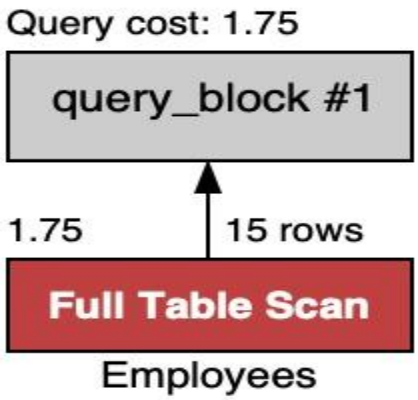


Figure 25: Execution flow for Select query

Query statistics:

Checking for the statistics for the optimization by joining the Employee and Department table.

Timing (as measured at client side):
Execution time: 0:00:0.00580788

Timing (as measured by the server):
Execution time: 0:00:0.00563100
Table lock wait time: 0:00:0.00282900

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 8
Rows examined: 15

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

Joins per Type:
Full table scans (Select_scan): 1
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:
No Index used

Other Info:
Event Id: 107
Thread Id: 51

Figure 26: Query stats after using Select query in the table of Employee

Join Query:

In this section of the **Join query**, we joined two different tables which have the same column name with help of Join and Group By command.

Using count as an aggregate function, we are determining the count of employees in each department by joining 2 tables Employees & Department, and grouped the results by Department name using the GroupBy clause.

Employee table:

The below table represents the columns in the employee table with the data.

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 27: Data before using join query in the table of Employee

Department table:

The below table represents the columns in the department table with their data respectively.

ID	DName	Supervisor_ID
0	Administration	0
1	Human Resources	0
2	Information Technology	1
3	Quality Control	2
4	Marketing	2
5	Telecommunications	1
6	Programming	1
7	Customer Service	2
8	General Services	3
9	Maintenance	3
NULL	NULL	NULL

Figure 28: Data returned before using join query in the table of Department

Query:

Here we are viewing the count of employees ID and Department name by using GROUPBY and JOIN.

```
SELECT count(Employees.ID), Dname from Employees JOIN Department ON
Employees.Department_ID= Department.ID GROUP BY (Dname) ;
```

Result for JOIN:

count(Employees.ID)	Dname
3	Administration
2	Human Resources
2	Information Technology
2	Quality Control
1	Marketing
1	Telecommunications
1	Programming
1	Customer Service
1	General Services
1	Maintenance

Figure 29: Data returned after using join query on the table of Department

Execution Plan and Query Stats for the above query are shown below.

Execution Plan:

The below diagram represents the way it joined the two different tables employees and departments.

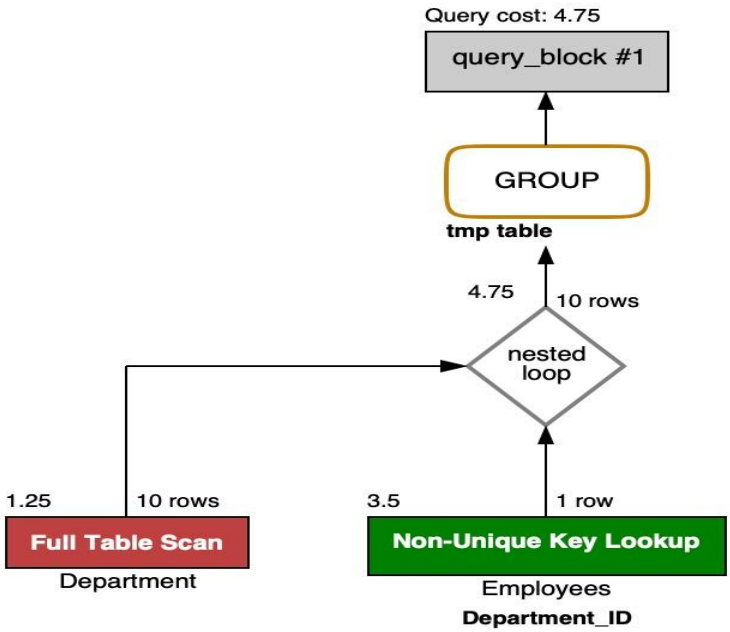


Figure 30: Execution flow for Join query

Query statistics:

Checking for the statistics for the optimization by joining the Employee and Department table.

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00409913

Timing (as measured by the server):
Execution time: 0:00:0.00394000
Table lock wait time: 0:00:0.00189500

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 10
Rows examined: 35

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 1

Joins per Type:
Full table scans (Select_scan): 1
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:
No Index used

Other Info:
Event Id: 123
Thread Id: 51

Figure 31: Query stats for Join query

Trigger:

In the below section, we are evaluating the trigger on the employee table.

Employee table:

The below table represents the columns in the employee table with the data.

ID	Fname	Lname	email	phoneNum	WorkNum	gender	age	Department_ID	Supervisor_ID
0	Ariana	Grande	agran@cms.com	0	1	F	30	0	0
1	Dwayne	TheRockJohnson	dther@cms.com	2	3	M	50	1	0
2	Barak	Obama	bobam@cms.com	4	5	M	56	2	0
3	Abraham	Lincoln	alinc@cms.com	6	7	M	200	3	0
4	Michael	Jackson	mjack@cms.com	8	9	M	43	0	0
5	Magic	Johnson	mjohn@cms.com	10	11	M	54	0	1
6	Jennifer	Lawrence	jlawr@cms.com	12	13	F	36	6	2
7	Taylor	Swift	tswif@cms.com	14	15	F	27	7	3
8	King	George III	kgeor@cms.com	16	17	M	300	8	0
9	Queen	Elizabeth	qeliz@cms.com	18	19	F	96	1	0
10	Megan	Markel	mmark@cms.com	20	21	F	29	5	1
11	Derek	Jeter	djete@cms.com	22	23	M	42	4	3
12	Will	Smith	wsmit@cms.com	24	25	M	55	3	3
13	Meryl	Streep	mstre@cms.com	26	27	F	64	2	2
14	Lizzo	NULL	lizzo@cms.com	28	29	F	28	9	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 32: Data before using trigger query on the table of Employee

Query:

Here we are using trigger on the Employees table and using the age attribute to check and insert the data if the age is greater than 18.

```
delimiter //
CREATE TRIGGER age_check AFTER INSERT
ON Employees
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Alert - Employee age is less than 18.';
END IF; //
delimiter ;
```

The below insert query is used to validate the trigger query that we have written for the employees table.

```
Insert into
Employees(ID,FName,LName,email,phonenum,worknum,gender,age,department_ID,Supervisor_ID)values
(16,'Invalid','Data','invalidData@cms.com',123456789,0000000050,'M',15,7,3);
```

Result for Trigger:

48	22:26:40	Explain Analyze Insert into Employees(ID,FName,LNam...	1 row(s) returned	0
49	22:27:24	Explain Analyze Insert into Employees(ID,FName,LNam...	1 row(s) returned	0
50	22:27:40	CREATE TRIGGER age_check AFTER INSERT ON Empl...	0 row(s) affected	0
51	22:27:54	Insert into Employees(ID,FName,LName,email,phonenu...	Error Code: 1644. Alert - Employee age is less than 18.	0

Figure 33: Result for the trigger query used on the table of Employee

Execution Plan for the above query are shown below.

Execution Plan:

The below diagram represents the way trigger is used on the employees table.

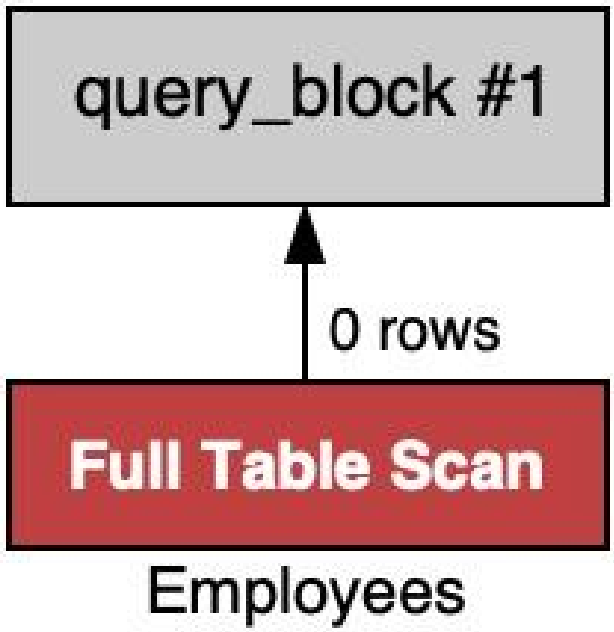


Figure 34: Execution flow for Trigger query

Graphical User Experience Design

The flow diagram below is a visual representation of the views available to end users. Additional views may be available to specific users. Authenticated employees have the least available permissions. Users permissioned as admins have the most views available to them once they are authenticated. Normal employees can access all of their data from the side nav pages. Normal employees may also change their password from the settings screen. Admin Employees have access to the same views as normal employees with the addition of the Admin panel. If a user is authenticated as an admin, the Admin user view will be available in the side navigation bar.

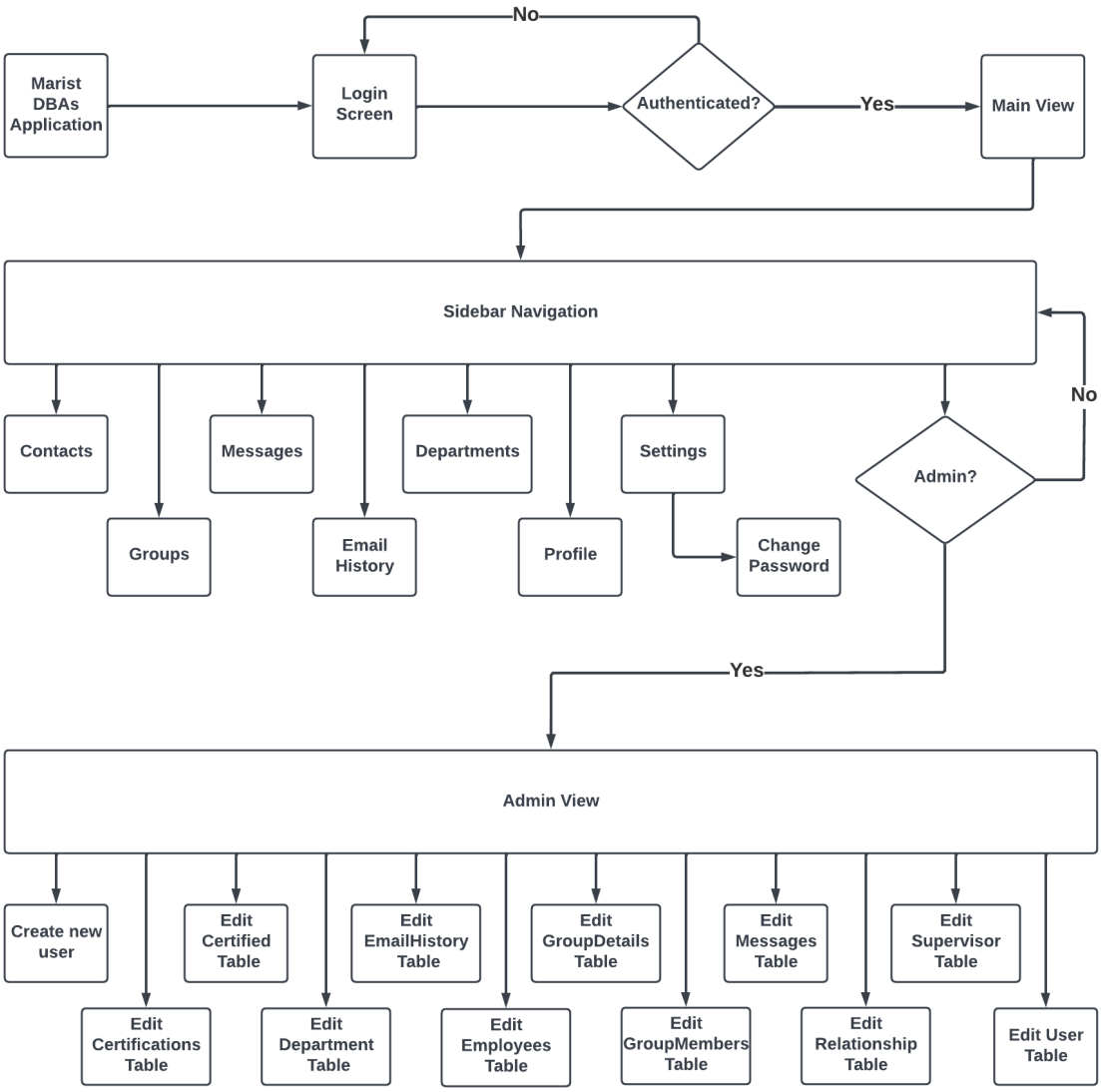


Figure 35: Graphical User Interface Flow Diagram designed by Lucidchart [9]

Graphical User Interface Design

The graphical user interface was designed using Node.js as a backend server for the application. The javascript framework packages electron.js and react.js were used to design and implement the front end of the application. MySQL.js is the database backend package being used to store and query user data.

