Contact Management System

MSCS542L

Section 816

Undercover Martyn

Marist University

School of Computer Science and Mathematics

Submitted To: Dr. Reza Sadeghi

Spring 2025

Table of Contents

Table of Figures

Project Report of Contact Management System

Team Name

Undercover Martyn

Team Members

1. Reece Schenck reece.schenck1@marist.edu (Team Head)
2. Roman Huerta Manrique Roman.HuertaManrique1@marist.edu
3. Michelle Macina Michelle.Macina1@marist.edu

Description of Team Members

1. Reece Schenck

Hello, I am Reece Schenck. I am a computer science and cybersecurity major and I am the manager of the club volleyball team here at Marist. As I started this project as the only member of this group that made me the team head automatically. I chose to add and work with my other team members as they seem very hard working, intelligent, and enthusiastic about this project.

2. Roman

I obtained my Bachelor's degree in Informatics Engineering from the Pontifical Catholic University of Peru and began my professional career as a Web Developer Analyst at the same institution, where I worked as a Full-Stack Developer. Subsequently, I transitioned into the field of education, serving as a Mathematics instructor for two years as part of the TeachForPeru program. Most recently, I worked as a Content Creator at Crack The Code, a coding school based in Latin America. I am proficient in PHP, HTML, JavaScript, Python, Java, and SQL. Currently, I am pursuing a Master of Science in Computer Science at Marist University, with a concentration in Artificial Intelligence.

3. Michelle

I completed a Bachelor of Arts in International Studies at Elon University and am now looking to obtain a Master of Science in Computer Science at Marist University.

Project Objective

PROJECT TITLE: CONTACT MANAGEMENT SYSTEM

Summary: THE CONTACT MANAGEMENT SYSTEM (CMS) RECORDS VARIED INFORMATION, SUCH AS CELL NUMBER, FAX NUMBER, HOME NUMBER, EMAIL, AND ADDRESS. THE USERS SHOULD SECURELY ADD RECORDS & UPDATE THEM. THE CMS WILL STORE THE DATA OF DIFFERENT USER TYPES IN DISTINCT SQL TABLES. THIS SYSTEM SHOULD AT LEAST SUPPORT THE FOLLOWING CAPABILITIES:

1. REQUESTING ADMIN USER AND PASSWORD FOR LOG IN (A STRING OF AT LEAST 8 CHARACTERS)

2. CHANGE THE ADMIN USER AND PASSWORD

3. ADMIN USER SHOULD BE ABLE TO ADD A USER TO CMS BY CREATING A NEW USERNAME AND PASSWORD FOR NORMAL USERS, WHO ARE NOT ABLE TO DEFINE OR REMOVE A USER

4. ADMIN USER SHOULD BE ABLE TO REMOVE A USER FROM CMS BY REMOVING HIS/HER USERNAME, PASSWORD, AND HIS/HER OTHER CORRESPONDING DATA

5. EACH USER SHOULD BE ABLE TO:
    a. ADD MULTIPLE SETS OF CONTACT INFORMATION WITH THE FOLLOWING DETAILS: FIRST NAME, SURNAME, CELL PHONE, WORKPLACE PHONE, FAX NUMBER, EMAIL ADDRESS, GENDER, AND AGE
    b. REMOVE A CONTACT RECORD
    c. EDIT THE CONTACT RECORD'S DETAILS
    d. SEARCH THROUGH CONTACTS BASED ON ONE OR SEVERAL FEATURES AND LIST THE RESULTS ON THE SCREEN. FOR INSTANCE, IT SHOULD BE ABLE TO RETURN THE CELL PHONE NUMBER OF A SPECIFIC NAME.

6. CMS SHOULD BE A USER-FRIENDLY SOFTWARE, SUCH THAT:

a. IT SHOWS A WARNING IF A USER TRIES TO INPUT CONTACT INFORMATION WITH A NAME THAT EXISTS IN THE HISTORY.
b. CMS SHOULD SHOW A WELCOME PAGE
c. CMS SHOULD SHOW A MENU OF ALL FUNCTIONS TO THE USER
d. CMS SHOULD PROVIDE THE REPORTS IN A TABULAR FORM
e. CMS SHOULD PROVIDE AN EXIT FUNCTION

Review Related Works

As seen throughout similar Contact Management Systems, they typically are not all encompassing and can be hard to use. In particular, when using HubSpot(2), Pipedrive(5), and Insightly(6), it is clear that they all specialized in some way over the other, each with their own positive and negative aspects.

HubSpot offers many positive features. Some of which include its very user friendly interface offered completely for free. It also offers automation and easy integration with other business solutions. Along with these positives, HubSpot also has some drawbacks. The main Issue is that some advanced features are locked behind the paid version. Even so, regardless of the version of HubSpot being used, there is a severe lack of customization to each individual user's unique business models.

Diving into Pipedrive, it offers similar features as HubSpot except with a more tailored financial sales focus. For the positives offered, Pipedrive is very customisable with features like notifications and reminders for activities or follow up appointments. However, similar to HubSpot, Pipedrive also has limited available advanced features on account of it being tailored specifically for financial sales. On top of this, it is difficult to integrate Pipedrive without having to use other tools to work around its lack of compatibility.

Finally examining Insightly, it offers numerous positive features. These include a user friendly interface, easy integration, numerous comprehensive features offered such as automation and other tools. Some negatives include the pricing as it is far more expensive than other available options. Similar to HubSpot, Insightly also has very limited customizations for features such as fields and layouts.

The Merits of the Project

The Contact Management System(CMS) has several merits that make it an excellent solution for managing contact information. It allows users to store and manage comprehensive details, including cell numbers, email addresses, fax numbers, home numbers, and physical addresses, ensuring all relevant data is centralized and easily accessible. The system enforces security through mandatory admin credentials with a minimum character length of eight and provides robust access control. Admins can manage user accounts by adding or removing users and assigning credentials, while normal users are restricted to contact management tasks, ensuring data security and administrative control.

The CMS offers a range of features for users, such as the ability to add, edit, and remove detailed contact records. Its efficient search filters enable users to quickly locate specific contacts based on fields like name or phone number. The system enhances user-friendliness by providing warnings for duplicate entries, a welcoming interface, menu-driven navigation, and tabular reports for clear and professional data presentation. Additionally, it includes error handling mechanisms and a convenient exit function to streamline operations.

End users benefit from the CMS's centralized and organized data storage, which uses SQL tables for fast and reliable data retrieval. The system prioritizes security and data protection through role-based access and strict login protocols. It is highly adaptable and scalable, catering to both administrators and regular users. The system saves time with its efficient search capabilities and customizable filters while offering professional reporting features for business or personal use.

The CMS is an all-in-one solution, integrating core features that are tailored to user needs. Unlike other tools such as HubSpot, Pipedrive, or Insightly, all of which specialize in certain areas and thus lack full customization or comprehensive functionality, the CMS balances adaptability, and functionality. It is designed with user ease in mind, featuring a welcoming interface and error handling to reduce the learning curve and maximize productivity. Furthermore, it eliminates dependency on external tools, serving as a reliable standalone solution. By addressing common pain points like limited customization and integration challenges found in competing tools, the CMS delivers a secure, user-friendly, and versatile product that effectively meets diverse end-user needs.

GitHub Repository Address

https://github.com/Reece-Schenck/MSCS542L_816_CONTACT-MANAGEMENT-SYSTEM_Undercover-Martyn

Entity Relationship Model

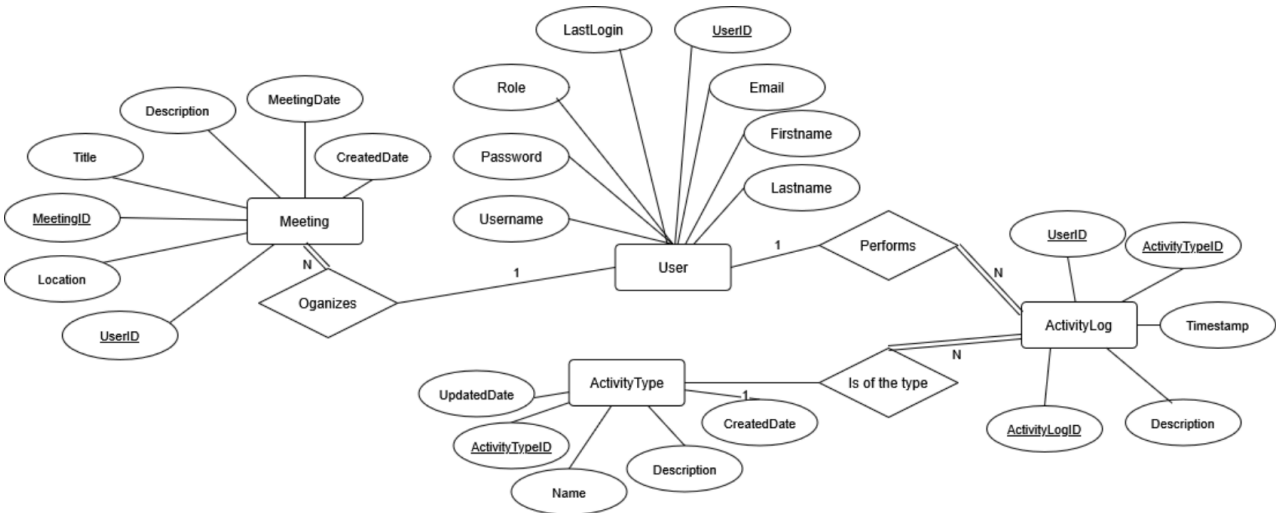
1. External Models

External Model 1: User and Activities

This model focuses on the relationship between users, their login credentials, and their relations to meetings and activity logs.

- **Entities**:
  - **User**: Represents the individuals who use the CMS.
  - **Meeting**: Stores details of meetings.
  - **ActivityLog**: Tracks user activities.
  - **ActivityType**: Categorizes logged activities.
- **Relationships**:
  - A **User** can perform many **ActivityLogs** (1:N).
  - A **User** can organize many **Meetings** (1:N).
  - An **ActivityLog** is an **ActivityType** (1:N).

External Model 1 Diagram:

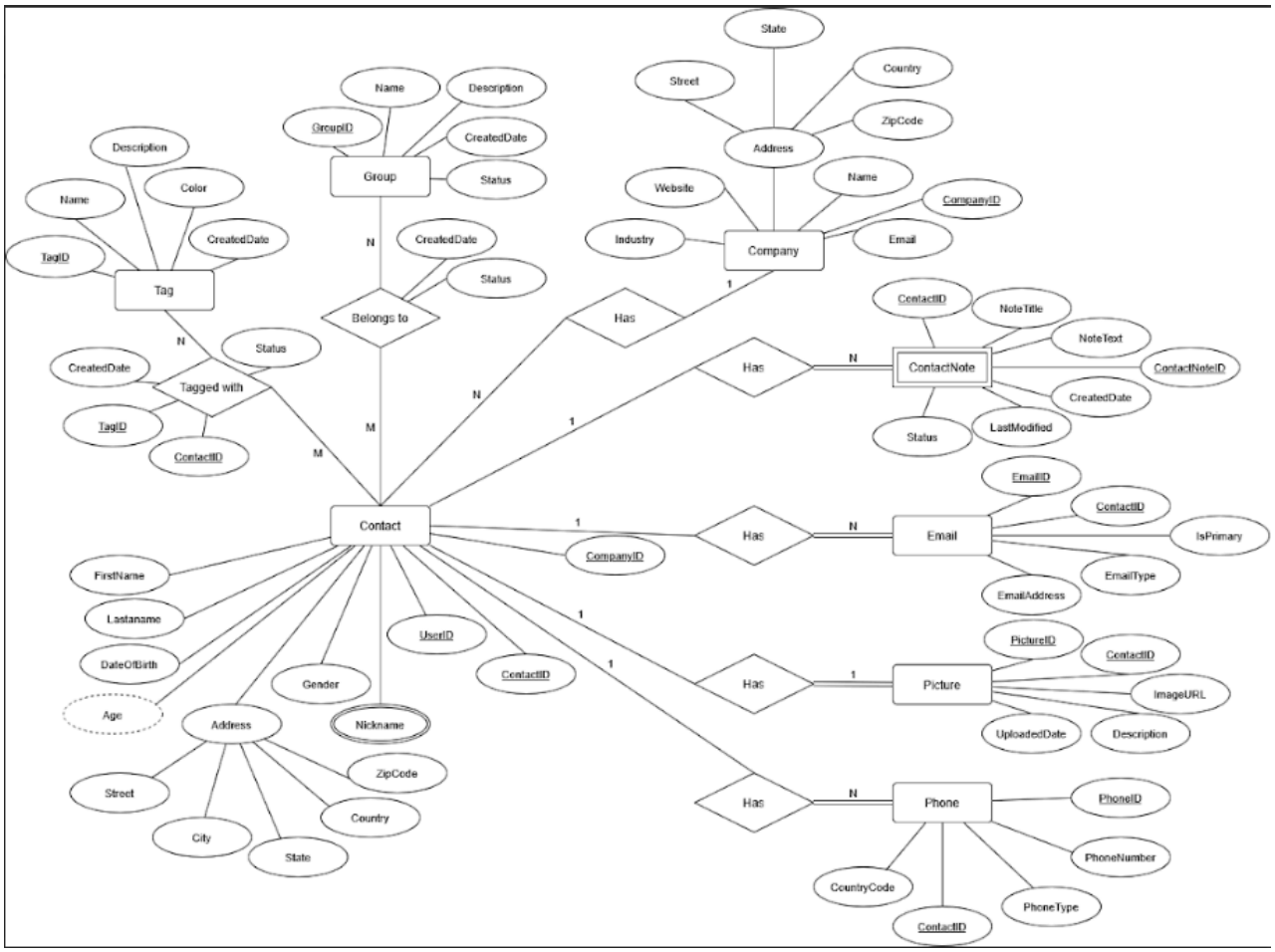


External Model 2: Contacts and Contact Management

This model focuses on the relationship between contacts and all their elements.

- **Entities**:
  - **Contact**: Stores contact details.
  - **Tag**: Labels assigned to contacts.
  - **Group**: Groups contacts for better organization.
  - **Company**: Companies that contacts are associated with.
  - **ContactNote**: Notes related to a contact.
  - **Email**: Stores email addresses.
  - **Picture**: Stores profile image.
  - **Phone**: Stores phone numbers and types.
- **Relationships**:
  - A **Contact** can have many **Phones** (1:N).
  - A **Contact** can have many **Emails** (1:N).
  - A **Contact** has only one profile **Picture** (1:N).
  - A **Contact** can have numerous **ContactNotes** (1:N).
  - Multiple **Contacts** can have a **Company** (1:N)
  - Multiple **Contacts** can belong to multiple **Groups** (M:N)
  - Multiple **Contacts** can have multiple **Tags** (M:N)

External Model 2 Diagram:



Selections:

Selecting the Entities

Entities represent real-world objects or concepts that are relevant to the CMS. I selected the entities based on the core operations and features of the system:

- **User**: The individuals who interact with the system, such as admins or normal users.
- **Contact**: Stores contact details, including personal and professional information.
- **Phone**: Stores different phone numbers associated with contacts.
- **Email**: Stores email addresses linked to contacts.
- **Picture**: Stores profile images of contacts.
- **ActivityLog**: Tracks user activities within the system.
- **ActivityType**: Categorizes types of activities logged.
- **ContactNote**: Stores notes related to a contact.
- **Group**: Represents contact groups for better organization.
- **Tag**: Represents keywords or labels assigned to contacts.

- **Meeting**: Stores details of meetings involving contacts.
- **Company**: Represents companies that contacts are associated with.

Selecting the Attributes

For each entity, I selected relevant attributes that capture all necessary details for effective functionality in the CMS:

- **User**: UserID, Username, Password, Role, Email, FirstName, LastName, LastLogin
- **Contact**: ContactID, UserID, CompanyID, FirstName, LastName, DateOfBirth, Gender, Nickname, Address (Street, City, State, Zip, Country), Age
- **Phone**: PhoneID, ContactID, PhoneNumber, PhoneType, CountryCode
- **Email**: EmailID, ContactID, EmailAddress, EmailType, IsPrimary
- **Picture**: PictureID, ContactID, ImageURL, UploadedDate, Description
- **ActivityLog**: ActivityLogID, UserID, ActivityTypeID, Timestamp, Description
- **ActivityType**: ActivityTypeID, Name, Description, UpdatedDate, CreatedDate
- **ContactNote**: NoteID, ContactID, NoteTitle, NoteText, CreatedDate, LastModified, Status
- **Group**: GroupID, Name, Description, CreatedDate, Status
- **Tag**: TagID, Name, Description, CreatedDate, Color
- **Meeting**: MeetingID, UserID, Title, Description, Location, MeetingDate, CreatedDate
- **Company**: CompanyID, Name, Industry, Address (Street, City, State, Zip, Country), Email, Website

Selecting the Relationships

The relationships define how entities are connected to one another. I chose relationships based on real-world interactions within the CMS system:

- **User "Owns" Contact (1:N)** → Each user can have multiple contacts, but each contact belongs to only one user.
- **Contact "Has" Phone (1:N)** → Each contact can have multiple phone numbers.
- **Contact "Has" Email (1:N)** → Each contact can have multiple email addresses.
- **Contact "Has" Picture (1:1)** → A contact can have only one profile picture.
- **User "Performs" ActivityLog (1:N)** → Each user can perform multiple activities logged in the system.
- **ActivityLog "Is of type" ActivityType (N:1)** → Each activity log entry must have a defined activity type.
- **Contact "Has" ContactNote (1:N)** → Each contact can have multiple notes (Weak Entity with Total Participation).
- **Contact "Belongs to" Group (M:N)** → Implemented using the **ContactGroupMapping** table.
- **Contact "Tagged with" Tag (M:N)** → Implemented using the **ContactTagMapping** table.
- **Meeting "Invites" Contact (M:N)** → Implemented using the

MeetingContactMapping table.
- **User "Organizes" Meeting (1:N)** → Each meeting is created by one user.
- **Contact "Has" Company (N:1)** → Multiple contacts can belong to the same company.

Selecting the Participations

I decided on partial or total participation based on the data requirements for each relationship:

Total Participation:

- **Contact** → **Phone**: Every contact must have at least one phone number.
- **Contact** → **ContactNote**: Every note must belong to a contact.
- **User** → **ActivityLog**: Every activity log must be associated with a user.

