Information Management Project

Gwen McArdle 18322248

Contents

[Application Description 1](#_Toc88765064)

[Entity Relationship Diagram 1](#_Toc88765065)

[Mapping to Relational Schema 2](#_Toc88765066)

[Functional Dependency Diagrams 2](#_Toc88765067)

[Explanation and SQL Code for Creating the Database Tables 1](#_Toc88765068)

[User 2](#_Toc88765069)

[Device 2](#_Toc88765070)

[Chat 2](#_Toc88765071)

[Participants 2](#_Toc88765072)

[Message 2](#_Toc88765073)

[Media 3](#_Toc88765074)

[Explanation and SQL Code for Creating Views 3](#_Toc88765075)

[User\_View 3](#_Toc88765076)

[Messages\_In\_Chat View 3](#_Toc88765077)

[Participants\_Detail View 5](#_Toc88765078)

[Explanation and SQL Code for Populating the Tables. 6](#_Toc88765079)

[Explanation and SQL Code for Retrieving Information from the Database 7](#_Toc88765080)

[Explanation and SQL Code for Triggers 7](#_Toc88765081)

[Explanation and SQL Code for Security 8](#_Toc88765082)

[App Developers 8](#_Toc88765083)

[Database Engineers 9](#_Toc88765084)

[Explanation of Additional SQL Features 10](#_Toc88765085)

[Recent\_Message Procedure 10](#_Toc88765086)

[Chat\_Participants Procedure 10](#_Toc88765087)

# Application Description

The assignment called for us to develop an ER Model for information to be represented for an application of our choice, and to implement it as a MySQL database. I selected a messaging app as the basis of my project, as I felt that it had a combination of entities with interesting relations. This application would use a user’s phone number as an identifier. A user could then send messages to other users in a chat. The recipients could reply in the chat.

The entities that I identified in this application are User, Device, Chat and Message. The data that needs to be stored includes:

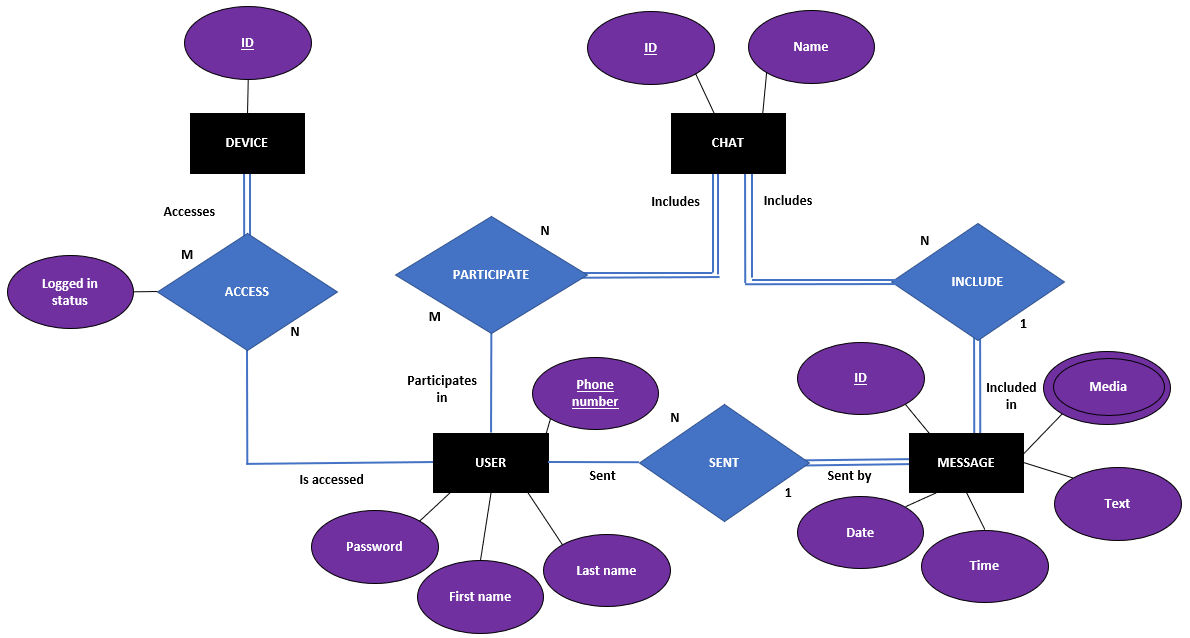
User – first name, last name, phone number and password.