Connection to database:

The connection to the database is initiated from the backend of the application by Node.js. The mysql package is required to import the methods and tools needed to create a connection. In this specific case, the database is running locally so the host is set to localhost. A special user is created to access the database with the username "marist_dbas" and the password "rootroot". By default the "myCms" schema is selected for use with this application.

This database connection code can be viewed below.

```
const mysql = require('mysql');
const con = mysql.createConnection({
  host: 'localhost',
  user:'marist_dbas',
  password:'rootroot',
  database: 'myCms'
});

con.connect(function(err) {
  if (err) {
    console.log("failed connection.");
    console.log(err);
  } else {
    console.log("created connection to database");
  }
});

module.exports = {
  con,
}
```

Custom Utils:

The custom utils main function is to encrypt and decrypt user passwords for login verification. When the user's profile is created their passwords are sent to the database encrypted using a caesar cypher so that in the event of a data leak their passwords are secure. When the user is logged in the passwords are unencrypted for verification of user. If the entered password matches the cyphered password located in the user table then the user is granted access to the application.

```
function newID() {
  return (Date.now() + Math.random()).toString().substr(4);
}

function formatDate(date) {
  let newDate = new Date(date);
  let m = newDate.getMonth();
  let d = newDate.getDate();
  let y = newDate.getFullYear().toString().slice(2,4);
  return m + "/" + d + "/" + y;
}

function caesarEncrypt(str, shift) {
  let encrypted = "";
    for (var i = 0; i < str.length; i++) {
      let code = str.charCodeAt(i);
      if (code >= 65 && code <= 90) {
        encrypted += String.fromCharCode(((code - 65 + shift) % 26) + 65);
      } else if (code >= 97 && code <= 122) {
        encrypted += String.fromCharCode(((code - 97 + shift) % 26) + 97);
      } else {
        encrypted += str.charAt(i).toString();
      }
    }
    return String(encrypted);
}

function caesarDecrypt(str, shift) {
    let plain = '';
    shift = (26 - shift) % 26;
    plain = caesarEncrypt(str, shift);
    return plain;
}
```

```
module.exports = {
  newID,
  formatDate,
  caesarEncrypt,
  caesarDecrypt,};
```

Login Page:

The image below is a screenshot of the Login screen for the Marist DBA application. Users can enter their email and password on this screen to access the application. When the "Login" button is pressed, the "Email" and "Password" data is sent to the backend and checked against the records in the database. If a match is found, the user is logged in with their respective "Employee" or "Admin" role.

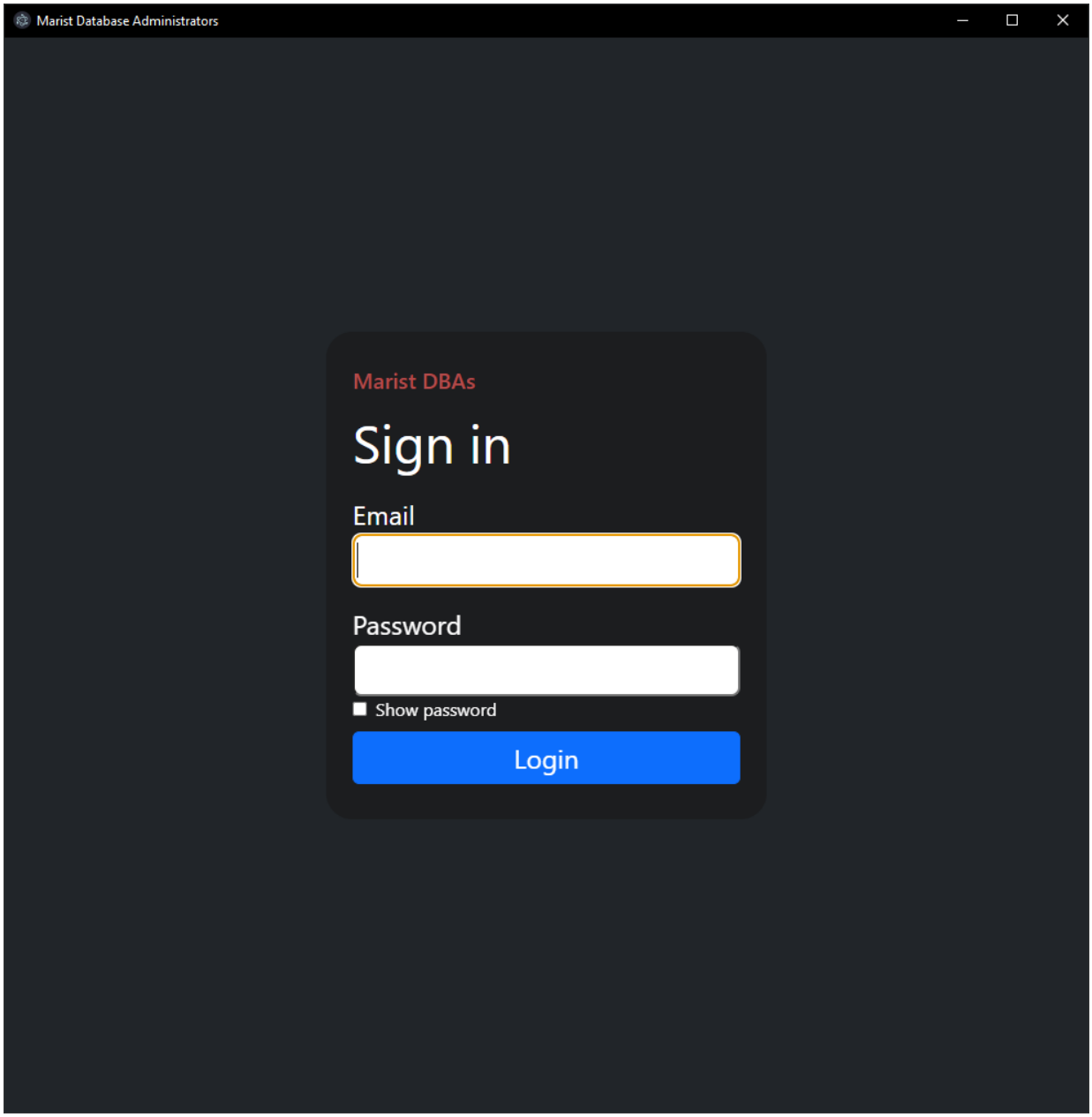


Figure 36: Login Page

The relative code for the Login screen can be seen below in blue.

```
import $ from 'jquery';
import "../css/Login.css";
const utils = require('../utils/utils.js');

const Login = (props) => {
  function handleSubmit(e) {
    e.preventDefault();
    window.dbConnection.checkLogin({
      userEmail: $("#userEmail").val(),
      password: $("#password").val()
    }).then((result) => {
      console.log(result);
      if (result.authenticated) {
        props.setAdmin(result.admin);
        props.loggedIn(true);
        props.setUserEmail($("#userEmail").val());
      } else {
        let loginError = "<p class='error'>Incorrect Email or Password</p>";
        $("#loginForm").before(loginError);
      }
    });
  }

  function togglePassword(e) {
    let inputType = $("#password").attr("type");
    if (inputType === "text") {
      $("#password").attr("type", "password");
    } else {
      $("#password").attr("type", "text");
    }

    let inputTypeLabel = $("label.password-helper").text();
    if (inputTypeLabel === "Show password") {
      $("label.password-helper").text("Hide password");
    } else {
      $("label.password-helper").text("Show password");
    }
  }
```

```
  return (
    <>
      <div id="login" className="container">
        <div>
          <h4 id="loginTitle" className="text-start">Marist DBAs</h4>
          <div className="header">
            Sign in
          </div>
          <form id="loginForm" onSubmit={handleSubmit}>
            <div>
              <div>
                <label>Email</label>
              </div>
              <div>
                <input id="userEmail" name="userEmail" className="fullWidth mb-3"
label="userEmail" type="text" required></input>
              </div>
              <div>
                <label>Password</label>
              </div>
              <div>
                <input id="password" name="password" className="fullWidth"
label="password" type="password" required></input>
              </div>
              <div>
                <input type="checkbox" onClick={togglePassword}></input>
                <label className="password-helper">Show password</label>
              </div>
            </div>
            <div>
              <button className="btn-primary btn" type="submit">Login</button>
            </div>
          </form>
        </div>
      </div>
    </>)};

export default Login;
```

Main Menu Page:

The image below is a screenshot of the Main Menu page that users will see when they are logged in. The current user's email is listed in the top right corner to identify the current account being used. Any of the pages listed in the side navigation bar can be accessed from the main menu.

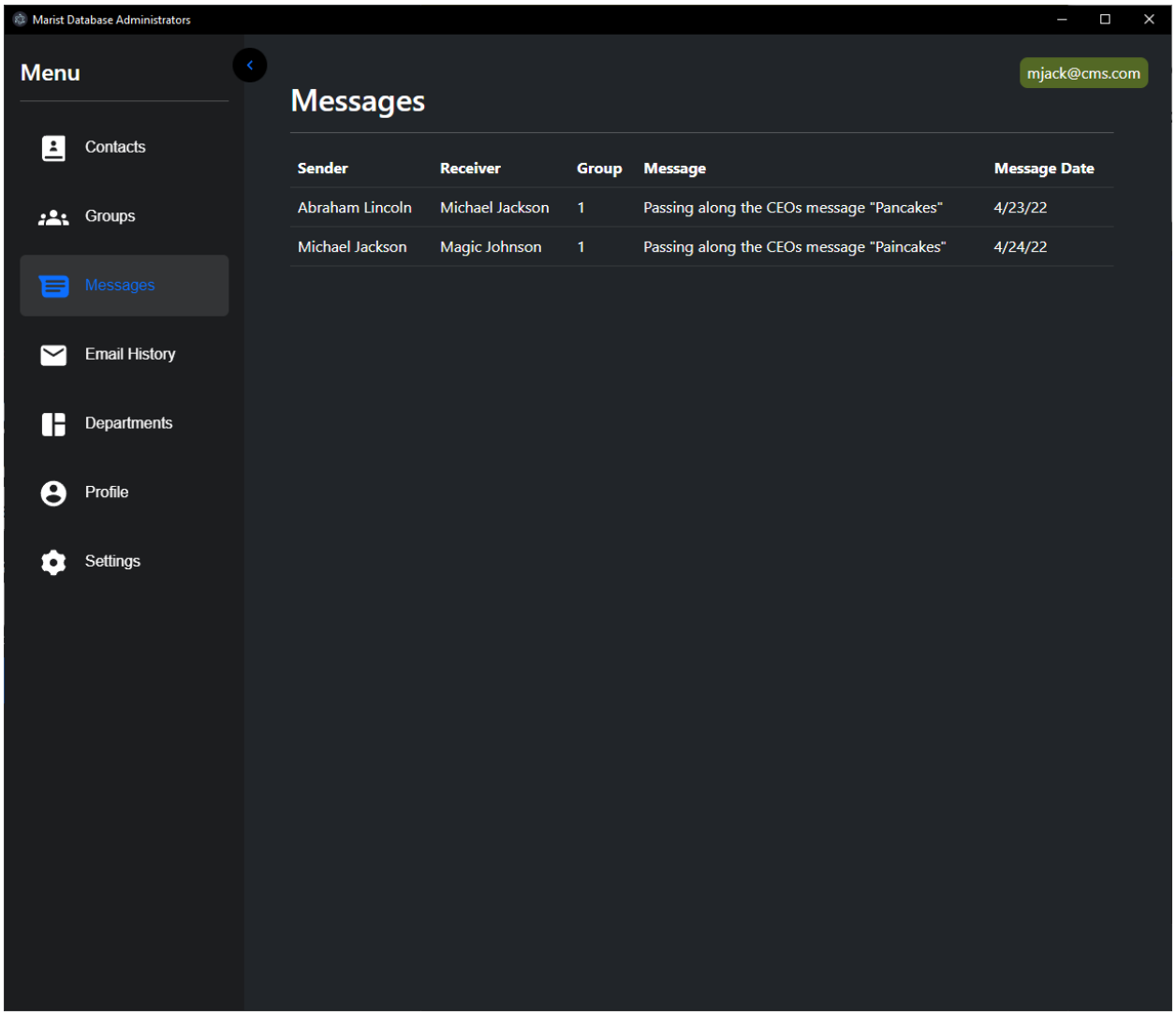


Figure 37: Main Menu Page - User View

The image below is a screenshot of the Main Menu page that users will see when they are logged in with the admin role. Just like normal users, the current user's email is listed in the top right corner with the text "admin mode" displayed just below.