Partial Participation:

- **Contact** → **Address**: Not every contact has an address.
- **Contact** → **Picture**: Some contacts may not have a profile picture.

Selecting the Cardinalities

Cardinalities define how many instances of one entity can be related to instances of another entity. I selected cardinalities based on how entities interact:

- **1:1 Cardinality**:
  - **Contact** → **Picture**: A contact can have only one profile picture.
- **1:N Cardinality**:
  - **User** → **Contact**: A user can have multiple contacts.
  - **Contact** → **Phone**: A contact can have multiple phone numbers.
- **M:N Cardinality**:
  - **Contact** → **Group**: A contact can belong to multiple groups.
  - **Contact** → **Tag**: A contact can have multiple tags.
  - **Meeting** → **Contact**: Multiple contacts can be invited to meetings.

Descriptions:

Entities

1. **User**
   - Represents the individuals interacting with the CMS, either admins or regular users.
2. **Contact**
   - Stores detailed information about each contact.
3. **Phone**
   - Stores multiple phone numbers for contacts.

4. **Email**
    - Stores multiple email addresses for contacts.
5. **Picture**
    - Stores profile images for contacts.
6. **ActivityLog**
    - Records actions performed by users.
7. **ActivityType**
    - Categorizes different types of logged activities.
8. **ContactNote**
    - Stores notes related to contacts.
9. **Group**
    - Represents categories for contacts.
10. **Tag**
    - Represents tags assigned to contacts.
11. **Meeting**
    - Stores meeting details involving contacts.
12. **Company**
    - Stores details about organizations related to contacts.

Relationships and Cardinalities

1. **User** → **Contact** (1:N)
    - A user can **have** multiple contacts, but each contact belongs to a single user.
2. **Contact** → **Phone** (1:N)
    - A contact can **have** multiple phone numbers, but each phone number belongs to only one contact.
3. **Contact** → **Email** (1:N)
    - A contact can **have** multiple email addresses, categorized by type.
4. **Contact** → **Picture** (1:1)
    - Each contact can **have** only one profile picture.
5. **User** → **ActivityLog** (1:N)
    - A user can **perform** multiple logged activities, such as updating a contact's information.
6. **ActivityLog** → **ActivityType** (1:N)
    - Each logged activity **is assigned** a predefined activity type.
7. **Contact** → **ContactNote** (1:N)
    - A contact can **have** multiple notes.
8. **Contact** → **Group** (M:N)
    - Multiple contacts can **belong to** multiple groups.
9. **Contact** → **Tag** (M:N)
    - A contact can **have** multiple tags.
10. **Meeting** → **Contact** (M:N)
    - Multiple contacts can **attend** a meeting, and a contact can be **invited** to multiple meetings.
11. **User** → **Meeting** (1:N)
    - Each meeting is **created** by a single user, but a user can **organize** multiple

meetings.

12. **Company** → **Contact** (1:N)
    - Multiple contacts can be associated with the same company.

Participations

1. **Partial Participation**:
    - **Contact** → **Address**: Not all contacts may have an address.
    - **Contact** → **Picture**: Some contacts may not have a profile picture.
    - **Contact** → **Company**: Some contacts may not have an associated company.
    - **Company** → **Contact**: Some companies may not have an associated contact.
2. **Total Participation**:
    - **Contact** → **Phone**: Every contact must have at least one phone number.
    - **Picture** → **Contact**: Every profile picture must have a contact.
    - **Contact** → **ContactNote**: Every contact note must be associated with a contact.
    - **User** → **ActivityLog**: Every activity log entry must be linked to a user.

Advanced Features:

- **Composite Attribute**: **Street, City, State, ZipCode,** and **Country** combine into a composite attribute for **Address**.
- **Multivalued Attribute**: **Nickname** in **Contact** allows multiple alternative names for a contact.
- **Derived Attribute**: **Age** in **Contact** is calculated based on **DateOfBirth**.

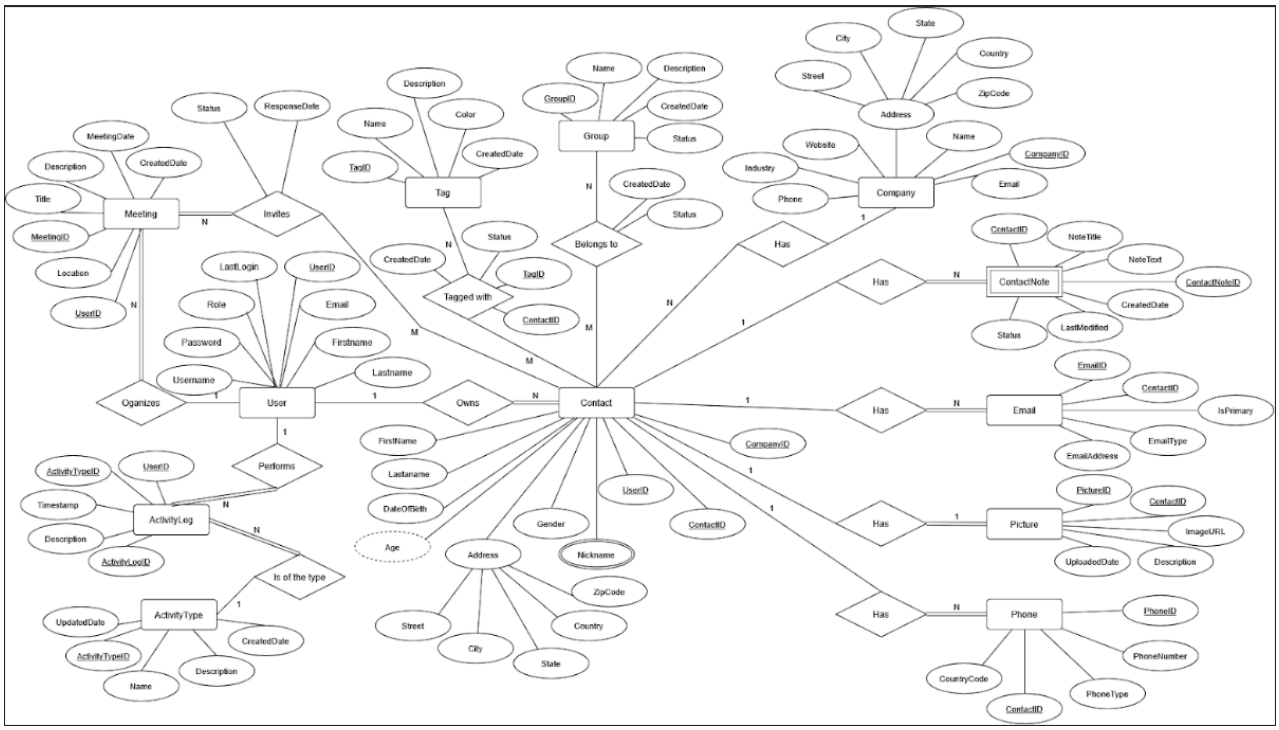
2. Conceptual Model

Entities and Attributes

1. **User**
    - Attributes: `UserID, Username, Password, Email, Role, FirstName, LastName, LastLogin`
2. **Contact**
    - Attributes: `ContactID, UserID, CompanyID, FirstName, LastName, DateOfBirth, Gender, Nickname, Address, Age`
3. **Phone**
    - Attributes: `PhoneID, ContactID, PhoneNumber, PhoneType, CountryCode`
4. **Email**
    - Attributes: `EmailID, ContactID, EmailAddress, EmailType, IsPrimary`

5. **Picture**
    - Attributes: `PictureID, ContactID, ImageURL, UploadedDate, Description`
6. **ActivityLog**
    - Attributes: `ActivityLogID, UserID, ActivityTypeID, Timestamp, Description`
7. **ActivityType**
    - Attributes: `ActivityTypeID, Name, Description, UpdatedDate, CreatedDate`
8. **ContactNote**
    - Attributes: `NoteID, ContactID, NoteTitle, NoteText, CreatedDate, LastModified, Status`
9. **Group**
    - Attributes: `GroupID, Name, Description, CreatedDate, Status`
10. **Tag**
    - Attributes: `TagID, Name, Description, CreatedDate, Color`
11. **Meeting**
    - Attributes: `MeetingID, Title, Description, Location, Time, CreatedDate`
12. **Company**
    - Attributes: `CompanyID, Name, Industry, Address, Email, Website`

ER Diagram:



Enhanced Entity Relationship Model

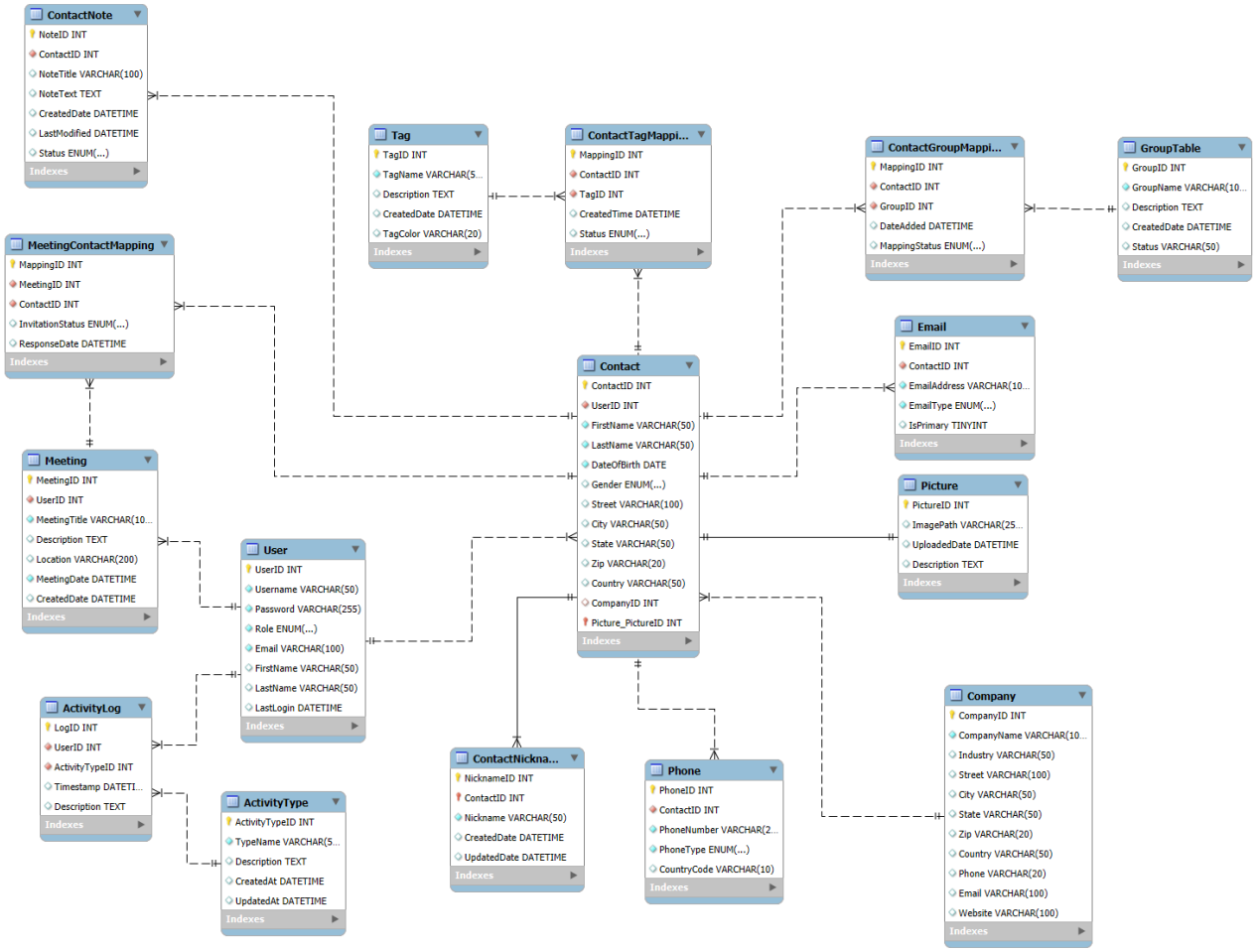
Each entity, such as **User**, **Contact**, and **Phone**, was designed with at least five attributes, including a primary key(PK) to ensure unique identification of records. For example, the **UserID** in the **User** entity and **ContactID** in the **Contact** entity serve as primary keys. Foreign keys (FK) were introduced to establish relationships between entities, such as the **UserID** in the **Contact** entity, which references the **UserID** in the **User** entity.

The relationships in the EER model were defined using 1:1, 1:N, and M:N cardinalities. A 1:1 relationship, such as between **Contact** and **Picture**, ensures that each contact may have only one picture. A 1:N relationship, such as between **User** and **Contact**, allows each user to have multiple contacts. Meanwhile, an M:N relationship, such as between **Contact** and **Tag**, enables many-to-many interactions where a contact may have zero or many tags, and a tag may be related to one or more contacts. Total participation was

applied where every instance of an entity must participate in a relationship, such as every **Contact** requiring at least one **Phone**. Partial participation was used where relationships are optional, such as not all **Contacts** having a Company.

The model also incorporates advanced features, including multivalued, composite, and derived attributes. For example, the **Nickname** attribute in the **Contact** entity supports multiple values. The **Address** entity was designed as a composite attribute, breaking down into **Street, City**, **State**, **ZipCode**, and **Country**. Additionally, a derived attribute, such as **Age** in the **Contact** entity, was calculated based on the **DateOfBirth** attribute.

EER Diagram:



Data Types Exploration

The DATE and TIME and Spatial data types serve different purposes. DATE and TIME data type is used to store temporal information such as dates, times, or both. These types ensure accurate recording and manipulation of time-related data, supporting operations like date comparisons, intervals, and calculations. We use date and time data types to manage events, track deadlines, and perform time-based queries in applications like scheduling, birthdays, holidays, transaction timestamps, and shift timings.

Spatial data type is used to store and manage geometric or geographic data, such as points, lines, and polygons. It enables efficient handling of spatial relationships, allowing for operations like distance calculations, intersections, and spatial indexing. We use spatial data types to accurately represent real-world locations, and integrate geographic information into applications like mapping, navigation, and urban planning.

Database Development

In the next section we will describe the SQL code to implement the database. We will explain some parts that will repeat.

Create schema(database):

```
CREATE SCHEMA IF NOT EXISTS `MyCMS` DEFAULT CHARACTER SET utf8 ;
USE `MyCMS` ;
```

This piece of code creates the schema where all the tables will be created in, the character set says what set of characters will be used, in this case the UTF8 character encoding will be used, this has more support for non english characters.

Table creation:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Contact` (
  `ContactID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `FirstName` VARCHAR(50) NOT NULL,
  `LastName` VARCHAR(50) NOT NULL,
  `DateOfBirth` DATE NOT NULL,
  `Gender` ENUM('M', 'F', 'Other') NULL DEFAULT NULL,
  `Street` VARCHAR(100) NULL DEFAULT NULL,
```

```
`City` VARCHAR(50) NULL DEFAULT NULL,
`State` VARCHAR(50) NULL DEFAULT NULL,
`Zip` VARCHAR(20) NULL DEFAULT NULL,
`Country` VARCHAR(50) NULL DEFAULT NULL,
`CompanyID` INT NULL DEFAULT NULL,
PRIMARY KEY (`ContactID`),
INDEX (`UserID` ASC) VISIBLE,
INDEX (`CompanyID` ASC) VISIBLE,
CONSTRAINT ``
    FOREIGN KEY (`UserID`)
    REFERENCES `MyCMS`.`User` (`UserID`)
    ON DELETE CASCADE,
CONSTRAINT ``
    FOREIGN KEY (`CompanyID`)
    REFERENCES `MyCMS`.`Company` (`CompanyID`)
    ON DELETE SET NULL;
```

This code will create a table with the name “Contact” and it specifies that it is in the “MyCMS” schema. Later it defines the attributes with the type of data and if they are or not NULL and the default value for it.

The primary keys are defined with the PRIMARY KEY command and the relations between the tables are defined by using CONSTRAINT and FOREIGN KEY where it specifies to what table and attribute the tables attribute is related, and also the actions that will be taken when updating and deleting are specified.

And the indexes are also defined using the INDEX command, it specifies what attribute the index is taking and if they are visible or not.

Tables description:

1. User Table

Description:

The User table stores information about system users, including their login credentials, personal details, and role within the system.

Relations:

- Referenced by the Contact table to associate a user with contacts.
- Referenced by the ActivityLog table to log user activities.

Attribute	Data Type	Key	Description
UserID	INT AUTO_INCREMENT	PK	Unique identifier for each user.
Username	VARCHAR(50) NOT NULL UNIQUE		Stores the user's login username, which must be unique.
Password	VARCHAR(255) NOT NULL		Securely stores hashed user passwords using a robust encryption mechanism.
Role	ENUM('ADMIN', 'NORMAL') NOT NULL		Defines user roles and their access levels.
Email	VARCHAR(100) NOT NULL UNIQUE		Stores user email, ensuring uniqueness.
FirstName	VARCHAR(50) NULL		First name of the user.
LastName	VARCHAR(50) NULL		Last name of the user.
LastLogin	DATETIME NULL		Stores the last login timestamp of the user to track activity.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`User` (
  `UserID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `Username` VARCHAR(50) NOT NULL,
  `Password` VARCHAR(255) NOT NULL,
  `Role` ENUM('ADMIN', 'NORMAL') NOT NULL,
  `Email` VARCHAR(100) NOT NULL,
  `FirstName` VARCHAR(50) NULL DEFAULT NULL,
  `LastName` VARCHAR(50) NULL DEFAULT NULL,
  `LastLogin` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE INDEX (`Username` ASC) VISIBLE,
  UNIQUE INDEX (`Email` ASC) VISIBLE);
```

2. Company Table

Description:

The Company table holds details about organizations that interact with the system. It is referenced by the Contact table to associate a contact with a company.

Relations:

- Referenced by the Contact table to associate contacts with companies.

