Contact Management System

MSCS542L

Section 816

Undercover Martyn

Marist College

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

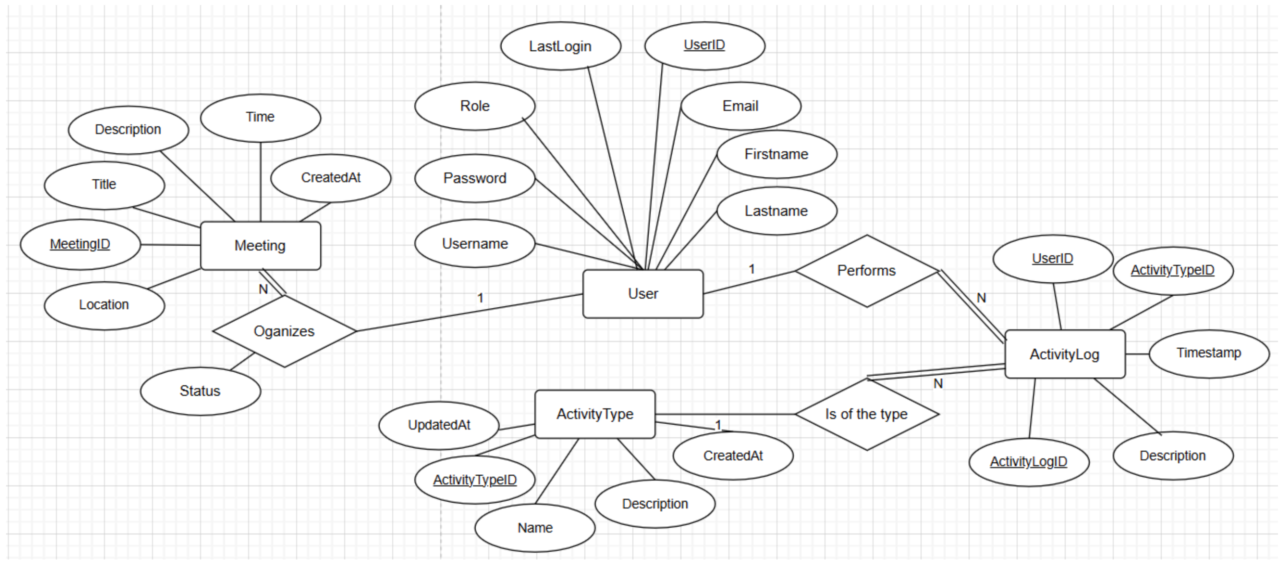
1. External Models

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:



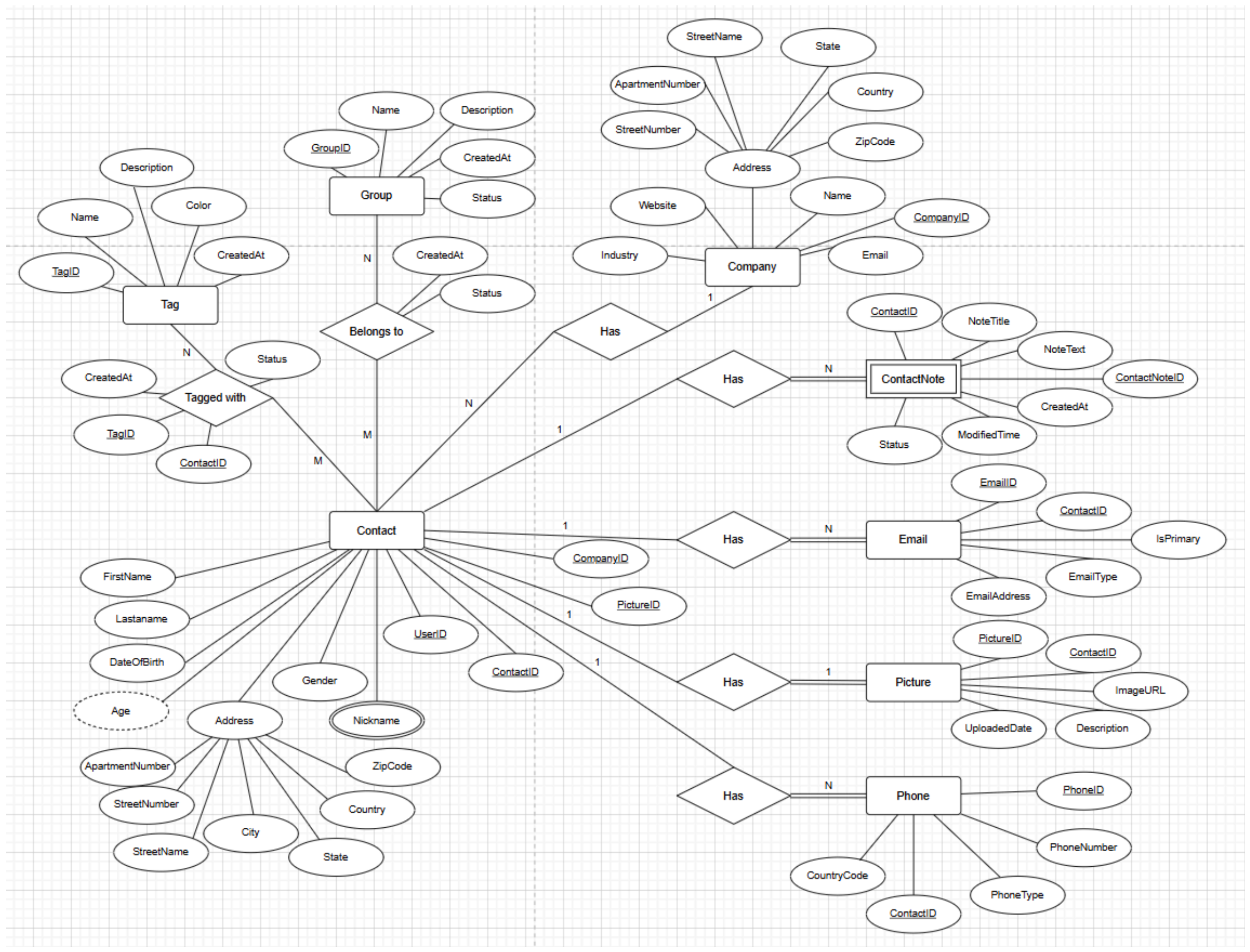
External Model 2: Contacts and Contact Management

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, PictureID, 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, UpdatedAt, CreatedAt
- **ContactNote**: NoteID, ContactID, NoteTitle, NoteText, CreatedAt, ModifiedTime, Status
- **Group**: GroupID, Name, Description, CreatedAt, Status
- **Tag**: TagID, Name, Description, CreatedAt, Color
- **Meeting**: MeetingID, Title, Description, Location, Time, CreatedAt
- **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