Contacts Page:

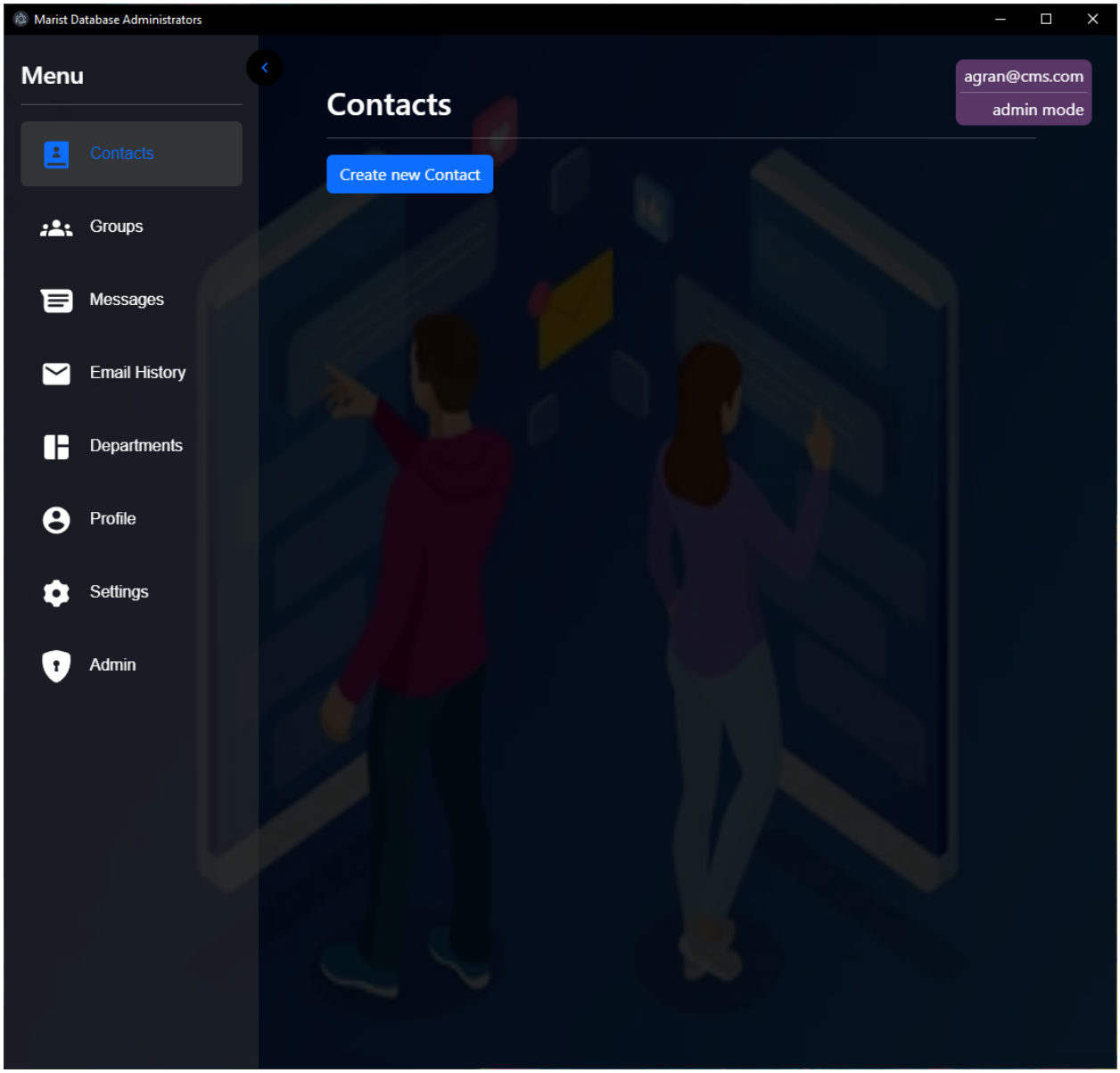


Figure 38: Main Menu Page - User Contacts View

Below is the relative code for the user Contacts view.

```
import { Link } from "react-router-dom";
import { useState, useEffect } from 'react';
import $ from 'jquery';
import contactImg from "../images/contact.png";
import "../css/Contacts.css";

const Contacts = (props) => {
```

```
const [cont, setCont] = useState([]);
let contacts = [];
let employees, empID;
let contList = [];

useEffect(() => {
    window.dbConnection.getContacts().then((result) => {
      contacts = result;

      window.dbConnection.getEmployees().then((result) => {
        employees = result;

        employees.forEach((emp, i) => {
          if (emp.email == $("#userProfile .userName").text()) {
            empID = emp;
          }
        });

        contacts.forEach((cont, i) => {
          employees.forEach((emp, i) => {
            if (cont.user2ID == emp.ID && cont.user1ID == empID.ID) {
              contList.push(emp);
            }
          });
        });
        setCont([...contList]);
      });
    });
}, []);

const handleDelete = (el) => {
  window.dbConnection.deleteContact({
    user2ID: el
  }).then((result) => {
    console.log(result);
  });

  cont.forEach((item, i) => {
    let tmp = cont;
    tmp = tmp.filter(tmp => tmp.ID != el);
```

```
      setCont(tmp);
    });
  }

  return (
    <>
      <div id="contacts" className="container">
        <img src={contactImg} className="backgroundImg"></img>
        <h2>Contacts</h2>
        <hr/>
        {cont.map((el, i) => {
          return (
            <>
            <div key={el.ID} id={el.ID} className="contactWrapper row">
              <div className="contact col">
                <div className="row">
                  <div className="col-3">Name<span
className="float-end">:</span></div>
                  <div className="col-9">{el.Fname + " " + el.Lname}</div>
                </div>
                <hr/>
                <div className="row">
                  <div className="col-3">Email<span className="float-end">:</span></div>
                  <div className="col-9">{el.email}</div>
                </div>
                <hr/>
                <div className="row">
                  <div className="col-3">Phone Number<span
className="float-end">:</span></div>
                  <div className="col-9">{el.phoneNum}</div>
                </div>
                <hr/>
                <div className="row">
                  <div className="col-3">Work Number<span
className="float-end">:</span></div>
                  <div className="col-9">{el.WorkNum}</div>
                </div>
                <hr/>
                <div className="row">
```

```
              <div className="col-3">Gender<span
className="float-end">:</span></div>
              <div className="col-9">{el.gender}</div>
            </div>
            <hr/>
            <div className="row mb-0">
              <div className="col-3">Age<span className="float-end">:</span></div>
              <div className="col-9">{el.age}</div>
            </div>
            <hr/>
            <div className="float-end">
              <button className="btn btn-danger" onClick={() =>
handleDelete(el.ID)}>Delete</button>
            </div>
          </div>
        </div>
        <hr/>
        </>
      )
    })}
    <Link to="/createContact">
      <button className="btn btn-primary">Create new Contact</button>
    </Link>
  </div>
 </>
 );
};

export default Contacts;
```

Contacts Page:

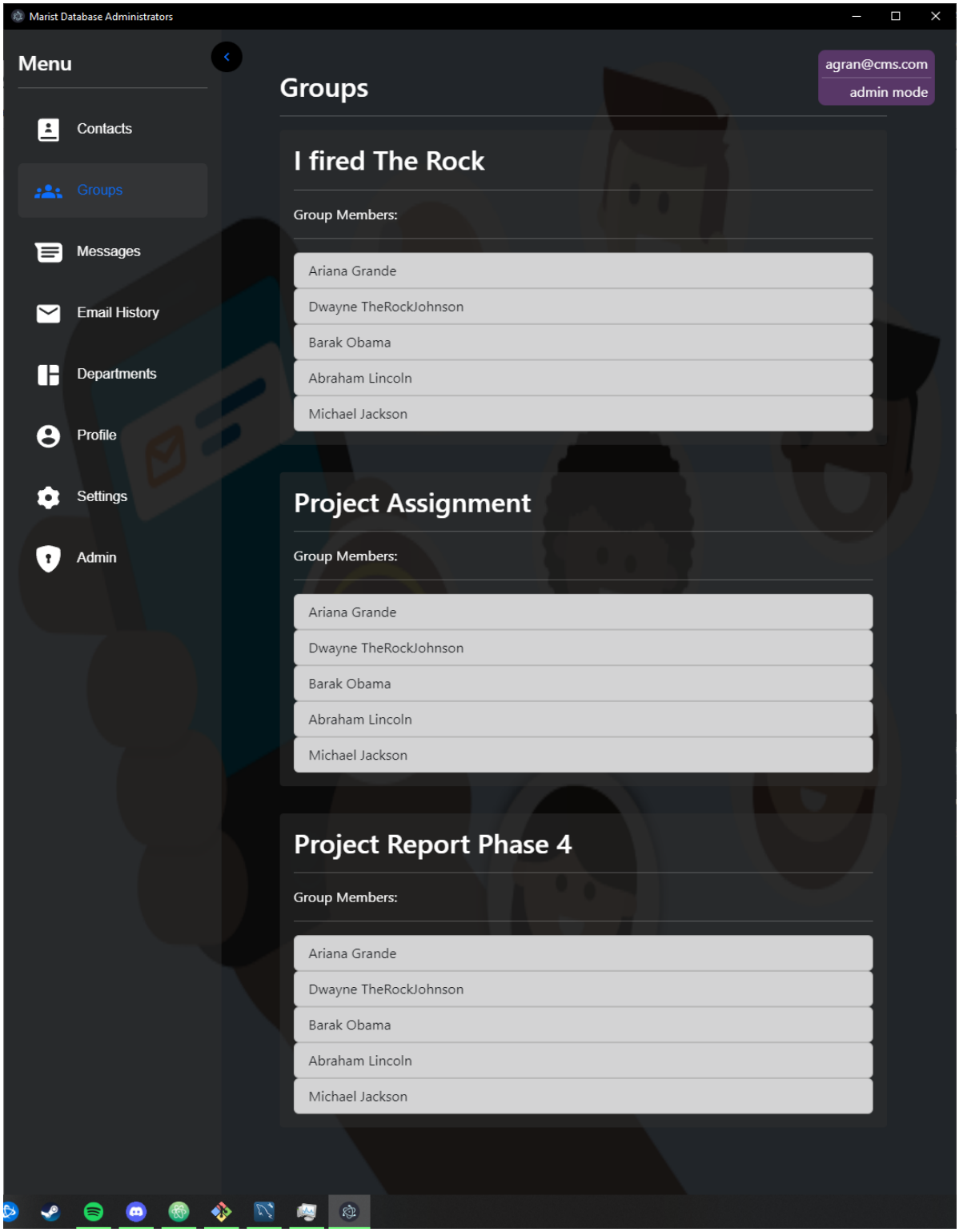


Figure 39: Main Menu Page - User Groups View

Below is the relative code for the user Groups view.

```
import { useEffect, useState } from 'react';
import $ from 'jquery';
import "../css/Groups.css";
import backImg from "../images/groups.png";
const utils = require('../utils/utils.js');

const Groups = (props) => {
  const [userGroups, setUserGroups] = useState([]);
  const [groupData, setGroupData] = useState([]);

  let groupMems;
  let employeeID;
  let employees = [];
  let groupIDs = [];
  let groupDetails = [];
  let tmpUserGroups = [];

  useEffect(() => {
    window.dbConnection.getGroupMembers().then((result) => {
      groupMems = result;
      window.dbConnection.getEmployees().then((result) => {
        employees = result;
        employees.forEach((item, i) => {
          if (item.email === $("#userProfile p.userName").text()) {
            employeeID = item.ID;
          }
        });

        groupMems.forEach((groupMem, i) => {
          if (groupMem.UserID === employeeID) {
            let tmp = [];
            let tmpID = groupMem.GroupID;

            groupMems.forEach((item, i) => {
              if (tmpID === item.GroupID) {
                tmp.push(item.UserID);
              }
            });
            groupIDs.push({groupID: groupMem.ID, empIDs: tmp});
```

```
        }
      });

      window.dbConnection.getGroupDetails().then((result) => {
        groupDetails = result;
        let tmpEmp = [];
        let tmpGroup = [];
        groupIDs.forEach((group, i) => {
          group.empIDs.forEach((empID, i) => {
            employees.forEach((emp, i) => {
              if (empID === emp.ID) {
                tmpEmp.push(emp.Fname + " " + emp.Lname);
              }
            });
          });

          groupDetails.forEach((groupDet, i) => {
            if (group.groupID === groupDet.GroupMembers_ID) {
              tmpGroup.push(groupDet);
            }
          });

          setUserGroups(tmpGroup);
          setGroupData(tmpEmp);
        });
      });
    });
  });
}, []);

const openBar = (el) => {
  $('#' + el + ' .collapsible').toggleClass('open');
}

return (
  <>
    <div id="contacts" className="container">
      <img src={backImg} className="backgroundImg"></img>
      <h2>Groups</h2>
      <hr/>
```

```
      <div id="groupContainer">
        {userGroups.map((item, i) => {
          return (
            <>
              <div id={item.ID} className="group" onClick={() =>  openBar(item.ID)}>
                <h2>{item.GroupName}</h2>
                <hr/>
                <div className="collapsible open">
                <p>Group Members:</p>
                <hr/>
                {groupData.map((group, i) => {
                  return (
                    <>
                      <ul className="list-group">
                        <li className="list-group-item  list-group-item-dark">{group}</li>
                      </ul>
                    </>
                  )
                })}
                </div>
              </div>
            </>
          )
        })}
      </div>
    </div>
  </>
 );
};

export default Groups;
```

Messages Page:

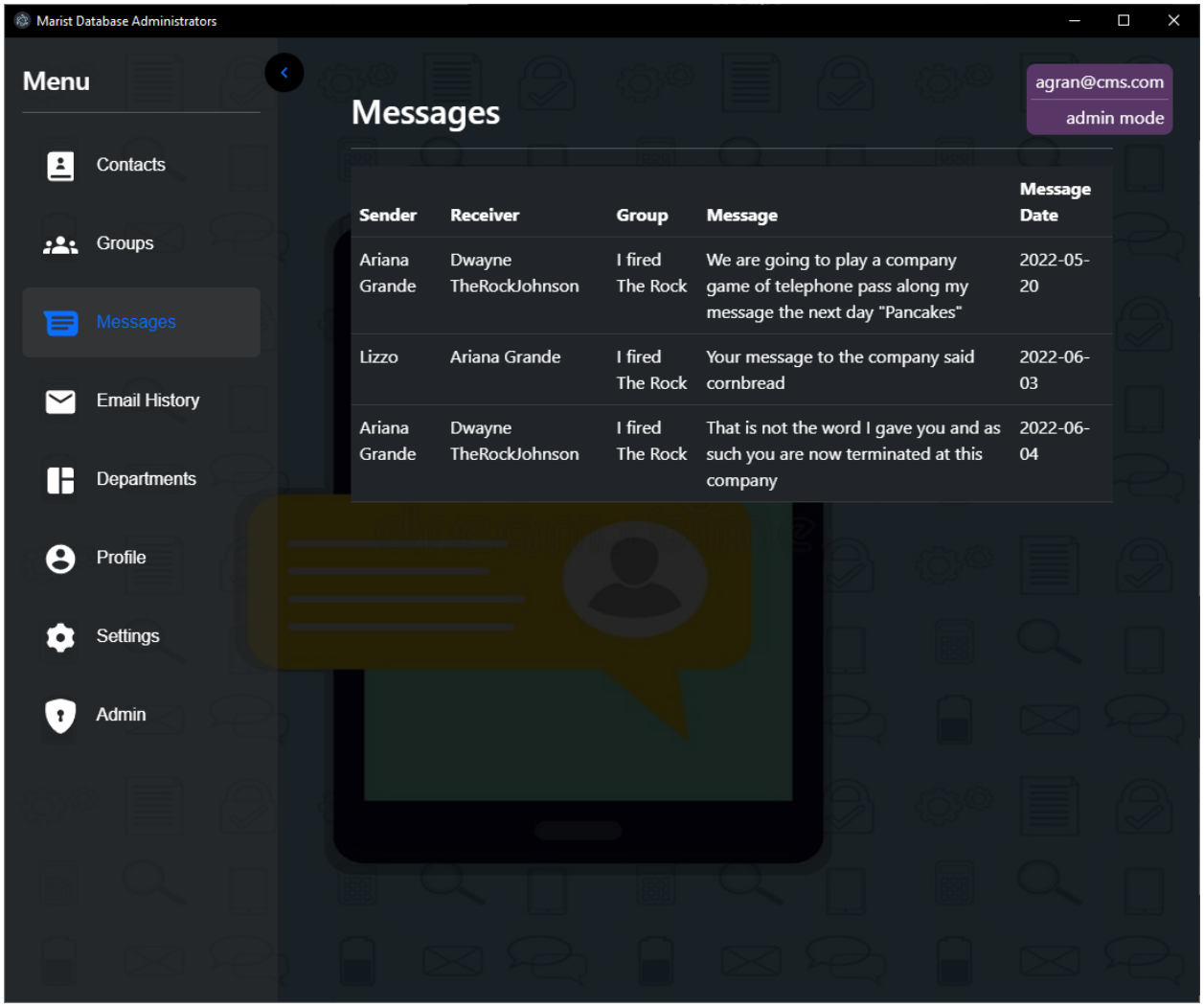


Figure 40: Main Menu Page - User Messages View

Below is the relative code for the main menu Messages view.

```
import { useState, useEffect } from 'react';
import $ from 'jquery';
import "../css/Messages.css";
import backImg from "../images/messages.png";
const utils = require('../utils/utils.js');

const Messages = (props) => {
  const [messages, setMessages] = useState([]);
  const [firstRun, setFirstRun] = useState(true);
  let groupDet;
```

```
useEffect(() => {
  if (firstRun) {
    window.dbConnection.getMessages().then((result) => {
      setMessages([...result]);
    });
    window.dbConnection.getEmployees().then((result) => {
      let employeeData = result;

      window.dbConnection.getGroupDetails().then((result) => {
        groupDet = result;

      }).then((result) => {
        window.dbConnection.getEmployeeIDs({
          email: $("#userProfile .userName").text()
        }).then((result) => {
          $(document).ready(() => {
            $("#messagesBody tr").each((i, el) => {
              let sender = $(el).find(".sender");
              let receiver = $(el).find(".receiver");
              let groupName = $(el).find(".groupName");
              if (sender.text() == result[0].ID) {
                sender.text(result[0].Fname + " " + result[0].Lname);
                employeeData.forEach((item, i) => {
                  for (let i = 0; i < Object.keys(item).length; i++) {
                    let key = Object.keys(item)[i];
                    if (item[key] === null) item[key] = "";
                  }
                  if (receiver.text() == item.ID) receiver.text(item.Fname + " " + item.Lname);
                });
              } else if (receiver.text() == result[0].ID) {
                receiver.text(result[0].Fname + " " + result[0].Lname);
                employeeData.forEach((item, i) => {
                  for (let i = 0; i < Object.keys(item).length; i++) {
                    let key = Object.keys(item)[i];
                    if (item[key] === null) item[key] = "";
                  }
                  if (sender.text() == item.ID) sender.text(item.Fname + " " + item.Lname);
                });
              } else {
```

```
            $(el).remove();
          }
          groupDet.forEach((item, i) => {
            if (groupName.text() == item.ID) {
              groupName.text(item.GroupName);
            }
          });

          // groupName.text()
        });
        $("#messagesBody").show();
      });
    });
  });
  setFirstRun(false);
 }
}, []);

return (
 <>
  <div id="contacts" className="container">
  <img src={backImg} className="backgroundImg"></img>
   <h2>Messages</h2>
   <hr/>
   <table className="table table-dark table-hover">
    <thead>
     <tr>
      <th scope="col">Sender</th>
      <th scope="col">Receiver</th>
      <th scope="col">Group</th>
      <th scope="col">Message</th>
      <th scope="col">Message Date</th>
     </tr>
    </thead>
    <tbody id="messagesBody">
     {messages.map((item, i) => {
      return (
       <>
        <tr>
```

```
                <td className="sender">{item.senderID}</td>
                <td className="receiver">{item.userID}</td>
                <td className="groupName">{item.groupID}</td>
                <td>{item.Message}</td>
                <td>{utils.formatDate(item.Messagedate)}</td>
              </tr>
            </>
          )
        })}
      </tbody>
    </table>
  </div>
 </>
 );
}

export default Messages;
```

Email History Page:

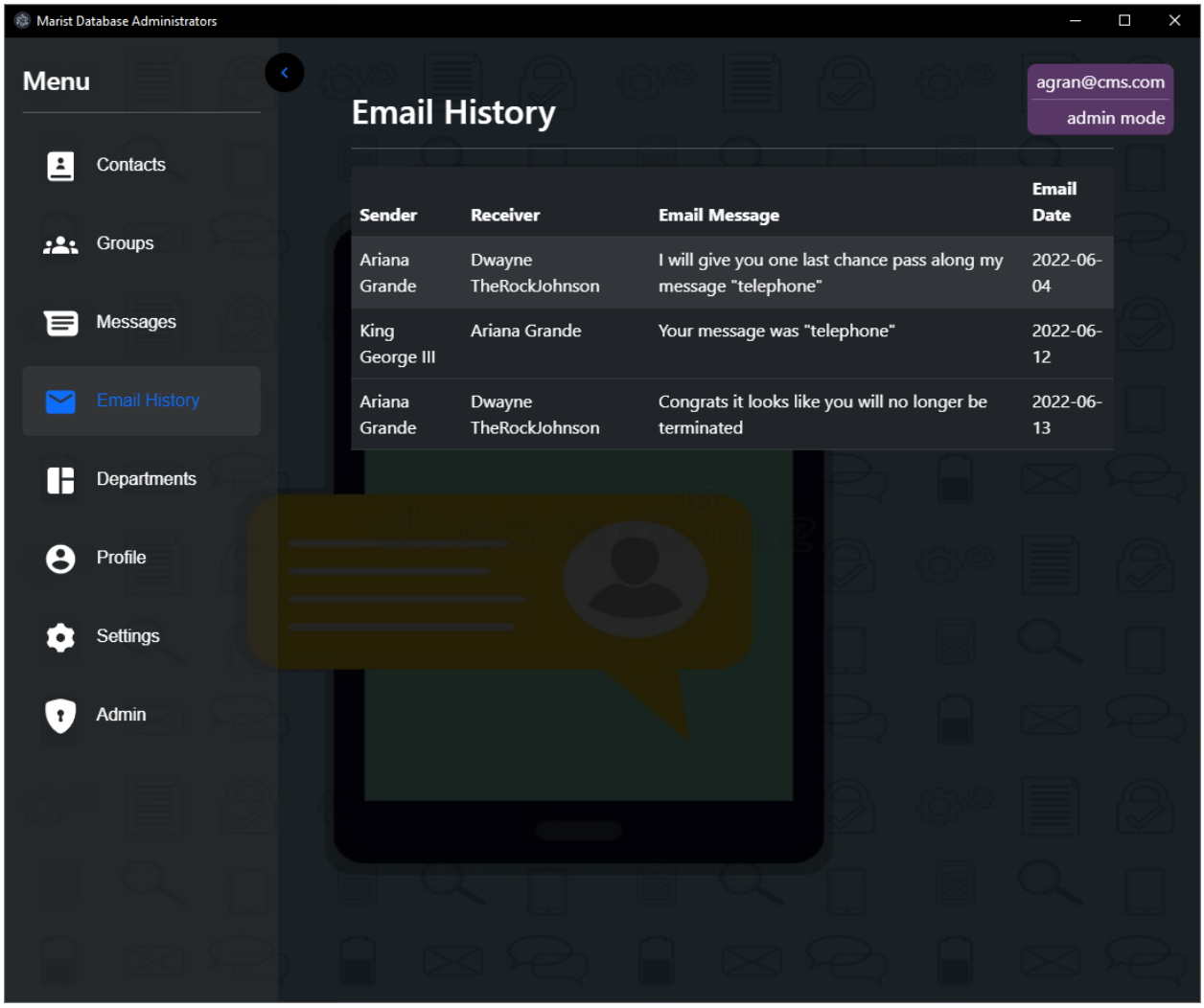


Figure 41: Main Menu Page - User Email History View

Below is the relative code for the main menu Email History view.

```
import { useState, useEffect } from 'react';
import $ from 'jquery';
import "../css/EmailHistory.css";
import backImg from "../images/messages.png";
const utils = require('../utils/utils.js');

const EmailHistory = (props) => {
  const [emails, setEmails] = useState([]);
  const [firstRun, setFirstRun] = useState(true);
```

```
useEffect(() => {
  if (firstRun) {
    window.dbConnection.getEmailHistory().then((result) => {
      setEmails([...result]);
    });
    window.dbConnection.getEmployees().then((result) => {
      let employeeData = result;
      window.dbConnection.getEmployeeIDs({
        email: $("#userProfile .userName").text()
      }).then((result) => {
        $(document).ready(() => {
          $("#EmailHistoryBody tr").each((i, el) => {
            let sender = $(el).find(".sender");
            let receiver = $(el).find(".receiver");
            if (sender.text() == result[0].ID) {
              sender.text(result[0].Fname + " " + result[0].Lname);
              employeeData.forEach((item, i) => {
                for (let i = 0; i < Object.keys(item).length; i++) {
                  let key = Object.keys(item)[i];
                  if (item[key] === null) item[key] = "";
                }
                if (receiver.text() == item.ID) receiver.text(item.Fname + " " + item.Lname);
              });
            } else if (receiver.text() == result[0].ID) {
              receiver.text(result[0].Fname + " " + result[0].Lname);
              employeeData.forEach((item, i) => {
                for (let i = 0; i < Object.keys(item).length; i++) {
                  let key = Object.keys(item)[i];
                  if (item[key] === null) item[key] = "";
                }
                if (sender.text() == item.ID) sender.text(item.Fname + " " + item.Lname);
              });
            } else {
              $(el).remove();
            }
            $("#EmailHistoryBody").show();
          });
        });
      });
    });
```

```
      setFirstRun(false);
    }
  }, []);

  return (
    <>
      <div id="contacts" className="container">
        <img src={backImg} className="backgroundImg"></img>
        <h2>Email History</h2>
        <hr/>
        <table className="table table-dark table-hover">
          <thead>
            <tr>
              <th scope="col">Sender</th>
              <th scope="col">Receiver</th>
              <th scope="col">Email Message</th>
              <th scope="col">Email Date</th>
            </tr>
          </thead>
          <tbody id="EmailHistoryBody">
            {emails.map((item, i) => {
              return (
                <>
                  <tr>
                    <td className="sender">{item.SenderID}</td>
                    <td className="receiver">{item.ReceiverID}</td>
                    <td>{item.Message}</td>
                    <td>{utils.formatDate(item.EmailDate)}</td>
                  </tr>
                </>
              )
            })}
          </tbody>
        </table>
      </div>
    </>
  );
};

export default EmailHistory;
```

Departments Page:

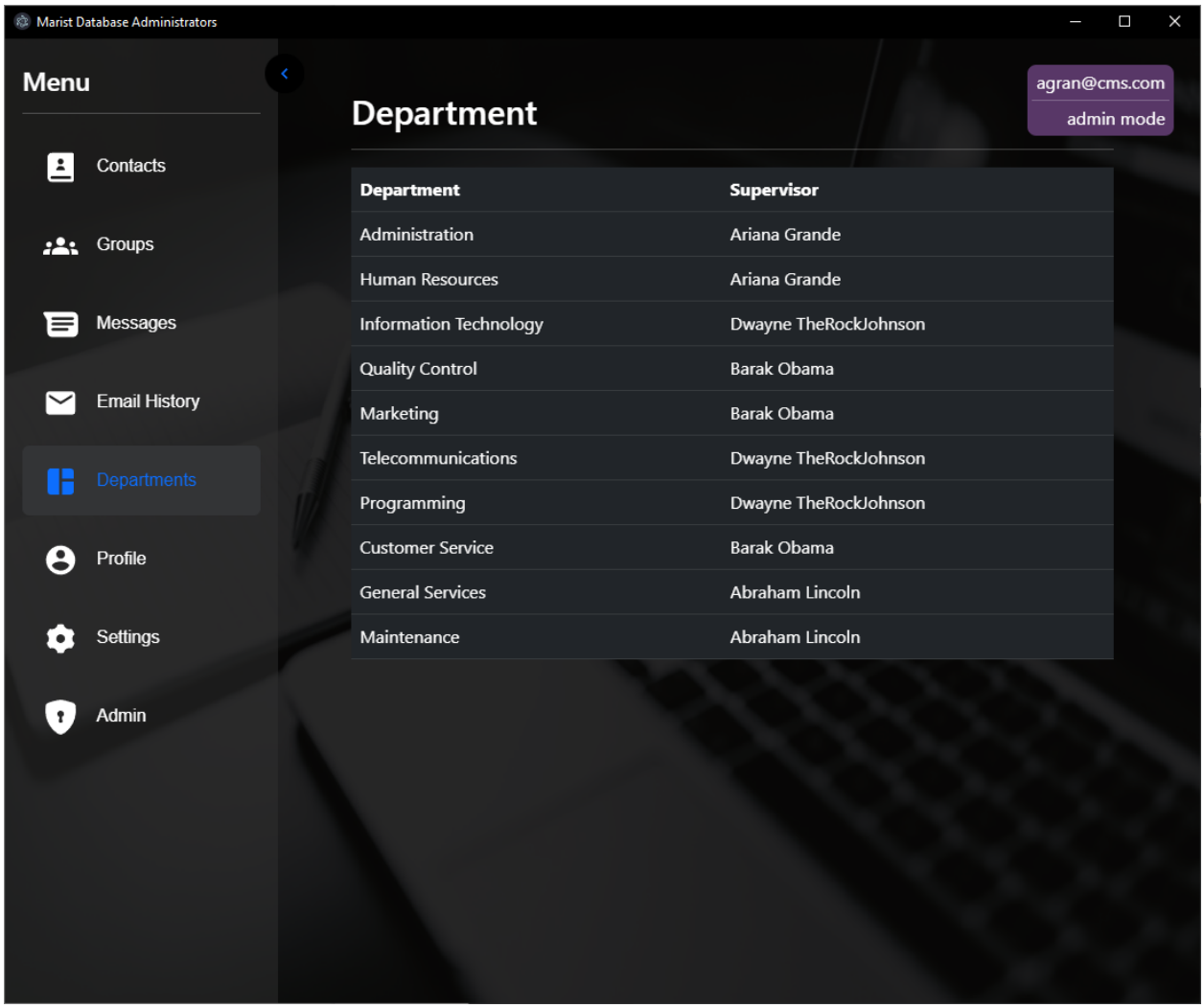


Figure 42: Main Menu Page - User Department View

Below is the relative code for the main menu Department view.

```
import { useState, useEffect } from 'react';
import $ from 'jquery';
import "../css/Department.css";
import backImg from "../images/departments.jpg";
const utils = require('../utils/utils.js');

const Department = (props) => {
  const [dprt, setDprt] = useState([]);
  const [sup, setSup] = useState([]);
  const [emp, setEmp] = useState([]);
  let supervisors;
```

```
let employees;

useEffect(() => {
  window.dbConnection.getDepartments().then((result) => {
    setDprt([...result]);

    window.dbConnection.getSupervisors().then((result) => {
      setSup([...result]);
      supervisors = result;

      window.dbConnection.getEmployees().then((result) => {
        setEmp([...result]);
        employees = result;

        $(document).ready(function() {
          $("#departmentBody tr .supervisorID").each(function(i, el) {
            supervisors.forEach((supervisor, i) => {
              if (Number($(el).text()) === supervisor.ID) {
                employees.forEach((res, j) => {
                  if (supervisor.UserID === res.ID) {
                    $(this).text(res.Fname + " " + res.Lname);
                  }
                });
              }
            });
          });
        });
      });
    });
  });
}, []);

return (
  <>
    <div id="department" className="container">
      <img src={backImg} className="backgroundImg"></img>
      <h2>Department</h2>
      <hr/>
      <table className="table table-dark table-hover">
        <thead>
```

```
            <tr>
              <th scope="col">Department</th>
              <th scope="col">Supervisor</th>
            </tr>
          </thead>
          <tbody id="departmentBody">
            {dprt.map((item, i) => {
              return (
                <>
                  <tr>
                    <td >{item.DName}</td>
                    <td className="supervisorID">{item.Supervisor_ID}</td>
                  </tr>
                </>
              )
            })}
          </tbody>
        </table>
      </div>
    </>
  );
};

export default Department;
```

Profile Page:

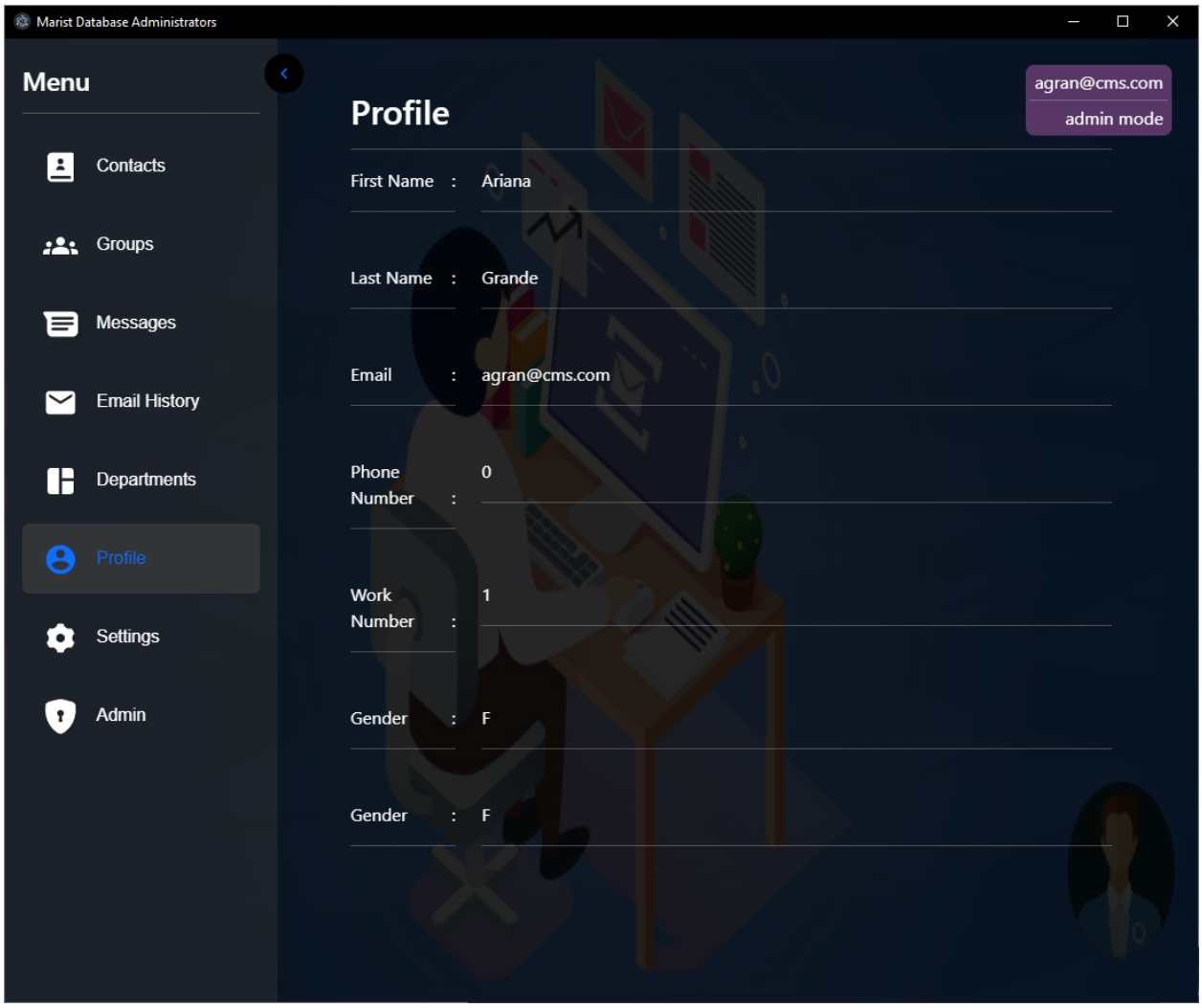


Figure 43: Main Menu Page - User Profile View

Below is the relative code for the main menu Profile view.

```
import { useEffect, useState } from 'react';
import $ from 'jquery';
import "../css/Profile.css";
import profileImg from "../images/profile.png";
const utils = require('../utils/utils.js');

const Profile = (props) => {
  const [employees, setEmployees] = useState([]);
  const [data, setData] = useState([]);
```

```
useEffect(() => {

  $(document).ready(() => {
    console.log($("#root"));
    $("#root").css({"background-image": "url(../images/profile.png)"});
  });
  window.dbConnection.getEmployeeIDs({
    email: $("#userProfile .userName").text()
  }).then((result) => {
    let user = result[0];
    Object.keys(user).forEach((item, i) => {
      if (user[item] === null) {
        user[item] = "";
      }
    });

    setEmployees([...[user]]);
  });
}, []);

const setProfileData = (res) => {
  let keys = Object.keys(res)
  let tmp = {}
  keys.forEach((item, i) => {
    console.log(item);
    console.log(res[item]);
    let tmpNull = res[item] == null ? "" : res[item];
    console.log(tmpNull);
    console.log(String(tmpNull).length);
    tmp[item] = String(tmpNull).length === 0 ? "null" : res[item];
  });
}

return (
  <>
    <div id="Profile" className="container">
    <img src={profileImg} className="backgroundImg"></img>
      <h2>Profile</h2>
      <hr/>
      {employees.map((item, i) => {
```

```
return (
  <>
    <div className="row">
      <div className="col-2">
        <div>First Name<span className="float-end">:</span></div>
        <hr/>
      </div>
      <div className="col-10">
        <div>{item.Fname}</div>
        <hr/>
      </div>
    </div>
    <div className="row">
      <div className="col-2">
        <div>Last Name<span className="float-end">:</span></div>
        <hr/>
      </div>
      <div className="col-10">
        <div>{item.Lname}</div>
        <hr/>
      </div>
    </div>
    <div className="row">
      <div className="col-2">
        <div>Email<span className="float-end">:</span></div>
        <hr/>
      </div>
      <div className="col-10">
        <div>{item.email}</div>
        <hr/>
      </div>
    </div>
    <div className="row">
      <div className="col-2">
        <div>Phone Number<span className="float-end">:</span></div>
        <hr/>
      </div>
      <div className="col-10">
        <div>{item.phoneNum}</div>
        <hr/>
```

```
            </div>
          </div>
          <div className="row">
            <div className="col-2">
              <div>Work Number<span className="float-end">:</span></div>
              <hr/>
            </div>
            <div className="col-10">
              <div>{item.WorkNum}</div>
              <hr/>
            </div>
          </div>
          <div className="row">
            <div className="col-2">
              <div>Gender<span className="float-end">:</span></div>
              <hr/>
            </div>
            <div className="col-10">
              <div>{item.gender}</div>
              <hr/>
            </div>
          </div>
          <div className="row">
            <div className="col-2">
              <div>Gender<span className="float-end">:</span></div>
              <hr/>
            </div>
            <div className="col-10">
              <div>{item.gender}</div>
              <hr/>
            </div>
          </div>
        </>
      )
    })}
  </div>
 </>
 );
};
```

```
export default Profile;
```

Settings Page:

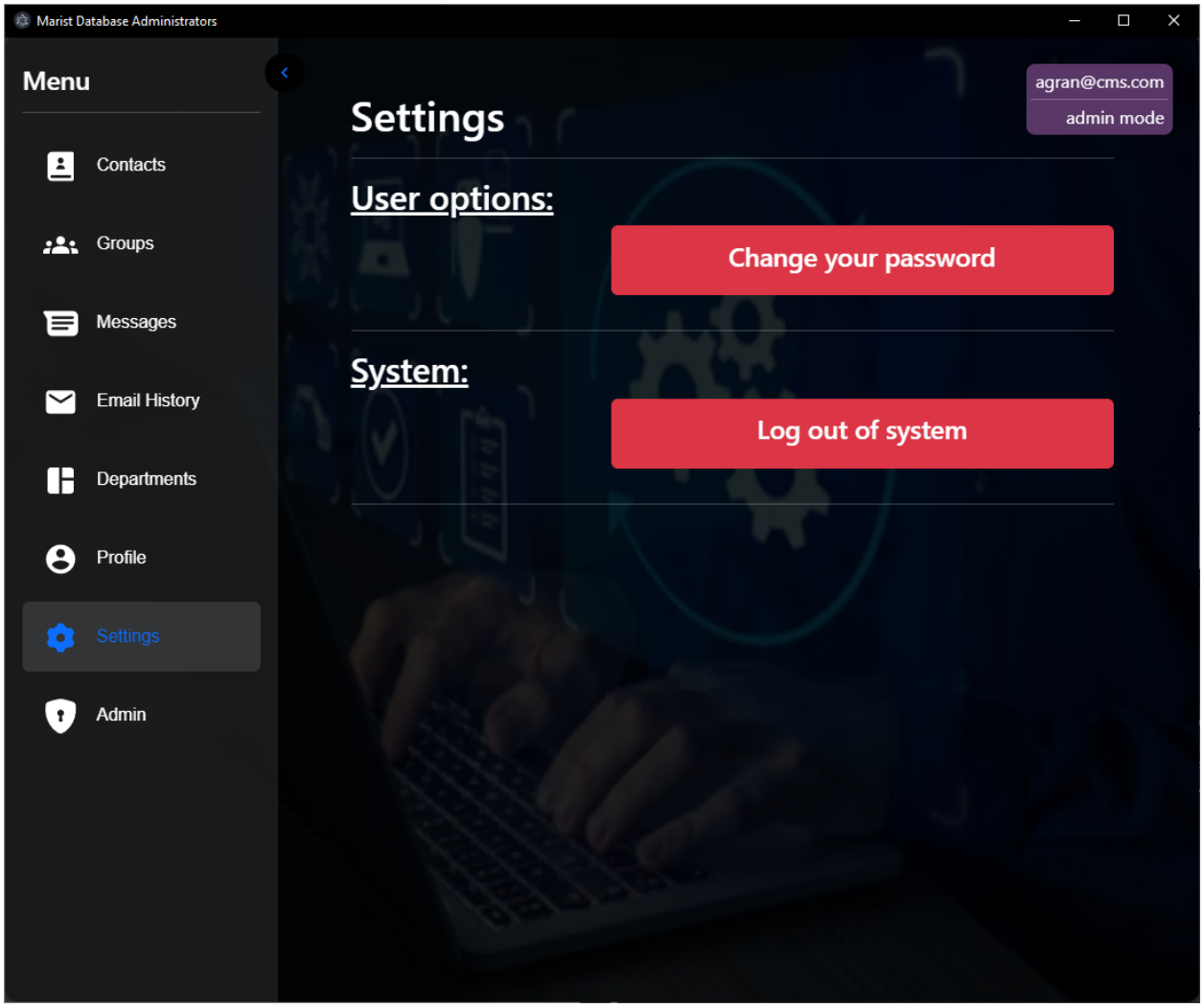


Figure 44: Main Menu Page - User Settings View

Below is the relative code for the main menu Settings view.

```
import { Link } from "react-router-dom";
import backImg from "../images/settings.png";
import "../css/Settings.css";

const Settings = (props) => {
  function handleLogout(e) {
    e.preventDefault();
    window.history.replaceState(null, null, "/");
    window.location.reload();
```

```
  }

  return (
    <>
      <div id="settings" className="container">
        <img src={backImg} className="backgroundImg"></img>
        <h1>Settings</h1>
        <hr/>
        <h2 className="sectionHeader">User options:</h2>
        <div className="container">
          <div className="row">
            <button className="btn btn-danger adminBtn">
              <Link to="/changePassword">
                <h4>Change your password</h4>
              </Link>
            </button>
          </div>
        </div>
        <hr/>
        <h2 className="sectionHeader">System:</h2>
        <div className="container">
          <div className="row">
            <div>
              <button className="btn btn-danger adminBtn" onClick={handleLogout}>
                <h4>Log out of system</h4>
              </button>
            </div>
          </div>
        </div>
        <hr/>
      </div>
    </>
  );
};

export default Settings;
```

Admin Main Menu Page:

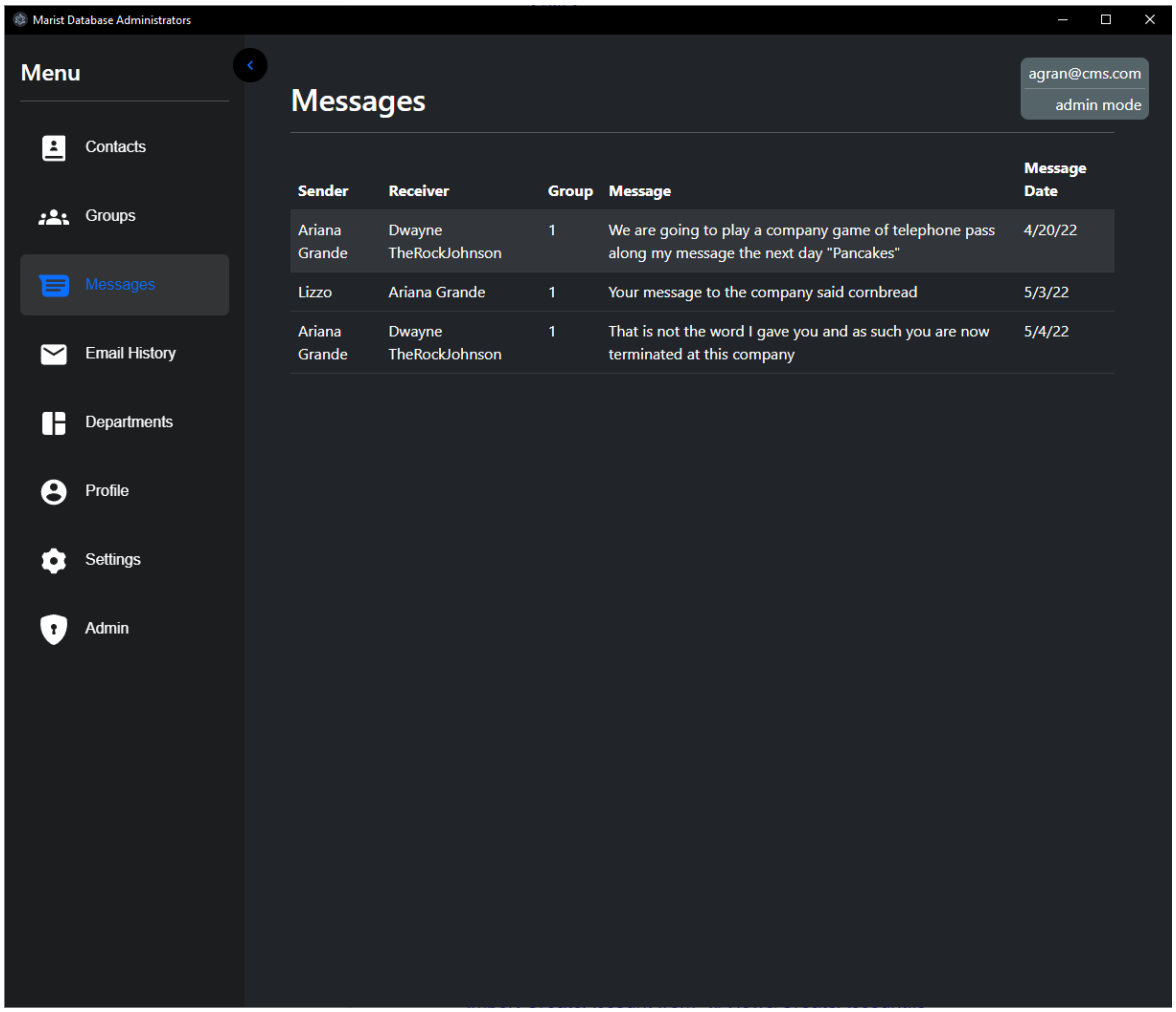


Figure 45: Main Menu Page - Admin View

Below is the relative code for the main menu view.

```
import SideNav from "../Components/SideNav";
import Login from "./Login";
import EditEmployeeTable from "./EditEmployeeTable";
import Test from "./Test";
import EditUserTable from "./EditUserTable";
import EditMessagesTable from "./EditMessagesTable";
import ChangePassword from "./ChangePassword";
import Department from "./Department";
import Messages from "./Messages";
import Employees from "./Employees";
import Users from "./Users";
```

```
import EmailHistory from "./EmailHistory";
import Groups from "./Groups";
import Contacts from "./Contacts";
import Profile from "./Profile";
import Settings from "./Settings";
import CreateAccount from "../Views/CreateAccount.js"
import Admin from "../Views/Admin.js"
import { Routes, Route, Navigate } from 'react-router-dom';
import { useState, useEffect } from 'react';
import '../css/Main.css';
import $ from 'jquery';

const Main = (props) => {
  const [newClass, setClass] = useState("");
  const [firstRun, setFirstRun] = useState(true);

  useEffect(() => {
    if (firstRun) {
      // max hex color value is (16*16) * (16*16) * (16*16) - 1 = 16777215
      let newHexVal = [ String(Math.floor(Math.random() * 63) + 16).toString(16),
String(Math.floor(Math.random() * 63) + 16).toString(16),
String(Math.floor(Math.random() * 63) + 16).toString(16) ];
      let finalNum = "#";
      newHexVal.forEach((item, i) => {
        finalNum += item;
      });

      $("body").append("<style>.rngColor{background:" + finalNum +
"!important;}</style>");
      setFirstRun(false);
    }
  }, []);

  return (
    <>
      <div id="main">
        <nav className={`sidenav p-3 py-4 ${newClass}`}>
          <SideNav setActive={setClass} admin={props.admin}/>
        </nav>
        <div id="userProfile" className="rngColor">
```

```
        <p className="userName">{props.name}</p>
        { props.admin ? <AdminView /> : null }
      </div>

      <div id="routes" className="p-4 p-md-5 pt-5">
        <Routes>
          <Route path="/" exact element={<Contacts />} />
          <Route path="/contacts" element={<Contacts/>}/>
          <Route path="/groups" element={<Groups/>}/>
          <Route path="/messages" element={<Messages/>}/>
          <Route path="/emailHistory" element={<EmailHistory/>}/>
          <Route path="/department" element={<Department/>}/>
          <Route path="/profile" element={<Profile/>}/>
          <Route path="/settings" element={<Settings/>}/>
          <Route path="/admin" element={<Admin admin={props.admin}/>}/>
          <Route path="/login" exact element={<Login />} />
          <Route path="/createAccount" element={<CreateAccount/>}/>
          <Route path="/editEmployeeTable" element={<EditEmployeeTable/>}/>
          <Route path="/editUserTable" element={<EditUserTable/>}/>
          <Route path="/editMessagesTable" element={<EditMessagesTable/>}/>
          <Route path="/changePassword" element={<ChangePassword/>}/>
          <Route path="/test" element={<Test/>}/>
          <Route render={
            ()=>(<Navigate to="/login"/>)}
          />
        </Routes>
      </div>
    </div>
  </>
 );
};

const AdminView = (props) => {
 return (
  <>
    <hr id="adminSeperator"/>
    <p className="admin">admin mode</p>
  </>
 );
};
```

```
export default Main;
```

Action Pages:

The image included below is a screenshot of the "Create Account" action page. Here, an admin can create a new user entry in the database. This will give new users a framework for creating and managing new contacts in the database.

Create Account Page:

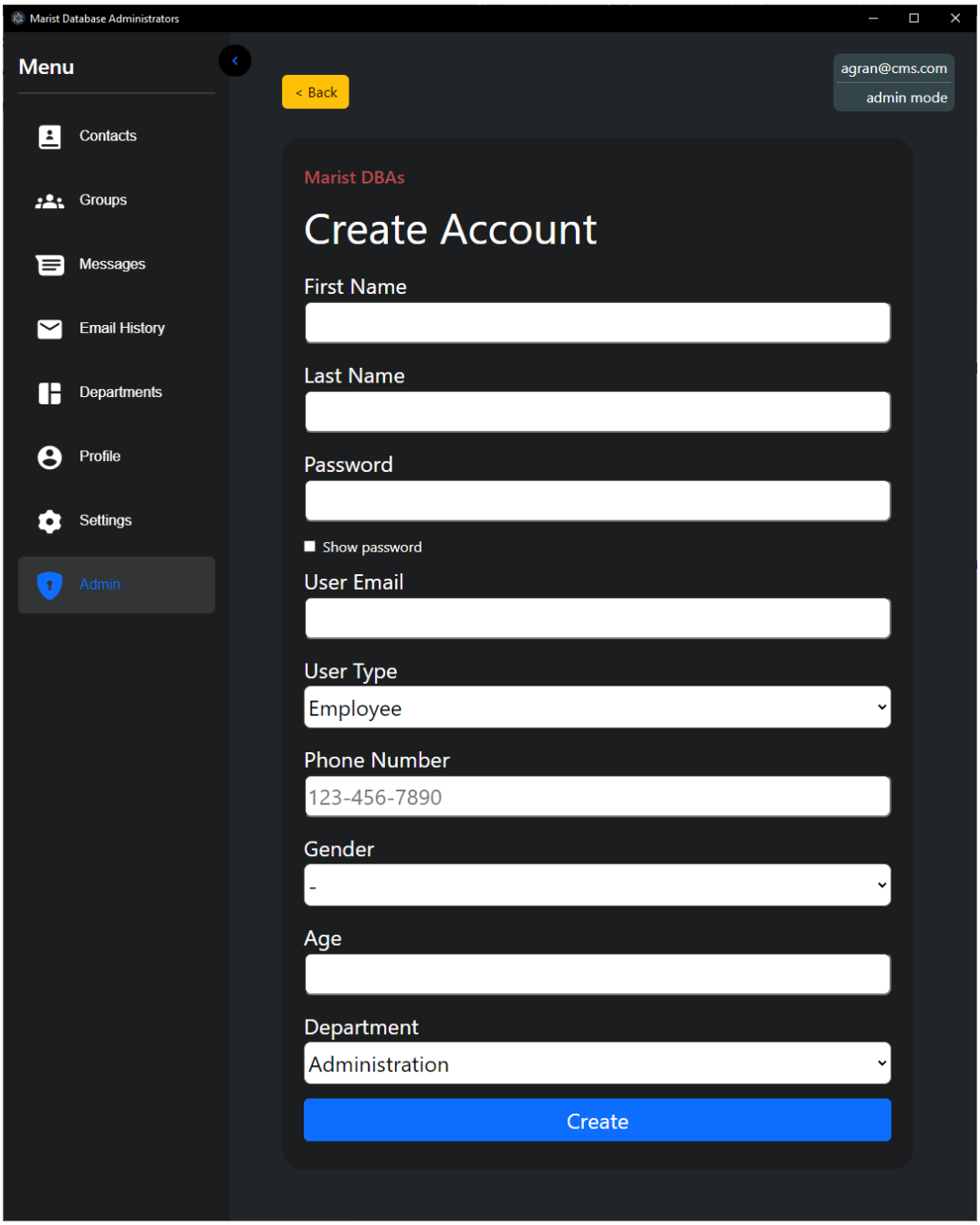


Figure 46: Create Account Action Page

Below is the relative code for the "Create Account" action page.

```
import { Link } from "react-router-dom";
import GenderData from '../Components/GenderData';
import GenderOption from '../Components/GenderOption';
import DepartmentData from '../Components/DepartmentData';
import DepartmentOption from '../Components/DepartmentOption';
import { useState, useEffect } from 'react';
import $ from 'jquery';
import "../css/CreateAccount.css";
import "../css/Login.css";

const CreateAccount = (props) => {
  const [firstRun, setFirstRun] = useState(true);
  const [genData, setGenValue] = useState("-");
  const [depData, setDepData] = useState(DepartmentData);
  const [depValue, setDepValue] = useState(null);
  const [supValue, setSupValue] = useState(null);
  const [userType, setUserType] = useState("employee");

  useEffect(() => {
    if (firstRun) {
      window.dbConnection.getDepartments().then((result) => {
        let newData = [];
        result.forEach((item, i) => {

          newData.push({
            id: item.ID,
            label: item.DName,
            value: item.Supervisor_ID
          });
        });
        setDepData([...newData]);
      });

      setFirstRun(false);
    }
  });
```

```
function handleDepChange(e) {
  // e.preventDefault();
}

async function handleSubmit(e) {
  e.preventDefault();
  window.dbConnection.createAccount({
    Fname: $("#firstName").val(),
    Lname: $("#lastName").val(),
    password: $("#password").val(),
    email: $("#userEmail").val(),
    userType: $("#userType").val(),
    phoneNum: $("#phoneNum").val(),
    gender: $("#gender").val(),
    age: $("#age").val(),
    Department_ID: $("#department").val(),
    Supervisor_ID: $("#department option[value='" + $("#department").val() + "']")[0].id
  }).then((result) => {
    console.log(result);
    if (typeof result === 'string') {
      if (result.slice(0,5) == "Error") {
        $("#createAccount .errorMessage").text(result);
        $("#createAccount .errorMessage").addClass("active");
        setTimeout(() => {
          $("#createAccount .errorMessage").removeClass("active");
        }, 4000);
      }
    } else {
      $("#createAccount .successMessage").addClass("active");
      setTimeout(() => {
        $("#createAccount .successMessage").removeClass("active");
      }, 4000);
    }
  });
}

function togglePassword(e) {
  let inputType = $("#password").attr("type");
  if (inputType === "text") {
    $("#password").attr("type", "password");
```

```
    } else {
      $("#password").attr("type", "text");
    }

    let inputTypeLabel = $("label.password-helper").text();
    if (inputTypeLabel === "Show password") {
      $("label.password-helper").text("Hide password");
    } else {
      $("label.password-helper").text("Show password");
    }
  }

  return (
    <>
      <div className="container">
        <button id="backButton" className="btn btn-warning">
          <Link to="/admin">&lt; Back</Link>
        </button>
      </div>
      <div id="createAccount" className="container">
        <div>
          <h4 id="createAccountTitle" className="text-start">Marist DBAs</h4>
          <div className="header">
            Create Account
          </div>
          <p className="successMessage">Account created successfully</p>
          <p className="errorMessage"></p>
          <form id="createAccountForm" onSubmit={handleSubmit} >
            <div>
              <div>
                <label>First Name</label>
              </div>
              <div>
                <input id="firstName" name="firstName" className="fullWidth mb-3"
type="text" required></input>
              </div>
              <div>
                <label>Last Name</label>
              </div>
              <div>
```

```
            <input id="lastName" name="lastName" className="fullWidth mb-3"
type="text" required></input>
          </div>
          <div>
            <label>Password</label>
          </div>
          <div>
            <input id="password" name="password" className="fullWidth mb-3"
type="password" required></input>
          </div>
          <div>
            <input type="checkbox" onClick={togglePassword}></input>
            <label className="password-helper">Show password</label>
          </div>
          <div>
            <label>User Email</label>
          </div>
          <div>
            <input id="userEmail" name="userEmail" className="fullWidth mb-3"
type="email" required></input>
          </div>
          <div>
            <label>User Type</label>
            <select id="userType" name="userType" className="fullWidth mb-3"
onChange={(e) => setUserType(e.target.value)} required>
              <option value="employee">Employee</option>
              <option value="admin">Admin</option>
            </select>
          </div>
          <div>
            <label>Phone Number</label>
          </div>
          <div>
            <input id="phoneNum" name="phoneNum" className="fullWidth mb-3"
type="tel" placeholder="123-456-7890" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"
required></input>
          </div>
          <div>
            <label>Gender</label>
```

```
            <select id="gender" name="gender" className="fullWidth mb-3"
onChange={(e) => setGenValue(e.target.value)} required>
              <GenderOption genderData={GenderData} />
            </select>
          </div>
          <div>
            <label>Age</label>
          </div>
          <div>
            <input id="age" name="age" className="fullWidth mb-3" type="age"
required></input>
          </div>
          <div>
            <label>Department</label>
            <select id="department" name="department" className="fullWidth mb-3"
onChange={(e) => setDepValue(e.target.value)} required>
              <DepartmentOption departmentData={depData} />
            </select>
          </div>
        </div>
        <div>
          <button className="btn-primary btn" type="submit">Create</button>
        </div>
      </form>
    </div>
  </div>
 </>
 );
};

export default CreateAccount;
```