Attribute	Data Type	Key	Description

CompanyID	INT AUTO_INCREMENT	PK	Unique identifier for each company.
CompanyName	VARCHAR(100) NOT NULL UNIQUE		Stores the company name, ensuring uniqueness.
Industry	VARCHAR(50) NULL		Represents the industry in which the company operates.
Street	VARCHAR(100) NULL		The street address of the company.
City	VARCHAR(50) NULL		The city where the company is located.
State	VARCHAR(50) NULL		The state where the company is located.
Zip	VARCHAR(20) NULL		Zip code of the company's address.
Country	VARCHAR(50) NULL		Country where the company operates.
Phone	VARCHAR(20) NULL		Contact phone number of the company.
Email	VARCHAR(100) NULL		The company's official email.
Website	VARCHAR(100) NULL		The company's website URL.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Company` (
  `CompanyID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `CompanyName` VARCHAR(100) NOT NULL,
  `Industry` VARCHAR(50) NULL DEFAULT NULL,
  `Street` VARCHAR(100) NULL DEFAULT NULL,
  `City` VARCHAR(50) NULL DEFAULT NULL,
  `State` VARCHAR(50) NULL DEFAULT NULL,
  `Zip` VARCHAR(20) NULL DEFAULT NULL,
  `Country` VARCHAR(50) NULL DEFAULT NULL,
  `Phone` VARCHAR(20) NULL DEFAULT NULL,
  `Email` VARCHAR(100) NULL DEFAULT NULL,
  `Website` VARCHAR(100) NULL DEFAULT NULL,
  PRIMARY KEY (`CompanyID`),
  UNIQUE INDEX (`CompanyName` ASC) VISIBLE);
```

3. Picture Table

Description:

The Picture table is used to store metadata about images uploaded to the system. The Contact table references it to associate a contact with a profile picture.

Relations:

- Referenced by the Contact table to assign profile pictures to contacts.

Attribute	Data Type	Key	Description
PictureID	INT AUTO_INCREMENT	PK	Unique identifier for each picture.
ContactID	INT	FK	Foreign key referencing Contact(ContactID).
ImagePath	VARCHAR(255) NULL		Stores the file path of the image.
UploadedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Records the timestamp of when the image was uploaded.
Description	TEXT NULL		Provides additional details about the image.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Picture` (
  `PictureID` INT AUTO_INCREMENT,
  `ContactID` INT NOT NULL UNIQUE,
  `ImagePath` VARCHAR(255) NULL DEFAULT NULL,
  `UploadedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `Description` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`PictureID`));
```

4. Contact Table

Description:

The Contact table stores personal and professional details about individuals who are associated with companies or users.

Relations:

- References the User table to associate contacts with users.
- References the Company table to link contacts to companies.
- References the Picture table to assign a profile picture to contacts.

Attribute	Data Type	Key	Description
ContactID	INT AUTO_INCREMENT	PK	Unique identifier for each contact.
UserID	INT NOT NULL	FK	Foreign key referencing User(UserID).
FirstName	VARCHAR(50) NOT NULL		Contact's first name.
LastName	VARCHAR(50) NOT NULL		Contact's last name.
DateOfBirth	DATE NOT NULL		Contact's date of birth.

Gender	ENUM('M', 'F', 'Other') NULL		Contact's gender.
Street	VARCHAR(100) NULL		Contact's street address.
City	VARCHAR(50) NULL		Contact's city.
State	VARCHAR(50) NULL		Contact's state.
Zip	VARCHAR(20) NULL		Contact's zip code.
Country	VARCHAR(50) NULL		Contact's country.
CompanyID	INT NULL	FK	Foreign key referencing Company(CompanyID).

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Contact` (
  `ContactID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `FirstName` VARCHAR(50) NOT NULL,
  `LastName` VARCHAR(50) NOT NULL,
  `DateOfBirth` DATE NOT NULL,
  `Gender` ENUM('M', 'F', 'Other') NULL DEFAULT NULL,
  `Street` VARCHAR(100) NULL DEFAULT NULL,
  `City` VARCHAR(50) NULL DEFAULT NULL,
  `State` VARCHAR(50) NULL DEFAULT NULL,
  `Zip` VARCHAR(20) NULL DEFAULT NULL,
  `Country` VARCHAR(50) NULL DEFAULT NULL,
  `CompanyID` INT NULL DEFAULT NULL,
  PRIMARY KEY (`ContactID`),
  INDEX (`UserID` ASC) VISIBLE,
  INDEX (`CompanyID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`UserID`)
      REFERENCES `MyCMS`.`User` (`UserID`)
      ON DELETE CASCADE,
  CONSTRAINT ``
      FOREIGN KEY (`CompanyID`)
      REFERENCES `MyCMS`.`Company` (`CompanyID`)
      ON DELETE SET NULL;
```

5. ContactNickname Table

Description:

The ContactNickname table stores alternative names used for contacts. This allows

flexibility in managing different names a contact might be known by.

Relations:

- References the Contact table to associate nicknames with a contact.
- A contact can have multiple nicknames.

Attribute	Data Type	Key	Description
ContactID	INT NOT NULL	PK	Primary key referencing Contact(ContactID).
Nickname	VARCHAR(50) NOT NULL	PK	The alternative name for the contact.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ContactNickname` (
  `ContactID` INT NOT NULL,
  `Nickname` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`Nickname`, `ContactID`),
  INDEX (`ContactID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE);
```

6. Phone Table

Description:

The Phone table stores phone numbers associated with contacts, allowing multiple phone numbers for each contact.

Relations:

- References the Contact table to associate phone numbers with a contact.
- Each contact can have multiple phone numbers (work, home, mobile, fax, etc.).

Attribute	Data Type	Key	Description
PhoneID	INT AUTO_INCREMENT	PK	Unique identifier for each phone number.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
PhoneNumber	VARCHAR(20) NOT NULL		The actual phone number.

PhoneType	ENUM('Cell', 'Home', 'Work', 'Fax') NOT NULL		Specifies the type of phone number.
CountryCode	VARCHAR(10) NULL		Stores the country code of the phone number.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Phone` (
  `PhoneID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `PhoneNumber` VARCHAR(20) NOT NULL,
  `PhoneType` ENUM('Cell', 'Home', 'Work', 'Fax') NOT NULL,
  `CountryCode` VARCHAR(10) NULL DEFAULT NULL,
  PRIMARY KEY (`PhoneID`),
  INDEX (`ContactID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE);
```

7. Email Table

Description:

The Email table stores email addresses associated with contacts, allowing multiple email addresses per contact.

Relations:

- References the Contact table to associate email addresses with a contact.
- Each contact can have multiple email addresses (personal, work, etc.).

Attribute	Data Type	Key	Description
EmailID	INT AUTO_INCREMENT	PK	Unique identifier for each email address.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
EmailAddress	VARCHAR(100) NOT NULL		The email address of the contact.
EmailType	ENUM('Personal', 'Work', 'Other') NOT NULL		Specifies the type of email address.
IsPrimary	TINYINT NULL DEFAULT FALSE		Indicates if this is the primary email for the contact.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Email` (
  `EmailID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `EmailAddress` VARCHAR(100) NOT NULL,
  `EmailType` ENUM('Personal', 'Work', 'Other') NOT NULL,
  `IsPrimary` TINYINT NULL DEFAULT FALSE,
  PRIMARY KEY (`EmailID`),
  INDEX (`ContactID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE);
```

8. ActivityType Table

Description:

The ActivityType table defines different types of activities that can be logged in the system, such as login attempts, updates, deletions, and other user actions.

Relations:

- Referenced by the ActivityLog table to classify logged activities.
- Used to maintain consistent activity tracking across the system.

Attribute	Data Type	Key	Description
ActivityTypeID	INT AUTO_INCREMENT	PK	Unique identifier for each activity type.
TypeName	VARCHAR(50) NOT NULL		The name of the activity type.
Description	TEXT NULL		A more detailed description of the activity type.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the activity type was added.
UpdatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP		Timestamp when the activity type was last updated.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ActivityType` (
  `ActivityTypeID` INT NULL DEFAULT NULL AUTO_INCREMENT,
```

```
  `TypeName` VARCHAR(50) NOT NULL,
  `Description` TEXT NULL DEFAULT NULL,
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `UpdatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`ActivityTypeID`),
  UNIQUE INDEX (`TypeName` ASC) VISIBLE);
```

9. ActivityLog Table

Description:

The ActivityLog table records system activities performed by users, such as logins, updates, and data modifications. It helps in tracking user interactions with the system for auditing purposes.

Relations:

- References the User table to associate an activity with the user who performed it.
- References the ActivityType table to classify the type of activity being logged.

Attribute	Data Type	Key	Description
LogID	INT AUTO_INCREMENT	PK	Unique identifier for each log entry.
UserID	INT NOT NULL	FK	Foreign key referencing User(UserID).
ActivityTypeID	INT NOT NULL	FK	Foreign key referencing ActivityType(ActivityTypeID).
Timestamp	DATETIME DEFAULT CURRENT_TIMESTAMP		The time when the activity was logged.
Description	TEXT NULL		Additional details about the logged activity.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ActivityLog` (
  `LogID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `ActivityTypeID` INT NOT NULL,
  `Timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `Description` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`LogID`),
  INDEX (`UserID` ASC) VISIBLE,
  INDEX (`ActivityTypeID` ASC) VISIBLE,
```

```
  CONSTRAINT ``
      FOREIGN KEY (`UserID`)
      REFERENCES `MyCMS`.`User` (`UserID`)
      ON DELETE CASCADE,
  CONSTRAINT ``
      FOREIGN KEY (`ActivityTypeID`)
      REFERENCES `MyCMS`.`ActivityType` (`ActivityTypeID`)
      ON DELETE CASCADE);
```

10. ContactNote Table

Description:

The ContactNote table stores notes associated with contacts, allowing users to maintain additional details or remarks about a contact.

Relations:

- References the Contact table to associate notes with a specific contact.

Attribute	Data Type	Key	Description
NoteID	INT AUTO_INCREMENT	PK	Unique identifier for each note.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
NoteTitle	VARCHAR(100) NULL		A title or short description for the note.
NoteText	TEXT NULL		The actual content of the note.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the note was created.
LastModified	DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP		Timestamp of the last modification to the note.
Status	ENUM('ACTIVE', 'REMOVED') DEFAULT 'ACTIVE'		The status of the note.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ContactNote` (
  `NoteID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `NoteTitle` VARCHAR(100) NULL DEFAULT NULL,
  `NoteText` TEXT NULL DEFAULT NULL,
```

```
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `LastModified` DATETIME NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `Status` ENUM('ACTIVE', 'REMOVED') NULL DEFAULT 'ACTIVE',
  PRIMARY KEY (`NoteID`),
  INDEX (`ContactID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE);
```

11. Group Table

Description:

The Group table stores groups that contacts can belong to. These groups help in organizing and categorizing contacts efficiently.

Relations:

- Referenced by the ContactGroupMapping table to associate contacts with groups.

Attribute	Data Type	Key	Description
GroupID	INT AUTO_INCREMENT	PK	Unique identifier for each group.
GroupName	VARCHAR(100) NOT NULL UNIQUE		The name of the group.
Description	TEXT NULL		Additional information about the group.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the group was created.
Status	VARCHAR(50) NULL		Status of the group (e.g., active, archived).

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Group` (
  `GroupID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `GroupName` VARCHAR(100) NOT NULL,
  `Description` TEXT NULL DEFAULT NULL,
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `Status` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`GroupID`),
  UNIQUE INDEX (`GroupName` ASC) VISIBLE);
```

12. ContactGroupMapping Table

Description:

The ContactGroupMapping table manages the many-to-many relationship between contacts and groups, allowing a contact to belong to multiple groups and a group to contain multiple contacts.

Relations:

- References the Contact table to associate contacts with groups.
- References the Group table to establish the group the contact belongs to.

Attribute	Data Type	Key	Description
MappingID	INT AUTO_INCREMENT	PK	Unique identifier for each mapping entry.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
GroupID	INT NOT NULL	FK	Foreign key referencing Group(GroupID).
DateAdded	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the contact was added to the group.
MappingStatus	ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE'		Status of the mapping.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ContactGroupMapping` (
  `MappingID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `GroupID` INT NOT NULL,
  `DateAdded` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `MappingStatus` ENUM('ACTIVE', 'INACTIVE') NULL DEFAULT 'ACTIVE',
  PRIMARY KEY (`MappingID`),
  INDEX (`ContactID` ASC) VISIBLE,
  INDEX (`GroupID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE,
  CONSTRAINT ``
      FOREIGN KEY (`GroupID`)
      REFERENCES `MyCMS`.`Group` (`GroupID`)
      ON DELETE CASCADE);
```

13. Tag Table

Description:

The Tag table stores different tags that can be assigned to contacts. Tags help in categorizing contacts for easy filtering and identification.

Relations:

- Referenced by the ContactTagMapping table to associate contacts with tags.

Attribute	Data Type	Key	Description
TagID	INT AUTO_INCREMENT	PK	Unique identifier for each tag.
TagName	VARCHAR(50) NOT NULL UNIQUE		The name of the tag.
Description	TEXT NULL		Additional details about the tag.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the tag was created.
TagColor	VARCHAR(20) NULL		Optional color assigned to the tag.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Tag` (
  `TagID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `TagName` VARCHAR(50) NOT NULL,
  `Description` TEXT NULL DEFAULT NULL,
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `TagColor` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`TagID`),
  UNIQUE INDEX (`TagName` ASC) VISIBLE);
```

14. ContactTagMapping Table

Description:

The ContactTagMapping table manages the many-to-many relationship between contacts and tags, allowing a contact to have multiple tags and a tag to be associated with multiple contacts.

Relations:

- References the Contact table to associate contacts with tags.
- References the Tag table to define the tag assigned to the contact.

Attribute	Data Type	Key	Description
MappingID	INT AUTO_INCREMENT	PK	Unique identifier for each mapping entry.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
TagID	INT NOT NULL	FK	Foreign key referencing Tag(TagID).
CreatedTime	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the tag was assigned to the contact.
Status	ENUM('ACTIVE', 'REMOVED') DEFAULT 'ACTIVE'		Status of the tag assignment.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ContactTagMapping` (
  `MappingID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `TagID` INT NOT NULL,
  `CreatedTime` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `Status` ENUM('ACTIVE', 'REMOVED') NULL DEFAULT 'ACTIVE',
  PRIMARY KEY (`MappingID`),
  INDEX (`ContactID` ASC) VISIBLE,
  INDEX (`TagID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE,
  CONSTRAINT ``
      FOREIGN KEY (`TagID`)
      REFERENCES `MyCMS`.`Tag` (`TagID`)
      ON DELETE CASCADE);
```

15. Meeting Table

Description:

The Meeting table stores details about meetings involving users and contacts. It helps in managing scheduled events, discussions, and appointments.

Relations:

- References the User table to associate a meeting with a user who organized it.
- Referenced by the MeetingContactMapping table to link contacts with meetings.

Attribute	Data Type	Key	Description
MeetingID	INT AUTO_INCREMENT	PK	Unique identifier for each meeting.
UserID	INT NOT NULL	FK	Foreign key referencing User(UserID).
MeetingTitle	VARCHAR(100) NOT NULL		Title of the meeting.
Description	TEXT NULL		Additional details about the meeting.
Location	VARCHAR(200) NULL		The location where the meeting is held.
MeetingDate	DATETIME NOT NULL		Scheduled date and time for the meeting.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the meeting was created.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`Meeting` (
  `MeetingID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `UserID` INT NOT NULL,
  `MeetingTitle` VARCHAR(100) NOT NULL,
  `Description` TEXT NULL DEFAULT NULL,
  `Location` VARCHAR(200) NULL DEFAULT NULL,
  `MeetingDate` DATETIME NOT NULL,
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`MeetingID`),
  INDEX (`UserID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`UserID`)
      REFERENCES `MyCMS`.`User` (`UserID`)
      ON DELETE CASCADE);
```

16. MeetingContactMapping Table

Description:

The MeetingContactMapping table manages the many-to-many relationship between meetings and contacts, allowing multiple contacts to participate in a single meeting.

Relations:

- References the Meeting table to associate a contact with a meeting.
- References the Contact table to specify the contact participating in a meeting.

Attribute	Data Type	Key	Description
MappingID	INT AUTO_INCREMENT	PK	Unique identifier for each mapping entry.
MeetingID	INT NOT NULL	FK	Foreign key referencing Meeting(MeetingID).
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
InvitationStatus	ENUM('Accepted', 'Declined', 'Pending') DEFAULT 'Pending'		The status of the meeting invitation for the contact.
ResponseDate	DATETIME NULL		The date the contact responded to the invitation.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`MeetingContactMapping` (
  `MappingID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `MeetingID` INT NOT NULL,
  `ContactID` INT NOT NULL,
  `InvitationStatus` ENUM('Accepted', 'Declined', 'Pending') NULL DEFAULT
'Pending',
  `ResponseDate` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`MappingID`),
  INDEX (`MeetingID` ASC) VISIBLE,
  INDEX (`ContactID` ASC) VISIBLE,
  CONSTRAINT ``
      FOREIGN KEY (`MeetingID`)
      REFERENCES `MyCMS`.`Meeting` (`MeetingID`)
      ON DELETE CASCADE,
  CONSTRAINT ``
      FOREIGN KEY (`ContactID`)
      REFERENCES `MyCMS`.`Contact` (`ContactID`)
      ON DELETE CASCADE);
```

Handling Foreign Key Constraints

To import data we created a SQL Script file that will have all the INSERTS that will be needed. This file inserts the data in order so that we don't have problems with the Foreign Key violations. But as part of the process we recreated a violation by trying to insert data to a dependent table before inserting data to the parent table.

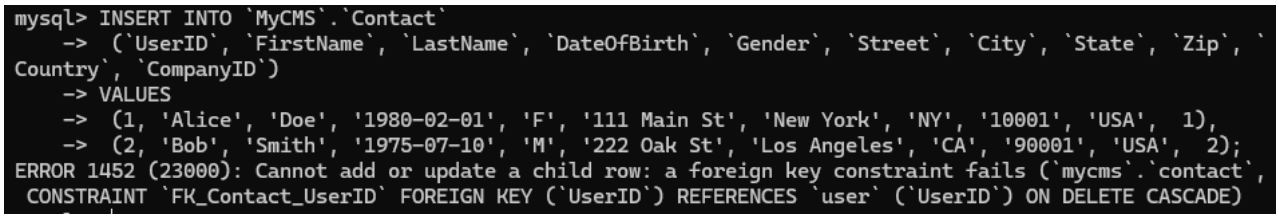
For example, trying to execute the next script will give us an error:

```
INSERT INTO `MyCMS`.`Contact`
      (`UserID`, `FirstName`, `LastName`, `DateOfBirth`, `Gender`, `Street`,
`City`, `State`, `Zip`, `Country`, `CompanyID`)
VALUES
      (1, 'Alice', 'Doe', '1980-02-01', 'F', '111 Main St', 'New York', 'NY',
'10001', 'USA',  1),
      (2, 'Bob', 'Smith', '1975-07-10', 'M', '222 Oak St', 'Los Angeles',
'CA', '90001', 'USA',  2);

INSERT INTO `MyCMS`.`User`
      (`Username`, `Password`, `Role`, `Email`, `FirstName`, `LastName`,
`LastLogin`)
VALUES
      ('admin',
'ef92b778bafe771e89245b89ecbc70ec1e5e14bba5b7b43eaf5f6a2e5d0b62d1', 'ADMIN',
'admin@mycms.com', 'Admin', 'User', '2025-02-22 09:00:00'),
      ('jsmith',
'a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5c6d7e8f90123456789abcdef0', 'NORMAL',
'jsmith@mycms.com', 'John', 'Smith', '2025-02-21 16:45:00');

INSERT INTO `MyCMS`.`Company`
      (`CompanyName`, `Industry`, `Street`, `City`, `State`, `Zip`,
`Country`, `Phone`, `Email`, `Website`)
VALUES
      ('Tech Innovators Inc', 'Technology', '123 Tech Ave', 'San Francisco',
'CA', '94107', 'USA', '415-555-0100', 'info@techinnovators.com',
'www.techinnovators.com');
```

Because the Contact table has some fields that depend on the other tables, so as it doesn't find the records needed it will throw an FK violation. The error is shown in the next image:



To avoid this error we decided to use the SET FOREIGN_KEY_CHECKS statement to change the value of the variable to disable the foreign key check while the data was being inserted.