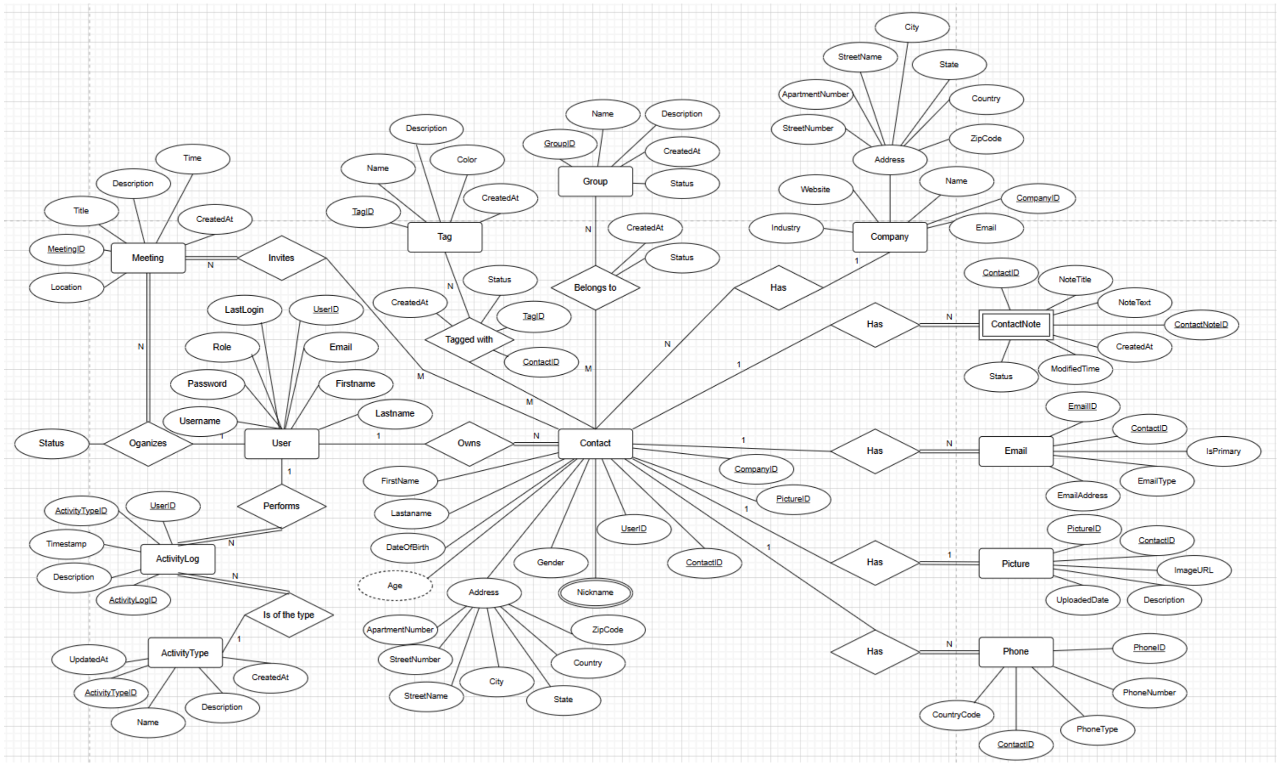
Entities and Attributes

1. **User**

- Attributes: `UserID, Username, Password, Email, Role, FirstName, LastName, LastLogin`

2. **Contact**
    - Attributes: `ContactID, UserID, CompanyID, PictureID, 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, UpdatedAt, CreatedAt`
8. **ContactNote**
    - Attributes: `NoteID, ContactID, NoteTitle, NoteText, CreatedAt, ModifiedTime, Status`
9. **Group**
    - Attributes: `GroupID, Name, Description, CreatedAt, Status`
10. **Tag**
    - Attributes: `TagID, Name, Description, CreatedAt, Color`
11. **Meeting**
    - Attributes: `MeetingID, Title, Description, Location, Time, CreatedAt`
12. **Company**
    - Attributes: `CompanyID, Name, Industry, Address, Email, Website`

ER Diagram:



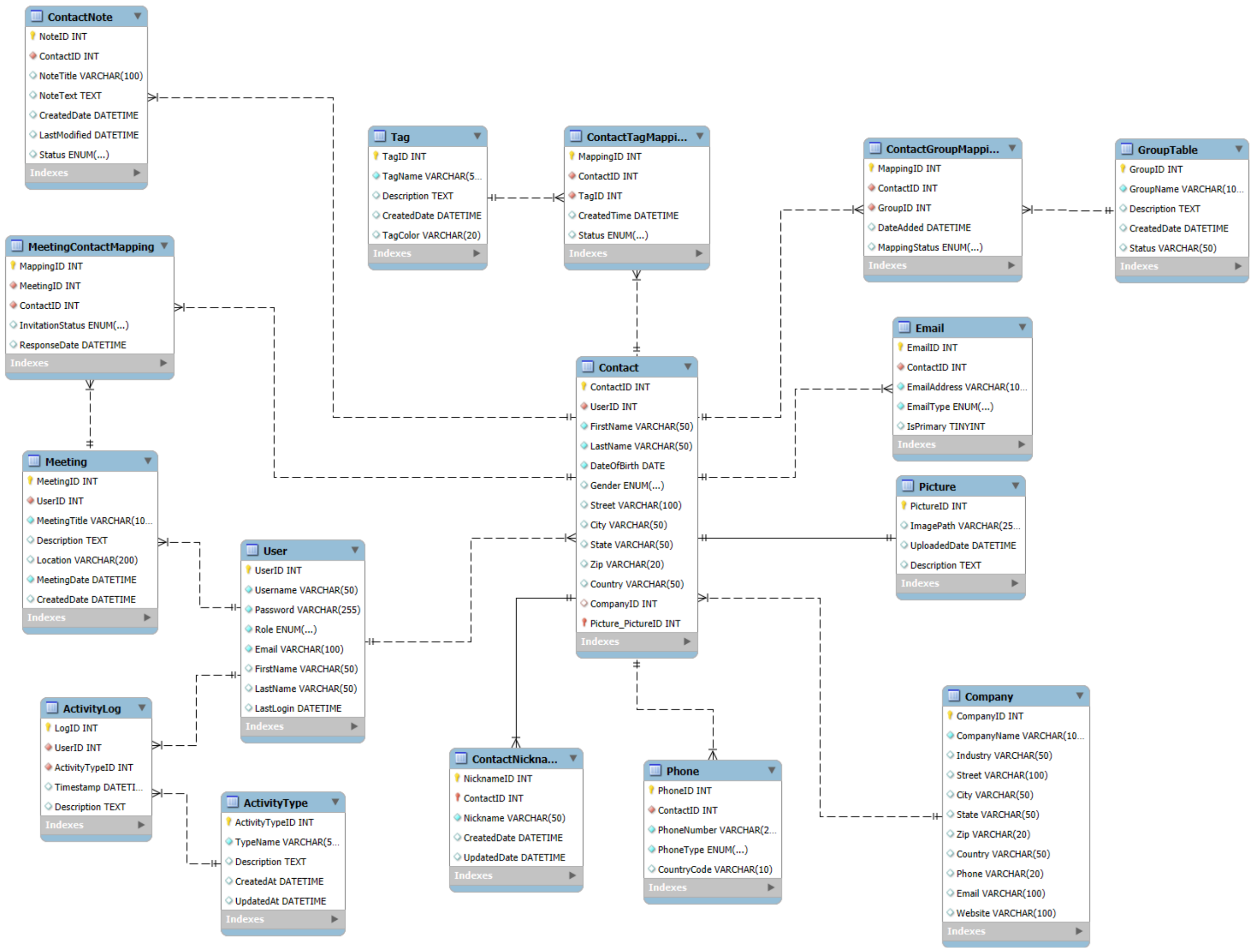
Enhanced Entity Relationship Model

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 **Apartment Number, Street Number**, **Street Name, 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,
  `Picture_PictureID` INT NOT NULL,
  PRIMARY KEY (`ContactID`, `Picture_PictureID`),
  INDEX (`UserID` ASC) VISIBLE,
  INDEX (`CompanyID` ASC) VISIBLE,
  INDEX `fk_Contact_Picture1_idx` (`Picture_PictureID` 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,
  CONSTRAINT `fk_Contact_Picture1`
    FOREIGN KEY (`Picture_PictureID`)
    REFERENCES `MyCMS`.`Picture` (`PictureID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

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.
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 NULL DEFAULT NULL AUTO_INCREMENT,
  `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).
PictureID	INT NOT NULL	FK	Foreign key referencing Picture(PictureID).

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,
  `Picture_PictureID` INT NOT NULL,
  PRIMARY KEY (`ContactID`, `Picture_PictureID`),
  INDEX (`UserID` ASC) VISIBLE,
  INDEX (`CompanyID` ASC) VISIBLE,
  INDEX `fk_Contact_Picture1_idx` (`Picture_PictureID` 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,
  CONSTRAINT `fk_Contact_Picture1`
    FOREIGN KEY (`Picture_PictureID`)
    REFERENCES `MyCMS`.`Picture` (`PictureID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

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
NicknameID	INT AUTO_INCREMENT	PK	Unique identifier for each nickname.
ContactID	INT NOT NULL	FK	Foreign key referencing Contact(ContactID).
Nickname	VARCHAR(50) NOT NULL		The alternative name for the contact.
CreatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the nickname was added.
UpdatedDate	DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP		Timestamp when the nickname was last updated.

Code:

```
CREATE TABLE IF NOT EXISTS `MyCMS`.`ContactNickname` (
  `NicknameID` INT NULL DEFAULT NULL AUTO_INCREMENT,
  `ContactID` INT NOT NULL,
  `Nickname` VARCHAR(50) NOT NULL,
  `CreatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `UpdatedDate` DATETIME NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`NicknameID`, `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.
CreatedAt	DATETIME DEFAULT CURRENT_TIMESTAMP		Timestamp when the activity type was added.
UpdatedAt	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,
  `CreatedAt` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `UpdatedAt` 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);
```

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/.