Admin Delete User Page:

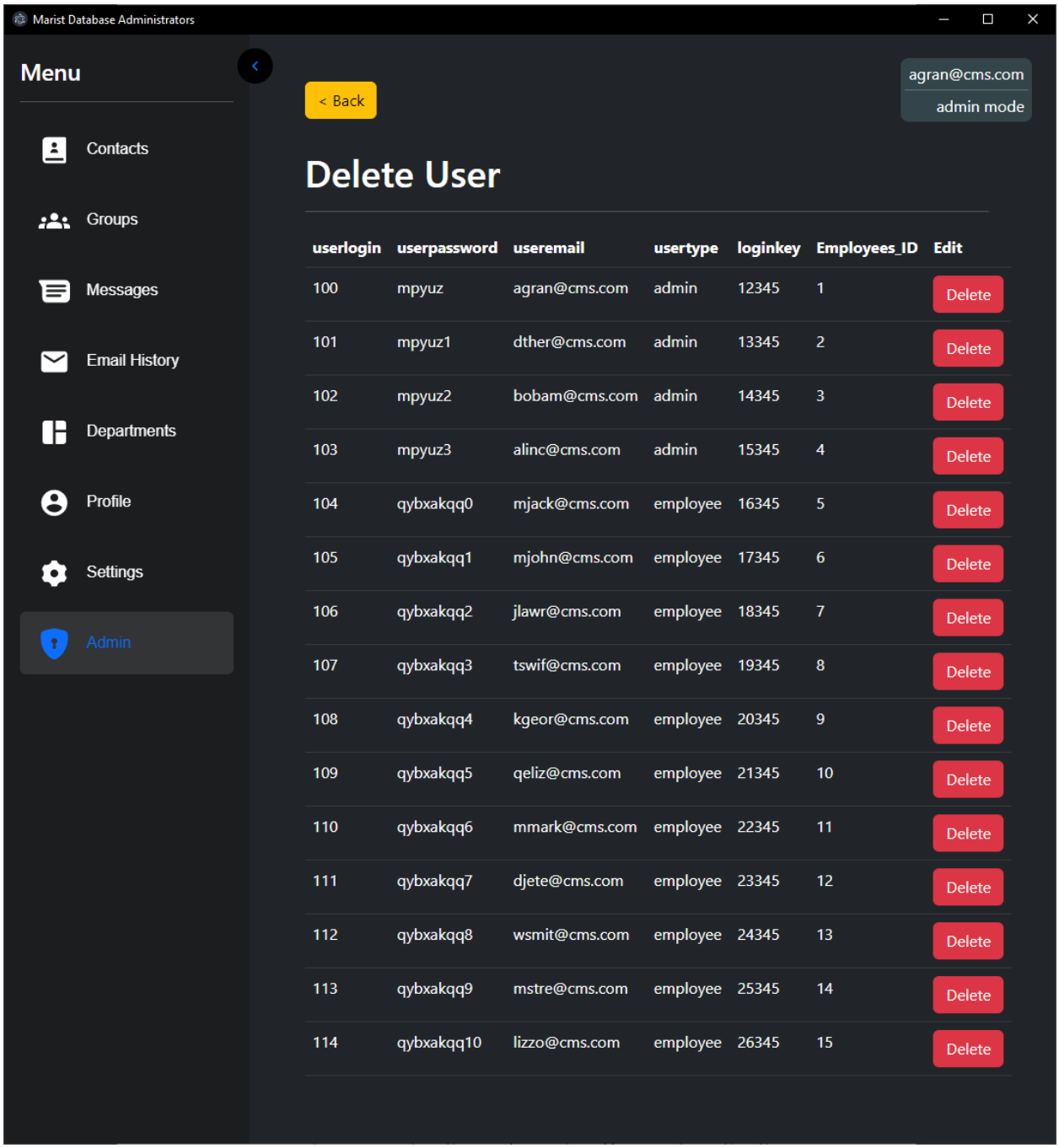


Figure 47: Delete Account Action Page

Below is the view to delete user accounts. Admins can view the table of current users and select which account to delete.

```
import { Link } from "react-router-dom";
```

```
import TableEditor from "./TableEditor";
import { useState, useEffect } from 'react';
import $ from 'jquery';
import "../css/DeleteAccount.css";
import "../css/Login.css";
import "../css/EditUserTable.css";

const DeleteAccount = (props) => {
  const [users, setUsers] = useState([]);
  const [showEdit, setShowEdit] = useState(false);
  const [colDat, setColDat] = useState(null);
  const [headers, setHeaders] = useState([]);

  useEffect(() => {
      window.dbConnection.getUsers().then((result) => {
        setUsers([...result]);
        let keys = Object.keys(result[0]);
        setHeaders([...keys]);
      });
  }, [users]);

  const cancelUserEdit = (e) => {
    $("#userEditContainer").hide();
  }

  const submitEdit = (res) => {
    console.log(res);
  }

  const handleUserEdit = (data) => {
    setColDat(data);
    console.log(colDat);
    setShowEdit(true);
  }

  async function handleSubmit(arg) {
    console.log(arg);
    window.dbConnection.deleteUser({
      email: arg.useremail
    }).then((result) => {
```

```
    console.log(result);
    if (typeof result === 'string') {
      if (result.slice(0,5) == "Error") {
        $("#deleteAccount .errorMessage").text(result);
        $("#deleteAccount .errorMessage").addClass("active");
        setTimeout(() => {
          $("#deleteAccount .errorMessage").removeClass("active");
        }, 4000);
      }
    } else {
      $("#deleteAccount .successMessage").addClass("active");
      setTimeout(() => {
        $("#deleteAccount .successMessage").removeClass("active");
      }, 4000);
    }
  });
}

 return (
   <>
    <div className="container">
      <Link to="/admin">
        <button id="backButton" className="btn btn-warning">&lt; Back</button>
      </Link>
    </div>
    <div id="deleteAccount" className="container">
      <div>
        <h4 id="deleteAccountTitle" className="text-start">Marist DBAs</h4>
        <div className="header">
          Delete Account
        </div>
        <p className="successMessage">Account deleted successfully</p>
        <p className="errorMessage"></p>

        <div id="UserTable">
          <table className="table table-dark table-hover">
            <thead>
              <tr>
                <th scope="col">userlogin</th>
                <th scope="col">userpassword</th>
```

```
                <th scope="col">useremail</th>
                <th scope="col">usertype</th>
                <th scope="col">loginkey</th>
                <th scope="col">Employees_ID</th>
                <th scope="col">Edit</th>
              </tr>
            </thead>
            <tbody>
              {users.map((attr) => {
                return (
                  <>
                    <tr id={attr.userlogin}>
                      <td>{attr.userlogin}</td>
                      <td>{attr.userpassword}</td>
                      <td>{attr.useremail}</td>
                      <td>{attr.usertype}</td>
                      <td>{attr.loginkey}</td>
                      <td>{attr.Employees_ID}</td>
                      <td><button className="btn btn-danger" onClick={() =>
handleSubmit(attr)}>Delete</button></td>
                    </tr>
                  </>
                )
              })}
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </>
 );
};

export default DeleteAccount;
```

Conclusion and Future Work

Throughout this project we have learned a lot regarding Database Management Systems (DBMS) in terms of how to create, manipulate, and provide user interaction with a database. Besides the knowledge gained through learning about DBMS this project also allowed for gaining experience working in a team environment. We were able to break up portions of the report and gave those sections to team members who were best suited for the job. In terms of learning about managing a database we were able to run into issues with our SQL code and work through the solutions so that in the future if we see these errors again we already have a foundational knowledge to work from. This also stressed how to design an interface that the user interacts with to make changes to the database.

In terms of functionality that could be added or enhanced in this project would come a few different forms. For one we can expand on the messaging aspect of this application where messages could be forwarded to the users company email address. Profile pictures could be added by the user or admin for their profile for ease of viewing and to make the user who sent the message much more clear. A more defined messaging system could be set up for creation of messages between users within the company.

References

[1] Fuchs, Jay. "The 14 Best Contact Management Software Tools in 2022." HubSpot Blog, HubSpot, 2 Sept. 2022, https://blog.hubspot.com/sales/contact-management-software.

[2] Keap, 2022, https://keap.com/features/client-management.

[3] Scheiner, Michael. "Keap CRM Review: Keap Software Prices, Features, Pros & Cons." CRM.org, 23 May 2022, https://crm.org/news/keap-crm-review.

[4] Scheiner, Michael. "Streak CRM Review: Pricing, Pros & Cons of Streak for Gmail Chrome Extension." CRM.org, 8 Feb. 2022, https://crm.org/news/streak-crm-review.

[5] Singleton, Chris. "Nimble CRM Review (2022) - Full Pros and Cons." Style Factory, 25 Jan. 2022, https://www.stylefactoryproductions.com/blog/nimble-crm-review.

[6] "DBMS Keys: Primary, Foreign, Candidate and Super Key - Javatpoint." www.javatpoint.com, www.javatpoint.com/dbms-keys. Accessed 3 Oct. 2022.

[7] "Primary and Foreign Key Constraints - SQL Server." Microsoft Learn, 17 Aug. 2022, learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver16.

[8] Techopedia. "Relationship." Techopedia.com, 23 Sept. 2014, www.techopedia.com/definition/24438/relationship-databases#:%7E:text=A%20relationship%2C%20in%20the%20context,while%20linking%20disparate%20data%20items.

[9] "Intelligent Diagramming." *Lucidchart*, https://www.lucidchart.com/pages/.