```
SET FOREIGN_KEY_CHECKS = 0;

-- Insert queries

SET FOREIGN_KEY_CHECKS = 1;
```

Importing data

Here is the complete SQL script for inserting the data:

```
SET FOREIGN_KEY_CHECKS = 0;  -- Disable foreign key checks

INSERT INTO `MyCMS`.`User`
      (`Username`, `Password`, `Role`, `Email`, `FirstName`, `LastName`,
`LastLogin`)
VALUES
      ('admin',
'ef92b778bafe771e89245b89ecbc70ec1e5e14bba5b7b43eaf5f6a2e5d0b62d1', 'ADMIN',
'admin@mycms.com', 'Admin', 'User', '2025-02-22 09:00:00'),
      ('jsmith',
'a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5c6d7e8f90123456789abcdef0', 'NORMAL',
'jsmith@mycms.com', 'John', 'Smith', '2025-02-21 16:45:00'),
      ('amiller',
'b1c2d3e4f5061728394a5b6c7d8e9f00112233445566778899aabbccddeeff00', 'NORMAL',
'amiller@mycms.com', 'Alice', 'Miller', '2025-02-20 11:30:00'),
      ('bjohnson',
'0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef', 'NORMAL',
'bjohnson@mycms.com', 'Bob', 'Johnson', '2025-02-19 14:20:00'),
      ('cwalker',
'fedcba9876543210fedcba9876543210fedcba9876543210fedcba9876543210', 'NORMAL',
'cwalker@mycms.com', 'Carol', 'Walker', '2025-02-18 08:15:00'),
      ('dlee',
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa', 'NORMAL',
'dlee@mycms.com', 'David', 'Lee', '2025-02-17 12:00:00'),
      ('eroberts',
'bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb', 'NORMAL',
'eroberts@mycms.com', 'Emma', 'Roberts', '2025-02-16 17:30:00'),
```

```
      ('fclark',
'ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc', 'NORMAL',
'fclark@mycms.com', 'Frank', 'Clark', '2025-02-15 09:45:00'),
      ('gmartin',
'ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd', 'NORMAL',
'gmartin@mycms.com', 'Grace', 'Martin', '2025-02-14 11:15:00'),
      ('hscott',
'eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee', 'NORMAL',
'hscott@mycms.com', 'Henry', 'Scott', '2025-02-13 10:30:00');

INSERT INTO `MyCMS`.`Company`
      (`CompanyName`, `Industry`, `Street`, `City`, `State`, `Zip`,
`Country`, `Phone`, `Email`, `Website`)
VALUES
      ('Tech Innovators Inc', 'Technology', '123 Tech Ave', 'San Francisco',
'CA', '94107', 'USA', '415-555-0100', 'info@techinnovators.com',
'www.techinnovators.com'),
      ('HealthPlus Corp', 'Healthcare', '456 Wellness Blvd', 'Los Angeles',
'CA', '90017', 'USA', '323-555-0200', 'contact@healthplus.com',
'www.healthplus.com'),
      ('Green Energy Solutions', 'Energy', '789 Green Way', 'Austin', 'TX',
'73301', 'USA', '512-555-0300', 'support@greenenergy.com',
'www.greenenergy.com'),
      ('EduLearn Inc', 'Education', '101 Learning St', 'Boston', 'MA',
'02108', 'USA', '617-555-0400', 'hello@edulearn.com', 'www.edulearn.com'),
      ('FinSecure LLC', 'Finance', '202 Money Rd', 'New York', 'NY', '10005',
'USA', '212-555-0500', 'services@finsecure.com', 'www.finsecure.com'),
      ('AutoDrive Motors', 'Automotive', '303 Car Ln', 'Detroit', 'MI',
'48226', 'USA', '313-555-0600', 'sales@autodrive.com', 'www.autodrive.com'),
      ('Fashion Forward', 'Retail', '404 Style Blvd', 'New York', 'NY',
'10001', 'USA', '212-555-0700', 'info@fashionforward.com',
'www.fashionforward.com'),
      ('Foodies Delight', 'Food & Beverage', '505 Gourmet Ave', 'Chicago',
'IL', '60616', 'USA', '312-555-0800', 'order@foodiesdelight.com',
'www.foodiesdelight.com'),
      ('BuildRight Construction', 'Construction', '606 Builder St',
'Seattle', 'WA', '98101', 'USA', '206-555-0900', 'contact@buildright.com',
'www.buildright.com'),
      ('TravelQuest', 'Travel', '707 Adventure Rd', 'Orlando', 'FL', '32801',
'USA', '407-555-1000', 'support@travelquest.com', 'www.travelquest.com');


INSERT INTO `MyCMS`.`Contact`
```

```
      (`UserID`, `FirstName`, `LastName`, `DateOfBirth`, `Gender`, `Street`,
`City`, `State`, `Zip`, `Country`, `CompanyID`)
VALUES
      (1, 'Alice', 'Doe', '1980-02-01', 'F', '111 Main St', 'New York', 'NY',
'10001', 'USA',  1),
      (2, 'Bob', 'Smith', '1975-07-10', 'M', '222 Oak St', 'Los Angeles',
'CA', '90001', 'USA',  2),
      (3, 'Charlie', 'Brown', '1990-03-15', 'M', '333 Pine St', 'Chicago',
'IL', '60601', 'USA',  3),
      (4, 'Diana', 'Prince', '1988-04-20', 'F', '444 Cedar St', 'Boston',
'MA', '02108', 'USA',  4),
      (5, 'Edward', 'Norton', '1982-12-05', 'M', '555 Maple Ave', 'Detroit',
'MI', '48226', 'USA',  5),
      (6, 'Fiona', 'Apple', '1995-09-09', 'F', '666 Birch Rd', 'Austin',
'TX', '73301', 'USA',  6),
      (7, 'George', 'Clooney', '1961-05-06', 'M', '777 Walnut St', 'San
Francisco', 'CA', '94107', 'USA',  7),
      (8, 'Helen', 'Mirren', '1945-07-26', 'F', '888 Spruce St', 'Seattle',
'WA', '98101', 'USA',  8),
      (9, 'Ian', 'McKellen', '1939-05-25', 'M', '999 Elm St', 'Chicago',
'IL', '60616', 'USA',  9),
      (10, 'Jane', 'Doe', '1985-11-30', 'Other', '1010 Ash Ave', 'Orlando',
'FL', '32801', 'USA', 10);

INSERT INTO `MyCMS`.`ContactNickname`
      (`ContactID`, `Nickname`)
VALUES
      (1, 'Ally'),
      (1, 'Ali'),
      (2, 'Bobby'),
      (3, 'Chuck'),
      (4, 'Di'),
      (4, 'D'),
      (5, 'Eddie'),
      (5, 'Ed'),
      (6, 'Fi'),
      (7, 'Geo'),
      (8, 'Leni'),
      (9, 'Mack'),
      (10, 'Janie');

INSERT INTO `MyCMS`.`Phone`
      (`ContactID`, `PhoneNumber`, `PhoneType`, `CountryCode`)
```

```
VALUES
      (1, '603-751-2124', 'Cell', '+1'),
      (1, '603-750-0027', 'Home', '+1'),
      (2, '651-751-2125', 'Home', '+1'),
      (3, '521-751-2126', 'Work', '+1'),
      (4, '501-751-2127', 'Fax', '+1'),
      (5, '401-751-2128', 'Cell', '+1'),
      (6, '307-751-2129', 'Home', '+1'),
      (7, '701-751-2130', 'Work', '+1'),
      (8, '201-751-2131', 'Fax', '+1'),
      (9, '410-751-2132', 'Cell', '+1'),
      (10, '710-751-2133', 'Home', '+1');

INSERT INTO `MyCMS`.`Email`
      (`ContactID`, `EmailAddress`, `EmailType`, `IsPrimary`)
VALUES
      (1, 'contact1@example.com', 'Personal', 1),
      (2, 'contact2@example.com', 'Work', 1),
      (3, 'contact3@example.com', 'Other', 1),
      (4, 'contact4@example.com', 'Personal', 1),
      (5, 'contact5@example.com', 'Work', 1),
      (6, 'contact6@example.com', 'Other', 1),
      (7, 'contact7@example.com', 'Personal', 1),
      (8, 'contact8@example.com', 'Work', 1),
      (9, 'contact9@example.com', 'Other', 1),
      (10, 'contact10@example.com', 'Personal', 1);

INSERT INTO `MyCMS`.`Picture`
      (`ImagePath`, `UploadedDate`, `Description`, `Contact_ContactID`)
VALUES
      ('images/contact1_pic1.jpg', '2025-02-21 10:00:00', 'Profile picture',
1),
      ('images/contact2_pic1.jpg', '2025-02-20 09:30:00', 'Office event', 2),
      ('images/contact3_pic1.jpg', '2025-02-19 12:00:00', 'Profile picture',
3),
      ('images/contact4_pic1.jpg', '2025-02-18 14:00:00', 'Profile picture',
4),
      ('images/contact5_pic1.jpg', '2025-02-17 10:30:00', 'Profile picture',
5),
      ('images/contact6_pic1.jpg', '2025-02-16 16:00:00', 'Profile picture',
6),
      ('images/contact7_pic1.jpg', '2025-02-15 09:15:00', 'Profile picture',
7),
```

```
      ('images/contact8_pic1.jpg', '2025-02-14 11:30:00', 'Profile picture',
8),
      ('images/contact9_pic1.jpg', '2025-02-13 08:45:00', 'Profile picture',
9),
      ('images/contact10_pic1.jpg', '2025-02-12 13:30:00', 'Profile picture',
10);

INSERT INTO `MyCMS`.`ActivityType`
      (`TypeName`, `Description`, `CreatedAt`, `UpdatedAt`)
VALUES
      ('Login',   'User logged in',       '2025-02-22 08:00:00', '2025-02-22
08:00:00'),
      ('Logout',   'User logged out',           '2025-02-22 08:05:00',
'2025-02-22 08:05:00'),
      ('Create',   'Created a record',          '2025-02-22 09:00:00',
'2025-02-22 09:00:00'),
      ('Update',   'Updated a record',          '2025-02-22 10:00:00',
'2025-02-22 10:00:00'),
      ('Delete',   'Deleted a record',          '2025-02-22 11:00:00',
'2025-02-22 11:00:00'),
      ('View',    'Viewed a record',      '2025-02-22 12:00:00', '2025-02-22
12:00:00'),
      ('Upload',   'Uploaded a file',           '2025-02-22 14:00:00',
'2025-02-22 14:00:00');

INSERT INTO `MyCMS`.`ActivityLog`
      (`UserID`, `ActivityTypeID`, `Timestamp`, `Description`)
VALUES
      (1, 1, '2025-02-22 08:10:00', 'User logged in'),
      (1, 3, '2025-02-22 08:15:00', 'Created a new contact'),
      (2, 1, '2025-02-22 09:00:00', 'User logged in'),
      (2, 6, '2025-02-22 09:10:00', 'Viewed a company profile'),
      (3, 2, '2025-02-22 10:30:00', 'User logged out'),
      (4, 4, '2025-02-22 11:00:00', 'Updated user settings'),
      (5, 5, '2025-02-12 12:00:00', 'Deleted a contact record'),
      (7, 7, '2025-02-22 13:45:00', 'Uploaded a document'),
      (8, 7, '2025-02-25 13:45:00', 'Uploaded a document'),
      (10, 1, '2025-02-22 16:30:00', 'User logged in');

INSERT INTO `MyCMS`.`ContactNote`
      (`ContactID`, `NoteTitle`, `NoteText`, `CreatedDate`, `LastModified`,
`Status`)
VALUES
```

```
      (1, 'Follow-up Meeting', 'Schedule a follow-up meeting with Alice.',
'2025-02-20 08:00:00', '2025-02-21 09:00:00', 'ACTIVE'),
      (1, 'Urgent Request', 'Alice requested additional documentation.',
'2025-02-21 10:15:00', '2025-02-21 11:00:00', 'ACTIVE'),
      (2, 'Project Update', 'Bob needs an update on project status.',
'2025-02-19 12:30:00', '2025-02-19 13:45:00', 'ACTIVE'),
      (3, 'Client Inquiry', 'Charlie asked about new service offerings.',
'2025-02-18 09:45:00', '2025-02-18 10:20:00', 'ACTIVE'),
      (4, 'Invoice Issue', 'Diana reported an issue with an invoice.',
'2025-02-17 11:10:00', '2025-02-17 12:00:00', 'ACTIVE'),
      (5, 'Training Session', 'Edward signed up for the training program.',
'2025-02-16 14:00:00', '2025-02-16 14:30:00', 'ACTIVE'),
      (6, 'Follow-up Email', 'Fiona needs a follow-up email.', '2025-02-15
08:15:00', '2025-02-15 08:45:00', 'ACTIVE'),
      (7, 'Meeting Notes', 'George attended a company meeting.', '2025-02-14
10:30:00', '2025-02-14 11:00:00', 'ACTIVE'),
      (8, 'Documentation Request', 'Helen requested access to documents.',
'2025-02-13 09:20:00', '2025-02-13 10:00:00', 'ACTIVE'),
      (9, 'Feedback Received', 'Ian provided feedback on the new policy.',
'2025-02-12 16:45:00', '2025-02-12 17:30:00', 'ACTIVE'),
      (10, 'Contract Review', 'Jane is reviewing the new contract terms.',
'2025-02-11 15:30:00', '2025-02-11 16:00:00', 'ACTIVE'),
      (10, 'Updated Contact Info', 'Jane updated her phone number.',
'2025-02-10 14:20:00', '2025-02-10 15:00:00', 'ACTIVE');

INSERT INTO `MyCMS`.`GroupTable`
      (`GroupName`, `Description`, `CreatedDate`, `Status`)
VALUES
      ('Business Partners', 'Contacts related to business partnerships.',
'2025-02-01 08:00:00', 'ACTIVE'),
      ('Clients', 'Clients of the company.', '2025-02-02 09:00:00',
'ACTIVE'),
      ('VIP Clients', 'High-value clients with special privileges.',
'2025-02-03 10:00:00', 'ACTIVE'),
      ('Suppliers', 'Suppliers and vendors for the company.', '2025-02-04
11:00:00', 'ACTIVE'),
      ('Employees', 'Internal employees and staff.', '2025-02-05 12:00:00',
'ACTIVE'),
      ('Investors', 'People who have invested in the company.', '2025-02-06
13:00:00', 'ACTIVE'),
      ('Consultants', 'External consultants and advisors.', '2025-02-07
14:00:00', 'ACTIVE'),
      ('Marketing Team', 'Marketing team members.', '2025-02-08 15:00:00',
```

```
'ACTIVE'),
      ('Developers', 'Software development team.', '2025-02-09 16:00:00',
'ACTIVE'),
      ('Board Members', 'Board of directors and executives.', '2025-02-10
17:00:00', 'ACTIVE');

INSERT INTO `MyCMS`.`ContactGroupMapping`
      (`ContactID`, `GroupID`, `DateAdded`, `MappingStatus`)
VALUES
      (1, 1, '2025-02-11 08:00:00', 'ACTIVE'),
      (1, 3, '2025-02-12 08:30:00', 'ACTIVE'),
      (2, 2, '2025-02-13 09:00:00', 'ACTIVE'),
      (3, 5, '2025-02-14 09:30:00', 'ACTIVE'),
      (4, 4, '2025-02-15 10:00:00', 'ACTIVE'),
      (5, 6, '2025-02-16 10:30:00', 'ACTIVE'),
      (6, 7, '2025-02-17 11:00:00', 'ACTIVE'),
      (7, 8, '2025-02-18 11:30:00', 'ACTIVE'),
      (8, 9, '2025-02-19 12:00:00', 'ACTIVE'),
      (9, 10, '2025-02-20 12:30:00', 'ACTIVE'),
      (10, 2, '2025-02-21 13:00:00', 'ACTIVE'),
      (10, 6, '2025-02-22 13:30:00', 'ACTIVE'),
      (2, 4, '2025-02-23 14:00:00', 'ACTIVE'),
      (3, 6, '2025-02-24 14:30:00', 'ACTIVE'),
      (5, 8, '2025-02-25 15:00:00', 'ACTIVE');

INSERT INTO `MyCMS`.`Tag`
      (`TagName`, `Description`, `CreatedDate`, `TagColor`)
VALUES
      ('VIP', 'High-value contact.', '2025-02-01 08:00:00', 'Gold'),
      ('Lead', 'Potential customer or client.', '2025-02-02 09:00:00',
'Blue'),
      ('Supplier', 'Supplier contact.', '2025-02-03 10:00:00', 'Green'),
      ('Investor', 'Investors in the company.', '2025-02-04 11:00:00',
'Silver'),
      ('Employee', 'Internal employee.', '2025-02-05 12:00:00', 'Red'),
      ('Consultant', 'External consultant or advisor.', '2025-02-06
13:00:00', 'Orange'),
      ('Marketing', 'Marketing-related contact.', '2025-02-07 14:00:00',
'Purple'),
      ('Developer', 'Software developer contact.', '2025-02-08 15:00:00',
'Teal'),
      ('Board Member', 'Part of the board of directors.', '2025-02-09
16:00:00', 'Brown'),
```

```
      ('Customer', 'Current paying customer.', '2025-02-10 17:00:00',
'Yellow');

INSERT INTO `MyCMS`.`ContactTagMapping`
      (`ContactID`, `TagID`, `CreatedTime`, `Status`)
VALUES
      (1, 1, '2025-02-11 08:00:00', 'ACTIVE'),
      (1, 2, '2025-02-12 08:30:00', 'ACTIVE'),
      (2, 3, '2025-02-13 09:00:00', 'ACTIVE'),
      (3, 5, '2025-02-14 09:30:00', 'ACTIVE'),
      (4, 4, '2025-02-15 10:00:00', 'ACTIVE'),
      (5, 6, '2025-02-16 10:30:00', 'ACTIVE'),
      (6, 7, '2025-02-17 11:00:00', 'ACTIVE'),
      (7, 8, '2025-02-18 11:30:00', 'ACTIVE'),
      (8, 9, '2025-02-19 12:00:00', 'ACTIVE'),
      (9, 10, '2025-02-20 12:30:00', 'ACTIVE'),
      (10, 2, '2025-02-21 13:00:00', 'ACTIVE'),
      (10, 4, '2025-02-22 13:30:00', 'ACTIVE'),
      (2, 5, '2025-02-23 14:00:00', 'ACTIVE'),
      (3, 6, '2025-02-24 14:30:00', 'ACTIVE'),
      (5, 7, '2025-02-25 15:00:00', 'ACTIVE');

INSERT INTO `MyCMS`.`Meeting`
      (`UserID`, `MeetingTitle`, `Description`, `Location`, `MeetingDate`,
`CreatedDate`)
VALUES
      (1, 'Project Kickoff', 'Initial meeting to discuss project details.',
'Conference Room A', '2025-03-01 10:00:00', '2025-02-20 08:00:00'),
      (2, 'Marketing Strategy', 'Discussing marketing plans for Q2.', 'Online
- Zoom', '2025-03-02 14:00:00', '2025-02-21 09:00:00'),
      (3, 'Investor Update', 'Quarterly investor update meeting.',
'Boardroom', '2025-03-03 16:00:00', '2025-02-22 10:00:00'),
      (4, 'Tech Team Sync', 'Weekly development team stand-up.', 'Engineering
Lab', '2025-03-04 09:30:00', '2025-02-23 11:00:00'),
      (5, 'Customer Feedback', 'Meeting with customers to collect feedback.',
'Client Office', '2025-03-05 13:00:00', '2025-02-24 12:00:00'),
      (6, 'Partnership Discussion', 'Discussing potential business
partnerships.', 'Main Office', '2025-03-06 11:00:00', '2025-02-25 13:00:00'),
      (7, 'Annual General Meeting', 'Company-wide annual update.', 'Main
Auditorium', '2025-03-07 10:00:00', '2025-02-26 14:00:00'),
      (8, 'Training Session', 'Employee training on new software.', 'Training
Room 2', '2025-03-08 15:00:00', '2025-02-27 15:00:00'),
      (9, 'Supplier Negotiation', 'Negotiation with key suppliers.',
```

```
'Supplier HQ', '2025-03-09 12:30:00', '2025-02-28 16:00:00'),
      (10, 'Board Meeting', 'Strategic planning meeting for executives.',
'Boardroom', '2025-03-10 17:00:00', '2025-03-29 17:00:00');

INSERT INTO `MyCMS`.`MeetingContactMapping`
      (`MeetingID`, `ContactID`, `InvitationStatus`, `ResponseDate`)
VALUES
      (1, 1, 'Accepted', '2025-02-25 08:30:00'),
      (1, 2, 'Pending', NULL),
      (2, 3, 'Accepted', '2025-02-26 09:15:00'),
      (2, 4, 'Declined', '2025-02-26 10:00:00'),
      (3, 5, 'Accepted', '2025-02-27 11:20:00'),
      (3, 6, 'Pending', NULL),
      (4, 7, 'Accepted', '2025-02-28 12:45:00'),
      (5, 8, 'Pending', NULL),
      (5, 9, 'Accepted', '2025-03-01 14:10:00'),
      (6, 10, 'Accepted', '2025-03-02 15:30:00'),
      (6, 1, 'Pending', NULL),
      (7, 2, 'Accepted', '2025-03-03 16:00:00'),
      (8, 3, 'Accepted', '2025-03-04 09:30:00'),
      (8, 4, 'Declined', '2025-03-04 10:20:00'),
      (9, 5, 'Accepted', '2025-03-05 11:45:00'),
      (9, 6, 'Pending', NULL),
      (10, 7, 'Accepted', '2025-03-06 13:10:00'),
      (10, 8, 'Accepted', '2025-03-06 14:30:00');

SET FOREIGN_KEY_CHECKS = 1;  -- Re-enable foreign key checks
```

Another way to import data is to use the LOAD DATA INLINE command, for this we need to use CSVs files with the data, and we have to use one file per table. To import data this way we should do it in the console using commands like the next:

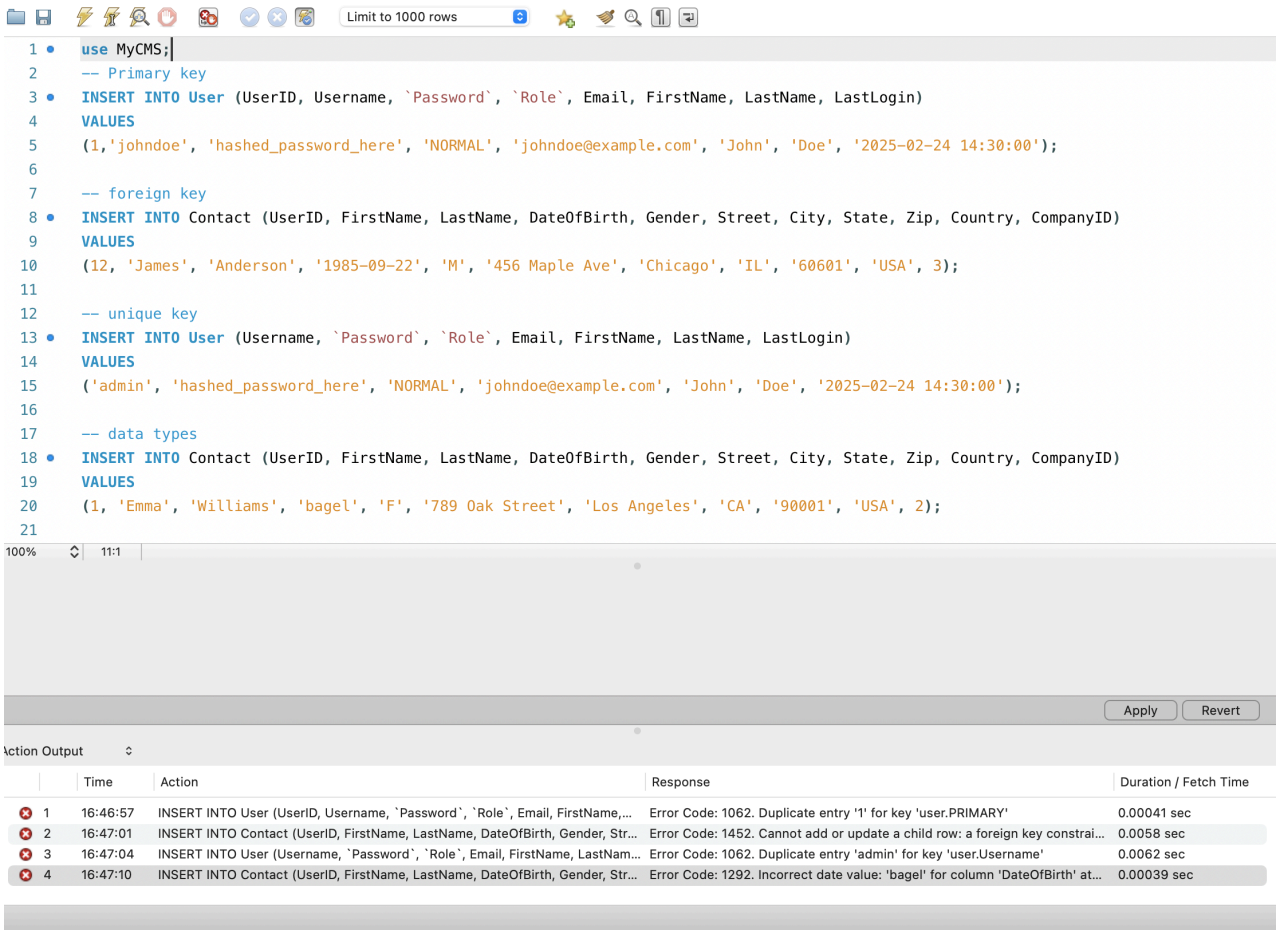
```
SET FOREIGN_KEY_CHECKS=0;

LOAD DATA INFILE './user.csv'
INTO TABLE `MyCMS`.`User`
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';

SET FOREIGN_KEY_CHECKS=1;
```

And like this for all every table. We'll attach all the CSVs files in a compressed file.

Retrieving data



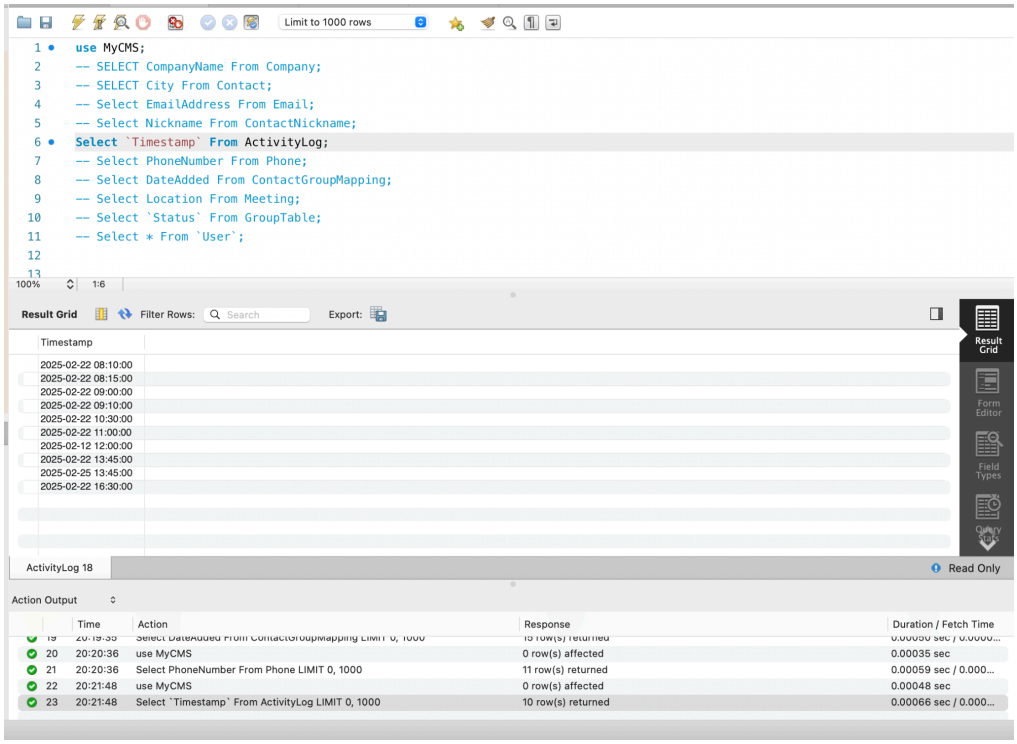
Above shows:

The error of a Primary Key by trying to add a UserID that is already being used.

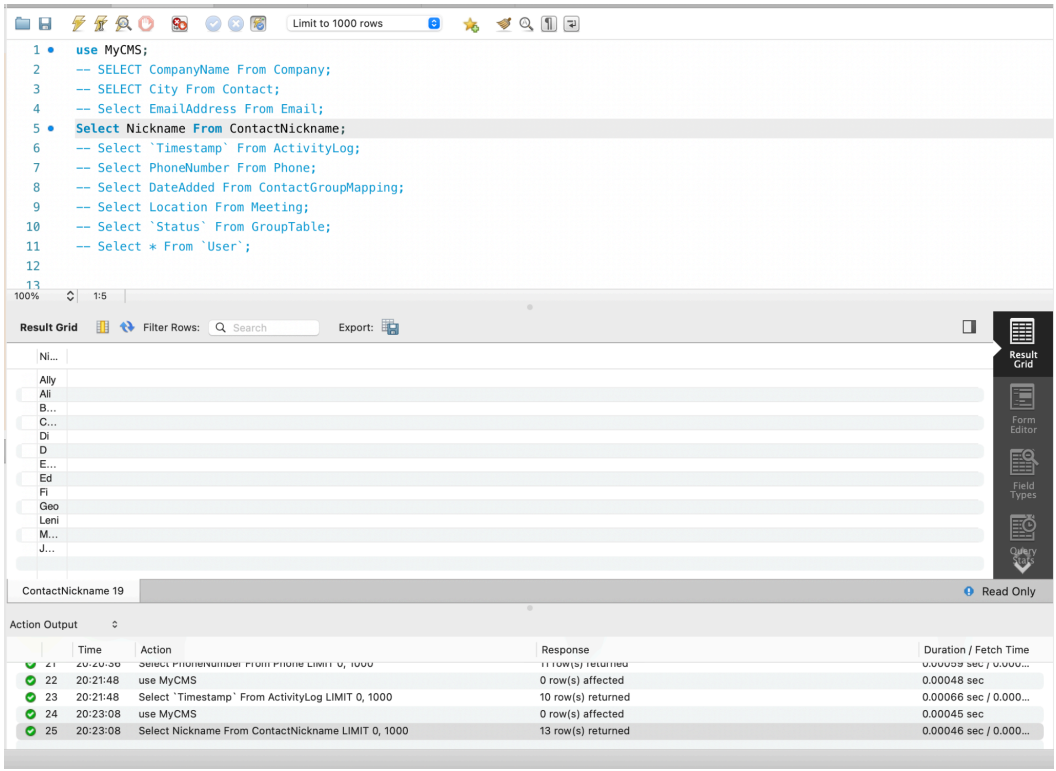
The error of a Foreign Key by trying to add information into the Contact table with a UserID that is not established in the User table.

The error of a Unique Key by trying to a Username that is already taken.

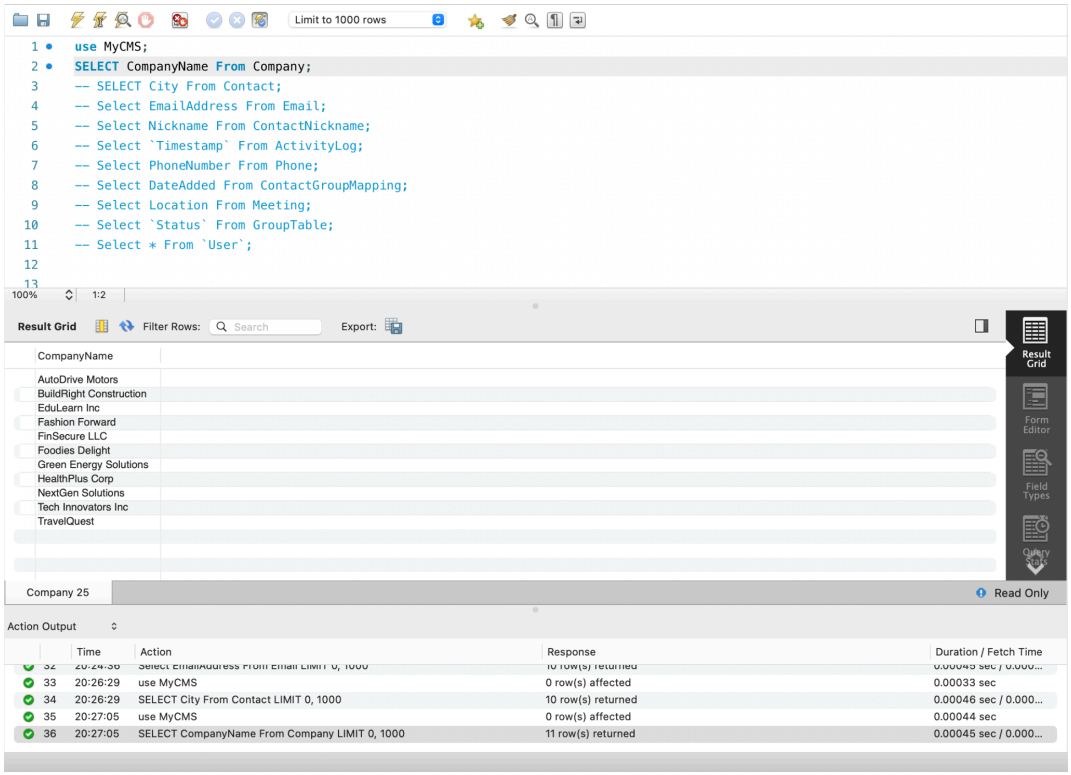
The error of Data Types by trying to add a DateOfBirth that is not a number value.



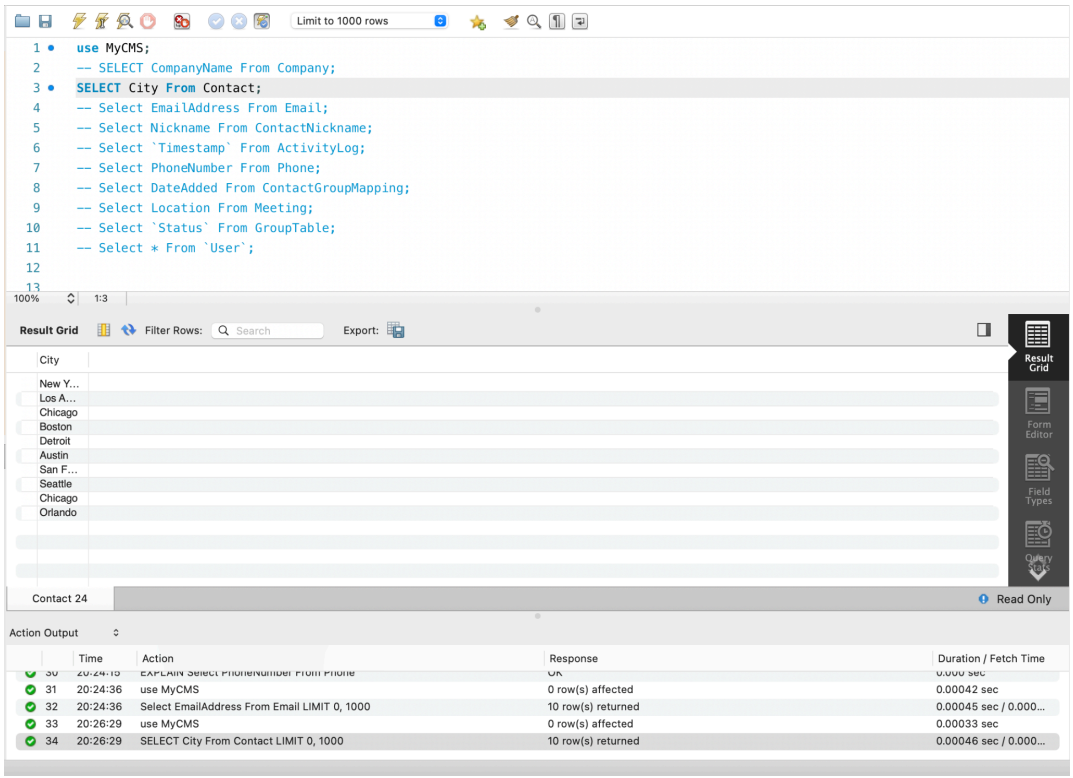
Above shows the selection of Timestamp information from the ActivityLog table.



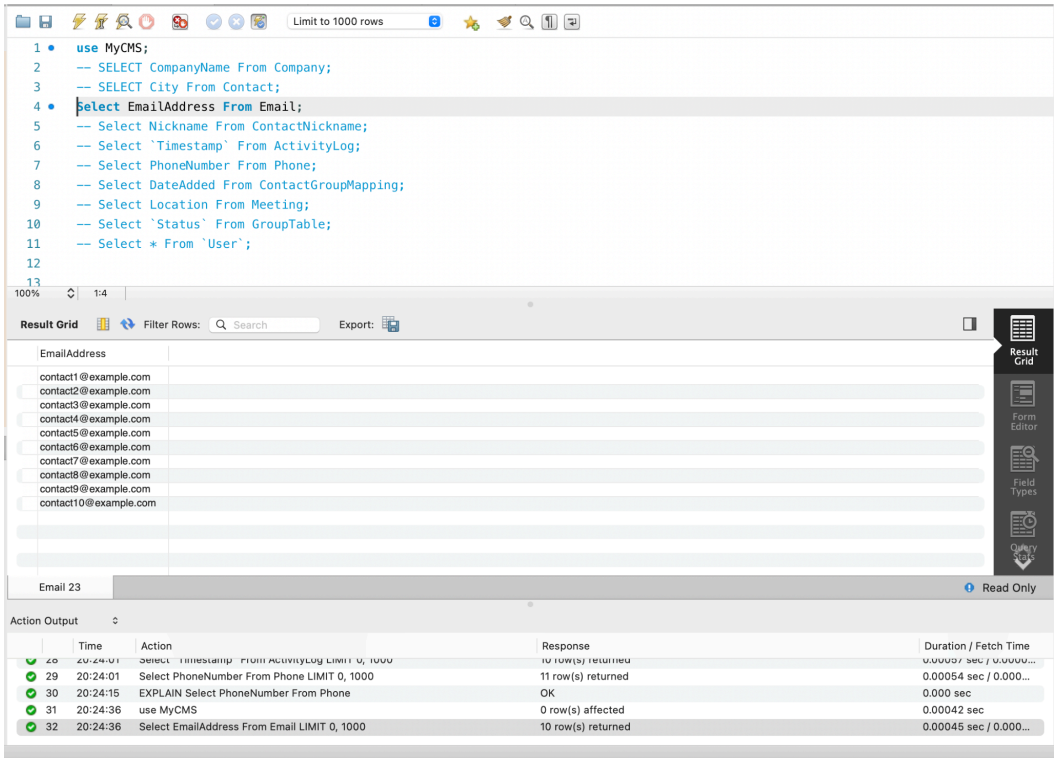
Above shows the selection of Nickname from the ContactNickname table.



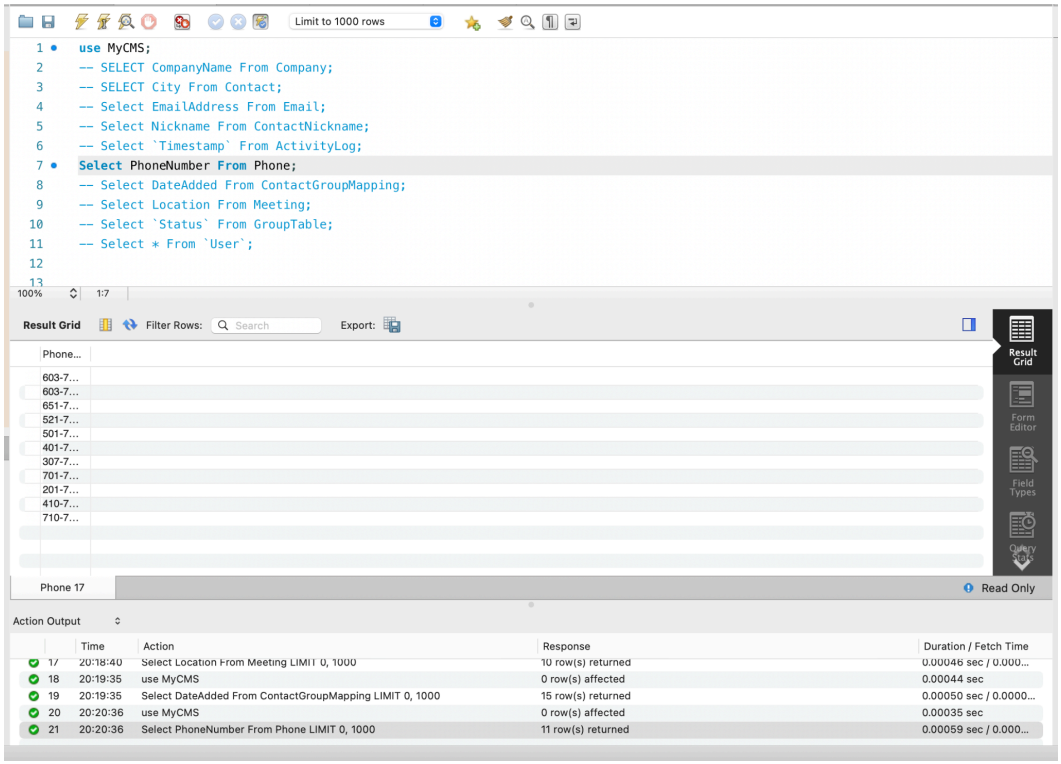
Above shows the selection of CompanyName from the Company Table.



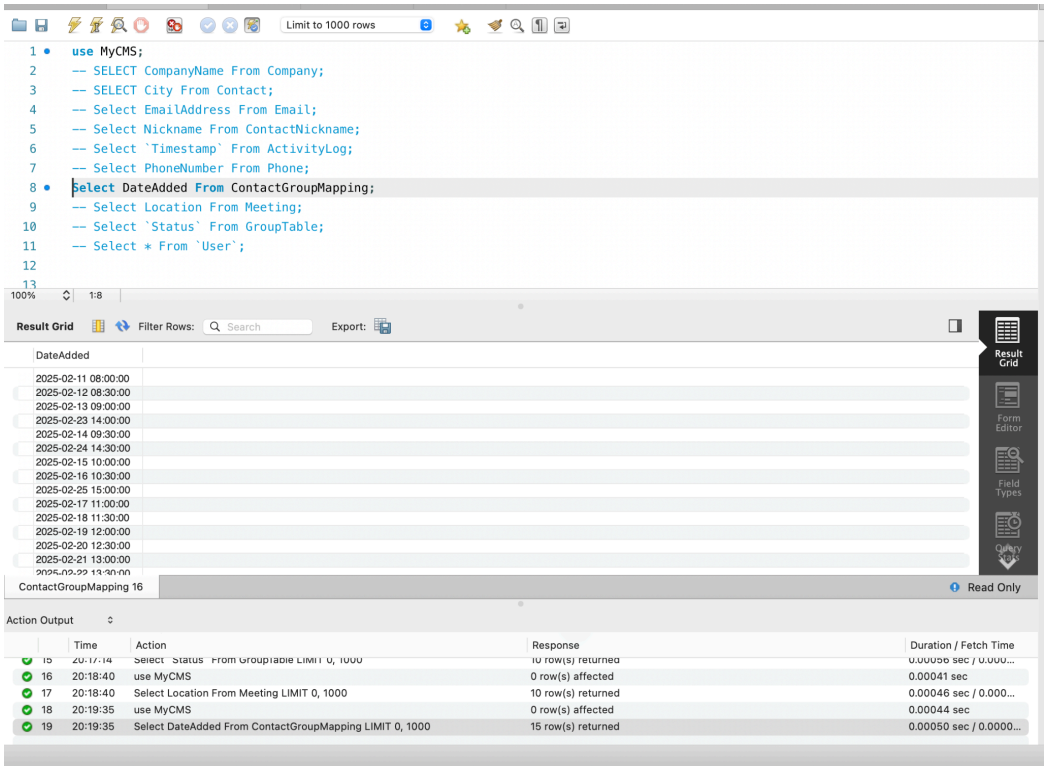
Above shows the selection of City from the Contact Table.



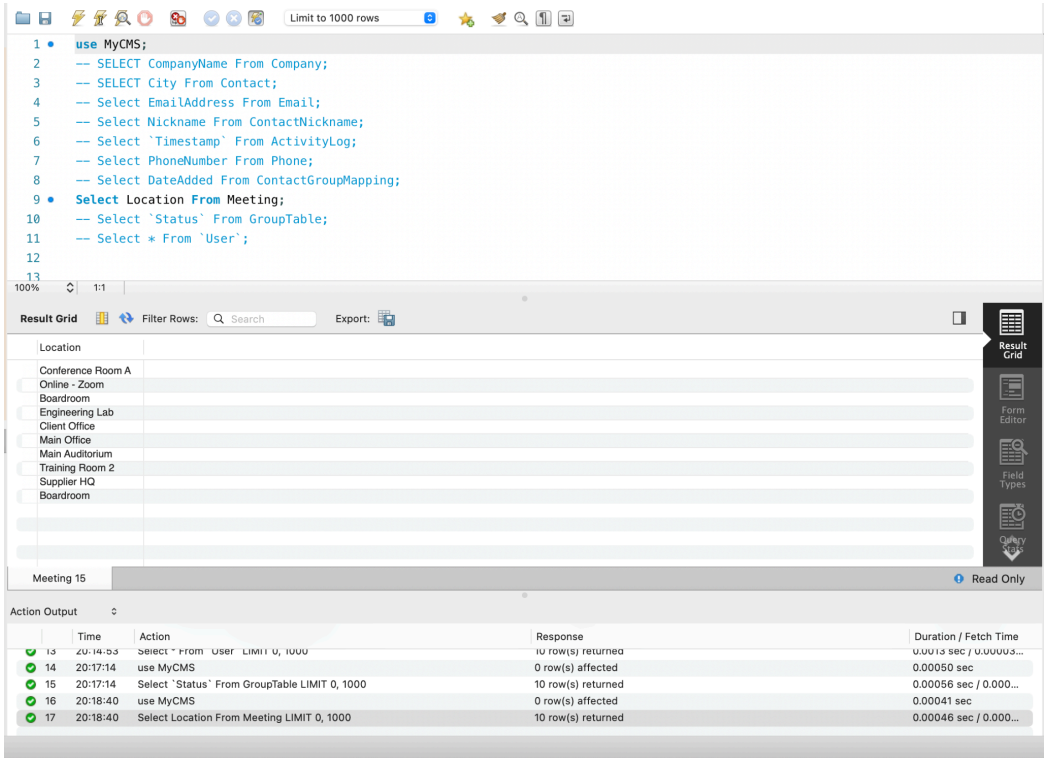
Above shows the selection of EmailAddress from the Email table.



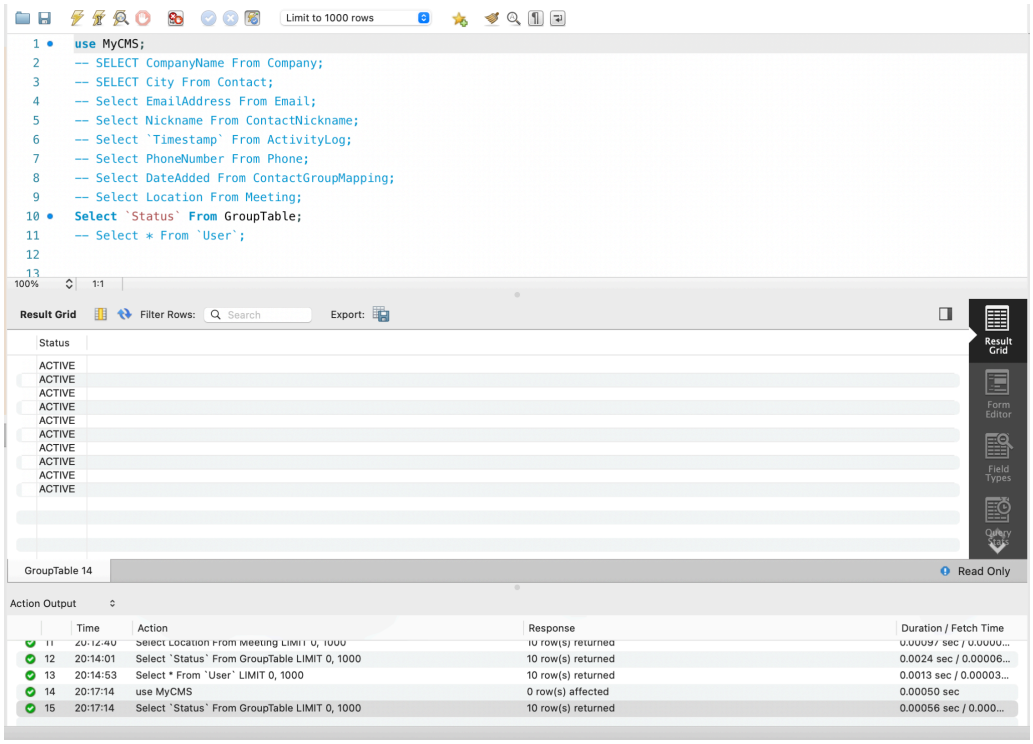
Above shows the selection of PhoneNumber from the Phone table.



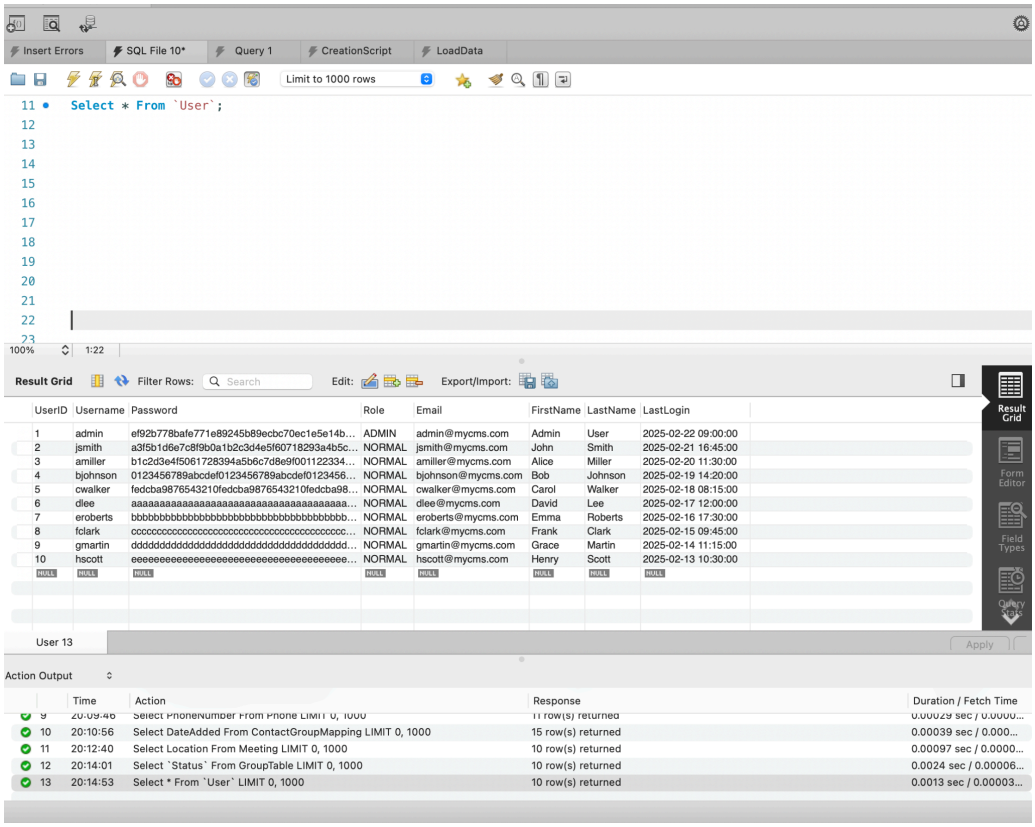
Above shows the selection of DateAdded from the ContactGroupMapping table.



Above shows the selection of the Location from the Meeting table.



Above shows the selection of Status from the GroupTable table.



Above shows the selection of all information from the User table.

Manipulating data

```
UPDATE User
SET Email = 'newemail2025@example.com'
WHERE UserID = 1;

select * from User;
```

Using the update statement shown above to update a value in User Table, we change this output:

UserID	Username	Password	Role	Email	FirstName	LastName	LastLogin
1	johndoe	hashedpassword	NORMAL	newemail@example.com	John	Doe	2025-02-21 16:45:00
2	admin	ef92b778bafe771e89245b89ecbc70ec1e5e14b...	ADMIN	admin@mycms.com	Admin	User	2025-02-22 09:00:00
3	jsmith	a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5...	NORMAL	jsmith@mycms.com	John	Smith	2025-02-21 16:45:00
4	amiller	b1c2d3e4f5061728394a5b6c7d8e9f001122334...	NORMAL	amiller@mycms.com	Alice	Miller	2025-02-20 11:30:00
5	bjohnson	0123456789abcdef0123456789abcdef0123456...	NORMAL	bjohnson@mycms.com	Bob	Johnson	2025-02-19 14:20:00
6	cwalker	fedcba9876543210fedcba9876543210fedcba9...	NORMAL	cwalker@mycms.com	Carol	Walker	2025-02-18 08:15:00
7	dlee	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	NORMAL	dlee@mycms.com	David	Lee	2025-02-17 12:00:00
8	eroberts	bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb...	NORMAL	eroberts@mycms.com	Emma	Roberts	2025-02-16 17:30:00
9	fclark	ccccccccccccccccccccccccccccccccccccccccc...	NORMAL	fclark@mycms.com	Frank	Clark	2025-02-15 09:45:00
10	gmartin	ddddddddddddddddddddddddddddddddddddddd...	NORMAL	gmartin@mycms.com	Grace	Martin	2025-02-14 11:15:00
11	hscott	eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee...	NORMAL	hscott@mycms.com	Henry	Scott	2025-02-13 10:30:00

Into this output:

UserID	Username	Password	Role	Email	FirstName	LastName	LastLogin
1	johndoe	hashedpassword	NORMAL	newemail2025@example.com	John	Doe	2025-02-21 16:45:00
2	admin	ef92b778bafe771e89245b89ecbc70ec1e5e14b...	ADMIN	admin@mycms.com	Admin	User	2025-02-22 09:00:00
3	jsmith	a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5...	NORMAL	jsmith@mycms.com	John	Smith	2025-02-21 16:45:00
4	amiller	b1c2d3e4f5061728394a5b6c7d8e9f001122334...	NORMAL	amiller@mycms.com	Alice	Miller	2025-02-20 11:30:00
5	bjohnson	0123456789abcdef0123456789abcdef0123456...	NORMAL	bjohnson@mycms.com	Bob	Johnson	2025-02-19 14:20:00
6	cwalker	fedcba9876543210fedcba9876543210fedcba9...	NORMAL	cwalker@mycms.com	Carol	Walker	2025-02-18 08:15:00
7	dlee	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	NORMAL	dlee@mycms.com	David	Lee	2025-02-17 12:00:00
8	eroberts	bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb...	NORMAL	eroberts@mycms.com	Emma	Roberts	2025-02-16 17:30:00
9	fclark	ccccccccccccccccccccccccccccccccccccccccc...	NORMAL	fclark@mycms.com	Frank	Clark	2025-02-15 09:45:00
10	gmartin	ddddddddddddddddddddddddddddddddddddddd...	NORMAL	gmartin@mycms.com	Grace	Martin	2025-02-14 11:15:00
11	hscott	eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee...	NORMAL	hscott@mycms.com	Henry	Scott	2025-02-13 10:30:00

Using an Insert into statements with ‘On Duplicate Key’ as shown below:

```
INSERT INTO User (`UserID`, `Username`, `Password`, `Role`, `Email`, `FirstName`, `LastName`, `LastLogin`)
VALUES (1, 'johndoe', 'hashedpassword', 'NORMAL', 'newemail@example.com', 'John', 'Doe', '2025-02-21 16:45:00')
ON DUPLICATE KEY UPDATE
`Username` = concat(`Username`, '_' , `LastActivity`);

select * from User;
```

We further change the results as seen here:

UserID	Username	Password	Role	Email	FirstName	LastName	LastLogin
1	johndoe_2025-02-24 16:02:41	hashedpassword	NORMAL	newemail2025@example.com	John	Doe	2025-02-21 16:45:00
2	admin	ef92b778bafe771e89245b89ecbc70ec1e5e14b...	ADMIN	admin@mycms.com	Admin	User	2025-02-22 09:00:00
3	jsmith	a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5...	NORMAL	jsmith@mycms.com	John	Smith	2025-02-21 16:45:00
4	amiller	b1c2d3e4f5061728394a5b6c7d8e9f001122334...	NORMAL	amiller@mycms.com	Alice	Miller	2025-02-20 11:30:00
5	bjohnson	0123456789abcdef0123456789abcdef0123456...	NORMAL	bjohnson@mycms.com	Bob	Johnson	2025-02-19 14:20:00
6	cwalker	fedcba9876543210fedcba9876543210fedcba9...	NORMAL	cwalker@mycms.com	Carol	Walker	2025-02-18 08:15:00
7	dlee	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	NORMAL	dlee@mycms.com	David	Lee	2025-02-17 12:00:00
8	eroberts	bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb...	NORMAL	eroberts@mycms.com	Emma	Roberts	2025-02-16 17:30:00
9	fclark	cccccccccccccccccccccccccccccccccccccccccccccccc...	NORMAL	fclark@mycms.com	Frank	Clark	2025-02-15 09:45:00
10	gmartin	ddddddddddddddddddddddddddddddddddddddddd...	NORMAL	gmartin@mycms.com	Grace	Martin	2025-02-14 11:15:00
11	hscott	eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee...	NORMAL	hscott@mycms.com	Henry	Scott	2025-02-13 10:30:00

Using an Alter Table statement to add LastActivity attribute of type TIMESTAMP as shown below:

```
ALTER TABLE User
ADD COLUMN LastActivity TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;


select * from User;
```

We get these results:

UserID	Username	Password	Role	Email	FirstName	LastName	LastLogin	LastActivity
1	johndoe_2025-02-24 16:02:41	hashedpassword	NORMAL	newemail2025@example.com	John	Doe	2025-02-21 16:45:00	2025-02-24 16:05:43
2	admin	ef92b778bafe771e89245b89ecbc70ec1e5e14b...	ADMIN	admin@mycms.com	Admin	User	2025-02-22 09:00:00	2025-02-24 14:47:49
3	jsmith	a3f5b1d6e7c8f9b0a1b2c3d4e5f60718293a4b5...	NORMAL	jsmith@mycms.com	John	Smith	2025-02-21 16:45:00	2025-02-24 14:47:49
4	amiller	b1c2d3e4f5061728394a5b6c7d8e9f001122334...	NORMAL	amiller@mycms.com	Alice	Miller	2025-02-20 11:30:00	2025-02-24 14:47:49
5	bjohnson	0123456789abcdef0123456789abcdef0123456...	NORMAL	bjohnson@mycms.com	Bob	Johnson	2025-02-19 14:20:00	2025-02-24 14:47:49
6	cwalker	fedcba9876543210fedcba9876543210fedcba9...	NORMAL	cwalker@mycms.com	Carol	Walker	2025-02-18 08:15:00	2025-02-24 14:47:49
7	dlee	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	NORMAL	dlee@mycms.com	David	Lee	2025-02-17 12:00:00	2025-02-24 14:47:49
8	eroberts	bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb...	NORMAL	eroberts@mycms.com	Emma	Roberts	2025-02-16 17:30:00	2025-02-24 14:47:49
9	fclark	cccccccccccccccccccccccccccccccccccccccccccccccc...	NORMAL	fclark@mycms.com	Frank	Clark	2025-02-15 09:45:00	2025-02-24 14:47:49
10	gmartin	ddddddddddddddddddddddddddddddddddddddddd...	NORMAL	gmartin@mycms.com	Grace	Martin	2025-02-14 11:15:00	2025-02-24 14:47:49
11	hscott	eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee...	NORMAL	hscott@mycms.com	Henry	Scott	2025-02-13 10:30:00	2025-02-24 14:47:49

Database Optimization

Inserting Instances one at a time results in a completion time of:

	execution_time_seconds
▶	0.2979

Inserting Instances all at once results in a completion time of:

	execution_time_seconds
▶	0.0702

Time difference:
Inserting all data at once is 0.2277 seconds faster than inserting data online at a time. That makes bulk inserts 76.42% faster than single line inserts.

Normalization Check

1NF (First Normal Form)

This is in First Normal Form (1NF) because all data is stored in an organized, structured, and efficient way with unique characteristics and all attributes hold atomic values. Each attribute holds a single, indivisible value, which means that fields such as phone numbers and emails are not grouped together in a single column but are stored in separate rows. In addition, the database eliminates repeating groups ensuring there are no multi-values meaning each column contains a single type of data. By implementing First Normal Form, the Contact Management System avoids redundancy, improves data integrity, simplifies, and creates a more scalable platform for storing and retrieving contact information.

2NF (Second Normal Form)

This is in Second Normal Form (2NF) because we showed that it is in First Normal Form and the non-prime attributes that depend on a candidate key. This means that it has eliminated partial dependency which ensures that all non-key attributes depend on the primary key. This can be shown with the ContactTagMapping Table where Status and CreatedTime relate to all the ID keys. The benefits of having Second Normal Form are the same as having First Normal Form as well as setting a foundation for further normalization.

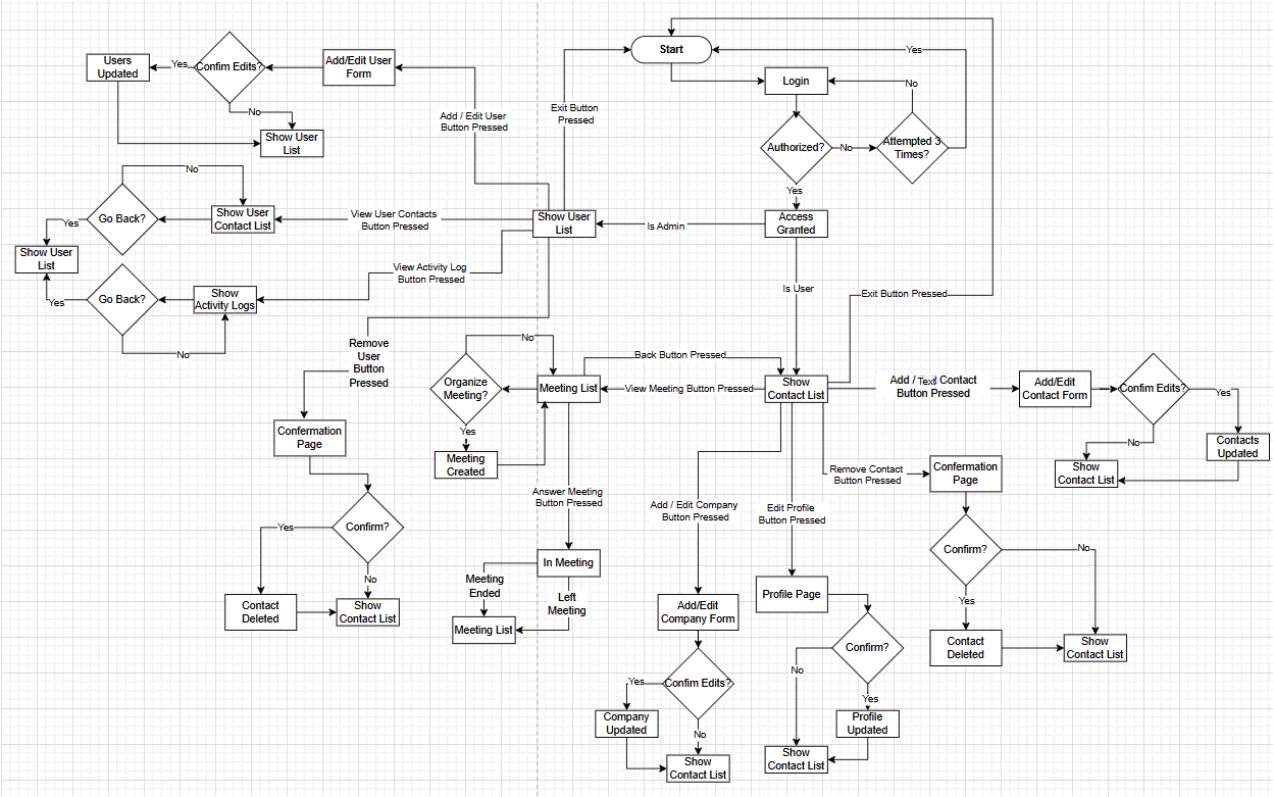
3NF (Third Normal Form)

This is Third Normal Form (3NF) because it is in Second Normal Form (2NF) as stated above and cannot have any transitive functional dependencies. A transitive functional dependency in a database occurs when an attribute in a table depends on another attribute, which in turn depends on the primary key, creating an indirect relationship where the first attribute is "transitively" dependent on the primary key. Our database does not have any of these transitive dependencies.

Application Development

User Experience Design

For our Interface design, we start with the login page. From there accesses are then granted and user or admin privileges are granted. Admins are shown a user list where they can add, edit, and remove users, view user contact lists, and view activity logs. Users are shown their contact list where they can add, edit, and remove contacts, edit their profile page and password, add and edit company information, as well as view and organise meetings.



Views' Implementation

The views that will be needed are the next:

1. Login page: A simple view based on the users table can provide login credentials and role information

```
CREATE OR REPLACE VIEW vw_login AS
SELECT
      UserID,
      Username,
      Password,
      Role
FROM users;
```

2. Admin User List: For listing users without exposing sensitive data.

```
CREATE OR REPLACE VIEW vw_users_list AS
```

```
SELECT
      UserID,
      Username,
      FirstName,
      LastName,
      Email,
      Role,
FROM users;
```

3. Admin User Add/Edit: this view have more detail, all the fields needed to add or edit a user, and it is an updatable view.

```
CREATE OR REPLACE VIEW vw_user_details AS
SELECT
      UserID,
      Username,
      Email,
      Role,
      FirstName,
      LastName,
FROM users;
```

4. Admin View User Contacts: to list all the contacts of the user:

```
CREATE OR REPLACE VIEW vw_admin_user_contacts AS
SELECT
      u.UserID,
      u.Username,
      c.ContactID,
      c.FirstName AS ContactFirstName,
      c.LastName AS ContactLastName,
      GROUP_CONCAT(DISTINCT CONCAT(p.PhoneNumber, ' (', p.PhoneType, ')')
SEPARATOR ', ') AS PhoneNumbers,
      GROUP_CONCAT(DISTINCT CONCAT(e.EmailAddress, ' (', e.EmailType, ')')
SEPARATOR ', ') AS EmailAddresses
FROM MyCMS.User u
JOIN MyCMS.Contact c ON u.UserID = c.UserID
LEFT JOIN MyCMS.Phone p ON c.ContactID = p.ContactID
LEFT JOIN MyCMS.Email e ON c.ContactID = e.ContactID
GROUP BY
      u.UserID,
```

```
      u.Username,
      c.ContactID,
      c.FirstName,
      c.LastName;
```

5. Activity Log: to show all the log activities.

```
CREATE OR REPLACE VIEW vw_activity_log AS
SELECT
      al.LogID,
      al.Timestamp,
      u.UserID,
      u.Username,
      u.Email,
      at.ActivityTypeID,
      at.TypeName,
      at.Description AS ActivityTypeDescription,
      al.Description AS LogDescription
FROM MyCMS.ActivityLog AS al
JOIN MyCMS.User AS u ON al.UserID = u.UserID
JOIN MyCMS.ActivityType AS at ON al.ActivityTypeID = at.ActivityTypeID;
```

6. Contacts List: this view will be used to show all the contacts of a user, this is not an updatable view as it uses joins and relations are not one-to-one.

```
CREATE OR REPLACE VIEW vw_contacts_list AS
SELECT
      c.ContactID,
      c.UserID,
      CONCAT(c.FirstName, ' ', c.LastName) AS FullName,
      c.DateOfBirth,
      c.Gender,
      c.Street,
      c.City,
      c.State,
      c.Zip,
      c.Country,
      c.CompanyID,
      GROUP_CONCAT(t.TagName SEPARATOR ', ') AS TagNames
FROM MyCMS.Contact c
LEFT JOIN MyCMS.ContactTagMapping ctm ON c.ContactID = ctm.ContactID
```

```
LEFT JOIN MyCMS.Tag t ON ctm.TagID = t.TagID
GROUP BY
      c.ContactID,
      c.UserID,
      c.FirstName,
      c.LastName,
      c.DateOfBirth,
      c.Gender,
      c.Street,
      c.City,
      c.State,
      c.Zip,
      c.Country,
      c.CompanyID;
```

7. Add/Edit Contact: a view directly based on the contacts table can be used to add or edit contact details, this is an updatable table.

```
CREATE OR REPLACE VIEW vw_contact_details AS
SELECT
      ContactID,
      UserID,
      FirstName,
      LastName,
      DateOfBirth,
      Gender,
      Street,
      City,
      State,
      Zip,
      Country,
      CompanyID
FROM MyCMS.Contact;
```

8. Company Add/Edit: for company details this view is also updatable.

```
CREATE OR REPLACE VIEW vw_company_details AS
SELECT
      CompanyID,
      CompanyName,
      Industry,
```

```
      Street,
      City,
      State,
      Zip,
      Country,
      Phone,
      Email,
      Website
FROM MyCMS.Company;
```

9. Profile Page: to display the user information, this view is also updatable.

```
CREATE OR REPLACE VIEW vw_profile AS
SELECT
      UserID,
      Username,
      Email,
      FirstName,
      LastName,
      LastLogin
FROM MyCMS.User;
```

10. Change Password: a dedicated view to update the passwords, this view is updatable.

```
CREATE OR REPLACE VIEW vw_change_password AS
SELECT
      UserID,
      password
FROM users;
```

11. Organize meeting and view meeting: a view to organize and view meeting, this view is updatable.

```
CREATE OR REPLACE VIEW vw_meetings AS
SELECT
      MeetingID,
      UserID AS OrganizerID,
      MeetingTitle AS Title,
      Description,
      MeetingDate AS ScheduledDate,
      Location,
```

```
      CreatedDate AS CreatedAt
FROM MyCMS.Meeting;
```

12. Meeting details: this view is to see more details of the meeting with the guests.

```
CREATE OR REPLACE VIEW vw_meeting_details AS
SELECT
      m.MeetingID,
      m.UserID AS OrganizerID,
      m.MeetingTitle AS Title,
      m.Description,
      m.MeetingDate AS ScheduledDate,
      m.Location,
      mcm.ContactID AS InviteeID,
      mcm.InvitationStatus AS ResponseStatus,
      mcm.ResponseDate AS RespondedAt
FROM MyCMS.Meeting m
LEFT JOIN MyCMS.MeetingContactMapping mcm ON m.MeetingID = mcm.MeetingID;
```

13. Meeting responses: this view is used to see and update the meeting invite responses, this view is updatable.

```
CREATE OR REPLACE VIEW vw_meeting_response AS
SELECT
      MeetingID AS meeting_id,
      ContactID AS invitee_id,
      InvitationStatus AS response_status,
      ResponseDate AS responded_at
FROM MyCMS.MeetingContactMapping;
```

Security Control Measures

Access Control

Creating two users:

```
CREATE USER 'admin_user'@'%' IDENTIFIED BY 'adminpassword';
CREATE USER 'normal_user'@'%' IDENTIFIED BY 'normalpassword';
```

Assigning privileges:

Admin user: as an admin user it will be granted all the privileges to all the tables on the database "mycms"

```
GRANT ALL PRIVILEGES ON MyCMS.* TO 'admin_user'@'%';
```

Normal user: For a regular user only viewing, inserting and updating privileges will be assigned. This may make us think about implementing a logical erase of the data.

```
GRANT SELECT, INSERT, UPDATE ON MyCMS.User TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Company TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Contact TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ContactNickname TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Phone TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Email TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Picture TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ActivityType TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ActivityLog TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ContactNote TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.GroupTable TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ContactGroupMapping TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Tag TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.ContactTagMapping TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.Meeting TO 'normal_user'@'%';
GRANT SELECT, INSERT, UPDATE ON MyCMS.MeetingContactMapping TO
'normal_user'@'%';
```

Accessible tasks example:

Admin user: as the admin user it has privileges for all the tasks, here some examples:

Creating a new table:

```
CREATE TABLE MyCMS.SampleTable (
      id INT PRIMARY KEY,
      name VARCHAR(100)
);
```

Reading data from a table:

```
SELECT * FROM MyCMS.User;
```

Adding a new row to a table:

```
INSERT INTO MyCMS.User (id, first_name, last_name)
VALUES (200, 'Alice', 'Admin');
```

Modifying existing data:

```
UPDATE MyCMS.User
SET last_name = 'Superadmin'
WHERE id = 200;
```

Removing rows:

```
DELETE FROM MyCMS.User
WHERE id = 200;
```

Deleting a table:

```
DROP TABLE MyCMS.SampleTable;
```

Modifying a table structure:

```
ALTER TABLE MyCMS.Company
ADD COLUMN founded_year INT;
```

Granting privileges to other user:

```
GRANT SELECT ON MyCMS.Contact TO 'normal_user'@'%';
```

Regular user: the regular user has some privileges such as viewing, inserting and updating.

Reading data from a table:

```
SELECT * FROM MyCMS.Contact;
```

Adding a new row to a table:

```
INSERT INTO MyCMS.Contact (id, first_name, last_name)
VALUES (301, 'Bob', 'User');
```

Modifying existing data:

```
UPDATE MyCMS.Contact
```

```
SET last_name = 'Patterson'
WHERE id = 301;
```

Out of privileges examples:

Admin user: given that the admin user has all the privileges over the database there are no actions out of its privileges.

Regular user:

Removing records:

```
DELETE FROM MyCMS.Contact WHERE id = 301;
```

Removing a table:

```
DROP TABLE MyCMS.Contact;
```

Changing a table structure:

```
DROP TABLE MyCMS.Contact;
```

Creating a table:

```
CREATE TABLE MyCMS.TestTable (
      id INT PRIMARY KEY,
      name VARCHAR(100));
```

Granting privileges to other user:

```
GRANT SELECT ON MyCMS.Contact TO 'other_user'@'%';
```

Inference Control

Before doing the task in this part of the assignment we create a role "administrator", given that in the task we are ask to grant access to a group of user, and in MySQL a way to do this is by creating a role that will act as a group:

Creating a role:

```
CREATE ROLE 'administrator';
```

Assigning the role to as user (adding the user to the group):

```
GRANT 'administrator' TO 'admin_user'@'%';
```

Now we continue with the first tasks.

Creating a summarized view: we create a view to know how many contacts the database has and what is the average age of the contacts.

```
CREATE OR REPLACE VIEW MyCMS.SummarizedView AS SELECT
      (SELECT COUNT(*) FROM MyCMS.Contact) AS TotalContacts,
      (SELECT AVG(TIMESTAMPDIFF(YEAR, DateOfBirth, CURDATE()))
FROM MyCMS.Contact) AS AverageContactAge;
```

Granting privileges to the view:

```
GRANT SELECT ON MyCMS.SummarizedView TO 'administrator';
```

Accessing and querying: now we access as the admin_user and perform a query to the view.

```
mysql -u admin_user -padminpassword

SELECT * FROM MyCMS.SummarizedView;
```

Flow Control

Altering the login table:

We use the 'user' table to manage the logins and we have the "Password" field that will store the password, first we created it as a VARCHAR(255), planning to save a SHA256, but we realized that only 64 characters are necessary and it is more efficient to have a CHAR field type given that the password will always have the same length.

```
ALTER TABLE MyCMS.User
MODIFY COLUMN Password CHAR(64) NOT NULL;
```

Inserting new users:

```
INSERT INTO MyCMS.User (Username, Password, Role, Email, FirstName, LastName)
VALUES ('alice', SHA2('alice_password', 256), 'ADMIN', 'alice@example.com',
'Alice', 'Smith'),
      ('bob', SHA2('bob_password', 256), 'NORMAL', 'bob@example.com', 'Bob',
'Jones'),
      ('charlie', SHA2('charlie_password', 256), 'NORMAL',
'charlie@example.com', 'Charlie', 'Brown');
```

Procedure to authenticate:

```
DELIMITER //
CREATE PROCEDURE MyCMS.AuthenticateUser (
      IN inUsername VARCHAR(50),
      IN inPassword VARCHAR(255)
)
BEGIN
      DECLARE hashedPassword CHAR(64);
      DECLARE storedPassword CHAR(64) DEFAULT NULL;

      -- Handler for when no row is found (SQLSTATE '02000')
      DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
      BEGIN
      SET storedPassword = NULL;
      END;

      -- Compute SHA-256 hash of the provided password
      SET hashedPassword = SHA2(inPassword, 256);

      -- Attempt to fetch the stored hashed password for the given username
      SELECT Password
      INTO storedPassword
      FROM MyCMS.User
      WHERE Username = inUsername
      LIMIT 1;

      -- Check if a user exists and if the hashed passwords match
      IF storedPassword IS NULL THEN
      SELECT 'Access Rejected: Username not found' AS Message;
      ELSEIF storedPassword = hashedPassword THEN
```

```
      SELECT 'Access Granted' AS Message;
      ELSE
      SELECT 'Access Rejected: Incorrect Password' AS Message;
      END IF;

END //

DELIMITER ;
```

Data Encryption

```
SET @start_time = SYSDATE(6);

SET FOREIGN_KEY_CHECKS = 0;  -- Disable foreign key checks

-- Insert Encrypted Data
INSERT INTO `MyCMS`.`Phone`
    (`ContactID`, `PhoneNumber`, `PhoneType`, `CountryCode`)
VALUES
    (10, HEX(AES_ENCRYPT('603-751-2155', 'MySecretKey')), 'Cell', '+1');

-- Decrypt Data
SELECT
      `ContactID`,
    CAST(AES_DECRYPT(UNHEX(`PhoneNumber`), 'MySecretKey') AS CHAR(32)) AS
decrypted_phone_number,
    `PhoneType`,
    `CountryCode`
FROM `MyCMS`.`Phone`;

SET FOREIGN_KEY_CHECKS = 1;  -- Re-enable foreign key checks

SET @end_time = SYSDATE(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) / 1000000 AS
execution_time_seconds;
```

Managing Passwords with a stored procedure

```
DELIMITER $$

CREATE PROCEDURE ChangePassword(
    IN p_username VARCHAR(255),
```

```
    IN p_password VARCHAR(255),
    IN p_newpassword VARCHAR(255)
)
BEGIN
    DECLARE KnowsOldPassword INT DEFAULT 0;

    -- Check if the provided username and password match
    SELECT COUNT(*)
    INTO KnowsOldPassword
    FROM user
    WHERE username = p_username
      AND password = SHA2(p_password, 256);

    -- If old password matches, update to the new password
    IF KnowsOldPassword > 0 THEN
        UPDATE user
        SET password = SHA2(p_newpassword, 256)
        WHERE username = p_username;

        SELECT 'The password has been updated successfully.' AS Message;
    ELSE
        SELECT 'The provided password is not valid.' AS Message;
    END IF;
END $$

DELIMITER ;
```

Creates a procedure that updates the User's password by first checking to see if the user and old password are valid.

Managing Contents by a trigger

```
DELIMITER $$

CREATE TRIGGER DeleteUserContents
BEFORE DELETE ON user
FOR EACH ROW
BEGIN
    -- Delete associated contacts
    DELETE FROM contact WHERE UserID = OLD.UserID;
```

```
    -- Delete contact notes
    DELETE FROM contactnote WHERE ContactID IN (SELECT ContactID FROM contact
WHERE UserID = OLD.UserID);

    -- Delete contact's phone numbers
    DELETE FROM phone WHERE ContactID IN (SELECT ContactID FROM contact WHERE
UserID = OLD.UserID);

    -- Delete contact's emails
    DELETE FROM email WHERE ContactID IN (SELECT ContactID FROM contact WHERE
UserID = OLD.UserID);

    -- Delete activity logs
    DELETE FROM activitylog WHERE UserID = OLD.UserID;

    -- Delete meeting participation records
    DELETE FROM meetingcontactmapping WHERE ContactID IN (SELECT ContactID
FROM contact WHERE UserID = OLD.UserID);

    -- Delete contact group mappings
    DELETE FROM contactgroupmapping WHERE ContactID IN (SELECT ContactID FROM
contact WHERE UserID = OLD.UserID);

    -- Delete contact tag mappings
    DELETE FROM contacttagmapping WHERE ContactID IN (SELECT ContactID FROM
contact WHERE UserID = OLD.UserID);
END $$

DELIMITER ;
```

Creates a trigger that deletes a contact from the database and removes it from all other tables where there is related data.

Database Integrity

Choosing proper storage engine

Created a Trush table using the Create Table statement in SQL. The table contains three columns: username, FailedAttempts, and SucessfulAttempt. Username is given Varchar 255 and set at the Primary Key which makes sure the username table is unique and not null. The following two columns FailedAttempts and SuccessfulAttempt are TinyInt which is useful when storing small

numbers. The storage engine used is BLACKHOLE, which is a special engine that discards all data entered, not storing any of it. It is useful for login where data does not need to be stored. Through this exercise, we learned how to define a table schema in SQL, set up constraints like primary keys, choose appropriate data types for different use cases, and apply a specific storage engine like BLACKHOLE.

```
CREATE TABLE Trush (
    username VARCHAR(255) PRIMARY KEY,
    FailedAttempts TINYINT,
    SuccessfulAttempt TINYINT
) ENGINE = BLACKHOLE;
```

Created an Inventory table using the create table statement in SQL. The table contains four columns: itemID, Item, Availability, and primaryUser. ItemID is given Int and defined as the Primary Key which ensures that it is unique and not null. Item is given Varchar 255, for storing the item name. Availability is given Int, to keep track of the availability of the items. PrimaryUser is given Varchar 50, this is to store the user of which has updated related inventory. The storage engine used is MYISAM, this is ideal for an inventory table where there are frequent lookups and multiple users that access it. This is due to the fact that it supports concurrent inserts and supports large datasets. Throughout this we learned a better understanding of how to design an inventory management system as well as practiced using the MYISAM storage engine and got to practice having different engines assigned to different tables in SQL.

```
CREATE TABLE Inventory (
    itemID INT PRIMARY KEY,
    item VARCHAR(255),
    Availability INT,
    primaryUser VARCHAR(50)
) ENGINE = MYISAM;
```

For the following 3 parts of code we have two different potential solutions:

Solution 1:

Inserted values into Trush using the Insert Into statement. Used the three columns: UserID, FailedAttempts, and SuccessfulAttempts to insert values sequentially. For example, 1,2,5, inserted 1 into UserID, 2 into FailedAttempts, and 5 into SucessfulAttempts.
Inserted values into Inventory using Insert statement. Used three columns: Item, Availability, and primaryUser to insert values sequentially. For example ‘Phone type: Cell’, 3, 1 has ‘Phone type: Cell’ inserted into Item and 3 inserted into availability and 1 inserted into primaryUser.

```
INSERT INTO Trush (UserID, FailedAttempts, SuccessfulAttempts) VALUES
(1, 2, 5),
(2, 0, 3),
(3, 4, 1);

INSERT INTO Inventory (item, Availability, primaryUser)
VALUES
  ('Phone type: Cell', 3, 1),
  ('Phone type: Cell', 4, 2),
  ('Phone type: Cell', 2, 3);
```

Created a Select Statement to show everything from Inventory and Trush. The statement used itemID,Item,Availability,FailedAttemps,and SuccessfulAttempts from both tables Inventory and Trush, then joins them together. However, since the engine is BLACKHOLE this select statement will return nothing as nothing is stored.

```
SELECT
    I.itemID,
    I.item,
    I.Availability,
    T.FailedAttempts,
    T.SuccessfulAttempts
FROM Inventory I
JOIN Trush T ON I.primaryUser = T.username;

-- But as the Trush table engine is BLACKHOLE, the select will return an empty
result
```

Solution 2:

Inserted values into Trush using the Insert Into statement. Used the three columns: UserID, FailedAttempts, and SuccessfulAttempts to insert values sequentially. For example, 1,2,5, inserted 1 into UserID, 2 into FailedAttempts, and 5 into SucessfulAttempts.

Inserted values into the Inventory table using the results of the select statement. The three columns being inserted are: Item, Availability, and primaryUser. This data is being inserted from both the contact and user tables using the join statement. For the inventory table, the values being inserted can be anything related to primaryUser such as, email, meeting, or contact.

```
INSERT INTO Trush (UserID, FailedAttempts, SuccessfulAttempts) VALUES
(1, 2, 5),
(2, 0, 3),
(3, 4, 1);

INSERT INTO Inventory (item, Availability, primaryUser)
SELECT 'contact' AS item, COUNT(*) AS Availability, u.Username
FROM contact c
JOIN user u ON c.UserID = u.UserID
GROUP BY u.Username;

INSERT INTO Inventory (item, Availability, primaryUser)
SELECT 'email' AS item, COUNT(*) AS Availability, u.Username
FROM email e
JOIN contact c ON e.ContactID = c.ContactID
JOIN user u ON c.UserID = u.UserID
GROUP BY u.Username;

INSERT INTO Inventory (item, Availability, primaryUser)
SELECT 'meeting' AS item, COUNT(*) AS Availability, u.Username
FROM meeting m
JOIN user u ON m.UserID = u.UserID
GROUP BY u.Username;
```

Created a Select Statement to show everything from Inventory and Trush. The statement used itemID,Item,Availability,FailedAttemps,and SuccessfulAttempts from both tables Inventory and Trush, then joins them together. However, since the engine is BLACKHOLE this select statement will return nothing as nothing is stored.

```
SELECT
    I.itemID,
    I.item,
    I.Availability,
    T.FailedAttempts,
    T.SuccessfulAttempts
FROM Inventory I
JOIN Trush T ON I.primaryUser = T.username;

-- But as the Trush table engine is BLACKHOLE, the select will return an empty
result
```

Using Transactions

Solution 1:

The transaction is done over 3 operational steps. For the first step, we get the target username. For the second step,we delete from the user table. For the third step everything is committed. This transaction improves our database security because if any part fails before this, no changes are persisted.

```
START TRANSACTION;

-- Target user to delete
SET @target_username := 'alice';

-- Step 1: Delete from Inventory
DELETE FROM Inventory
WHERE primaryUser = @target_username;

-- Step 2: Delete from user table
DELETE FROM user
WHERE username = @target_username;

-- Step 3: Commit everything
COMMIT;
```

Solution 2:

The transaction is done over 4 operational steps. For the first step, we get the target user ID. For the second step, a cursor is used to iterate over rows in the Inventory table, which contains table names (item) tied to a specific user. For each of these dynamic table names, it constructs and executes a DELETE statement using the prepared statement feature. This approach allows deleting user-related data from various tables that are not statically named in the procedure. For the third step the entries are deleted. And finally the last delete statements for Inventory and User are completed. Only if all operations succeed, the changes are committed. This transaction improves our database security because if any part fails before this, no changes are persisted.

```
DELIMITER $$

CREATE PROCEDURE DeleteUserData(IN target_username VARCHAR(50))
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(100);
    DECLARE user_id INT;
    DECLARE cur CURSOR FOR
```

```
        SELECT item FROM Inventory WHERE primaryUser = target_username;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Step 1: Get UserID
    SELECT UserID INTO user_id
    FROM user
    WHERE username = target_username;

    START TRANSACTION;

    -- Step 2: Loop through all tables in Inventory
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO tbl_name;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Step 3: Build and execute DELETE statement
        SET @sql_text = CONCAT('DELETE FROM ', tbl_name, ' WHERE UserID = ?');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt USING @user_id;
        DEALLOCATE PREPARE stmt;
    END LOOP;
    CLOSE cur;

    -- Step 4: Clean up Inventory and User
    DELETE FROM Inventory WHERE primaryUser = target_username;
    DELETE FROM user WHERE username = target_username;

    COMMIT;
END$$

DELIMITER ;
```

References

(1) HubSpot. "Top Contact Management Software for 2023." *Blog HubSpot*, 2023,

https://blog.hubspot.com/sales/contact-management-software.

(2) HubSpot. "Contact Management CRM | Free Tool for Tracking Customer Information."

HubSpot, 2023, https://www.hubspot.com/products/crm/contact-management.

(3) Capterra. "Best Contact Management Software 2023." *Capterra.com*, 2023, https://www.capterra.com/contact-management-software/.

(4) TechRadar. "Best Contact Management Software in 2023." *TechRadar*, 2023, https://www.techradar.com/best/best-contact-management-software.

(5) Pipedrive. "Contact Management Software for Sales Teams." *Pipedrive.com*, 2023, https://www.pipedrive.com/en/roles/crm-for-contact-managers.

(6) Insightly. "CRM Pricing Plans." *Insightly.com*, 2023, https://www.insightly.com/pricing-plans/.