Device – ID.

Chat – ID, name.

Message – ID, text, media, date and time. (Where media is any photos, videos or files that a user sends in a message.)

# Entity Relationship Diagram



In this model, I made the following assumptions:

* A user’s account can be connected to multiple devices, and multiple user’s can be connected to a device.
* A device should not be recorded in a database unless it is connected to a user.
* A user can participate in many chats, and a chat can have many participants.
* A chat cannot exist without any participants.
* A chat can have many messages, but a message can only be in one chat, duplicates will have different IDs.
* A message cannot exist if it is not in a chat, or does not have a sender.
* A user can only have one sender, but a user can send many

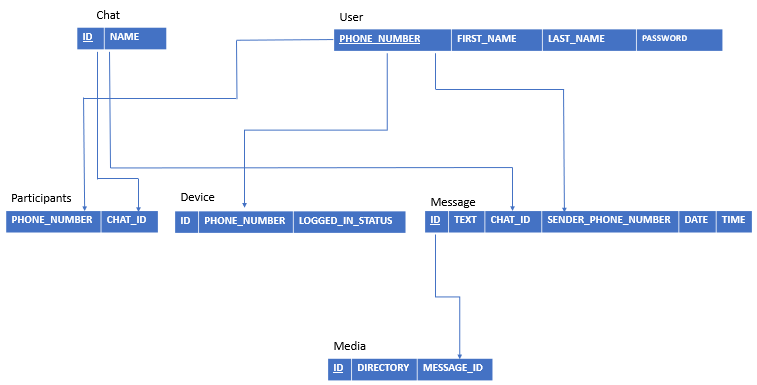
# Mapping to Relational Schema

Each entity must be represented in its own table, except for Device. This was unusual as there was only one data point that I wanted to store for each device; ID. Therefore, mapping the relation between Device and User was sufficient. I named this table Device.

The relation between chats and users is a N:M relation, requiring a separate table, which I named Participants.

The media attribute of Message is a multi-value attribute and therefore, required a separate table, named Media.

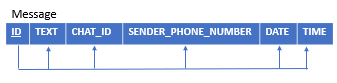
This maps to the below relational schema:



# Functional Dependency Diagrams

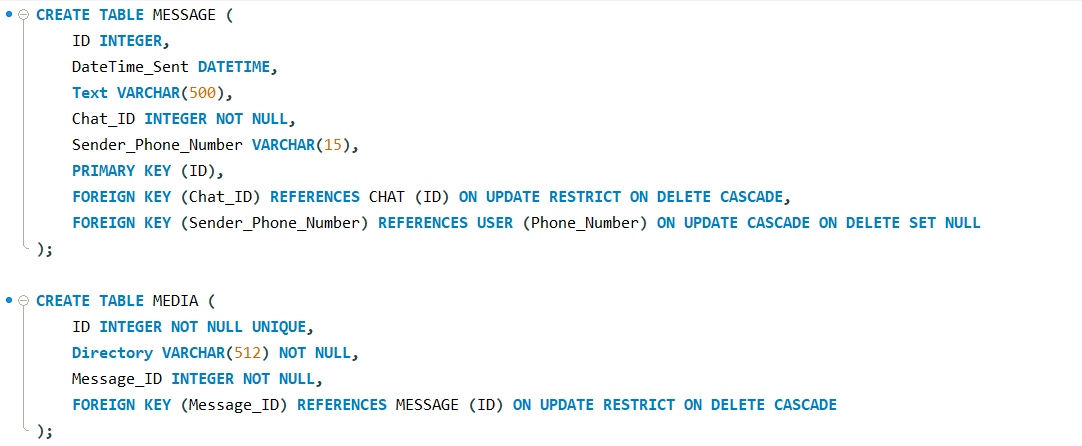
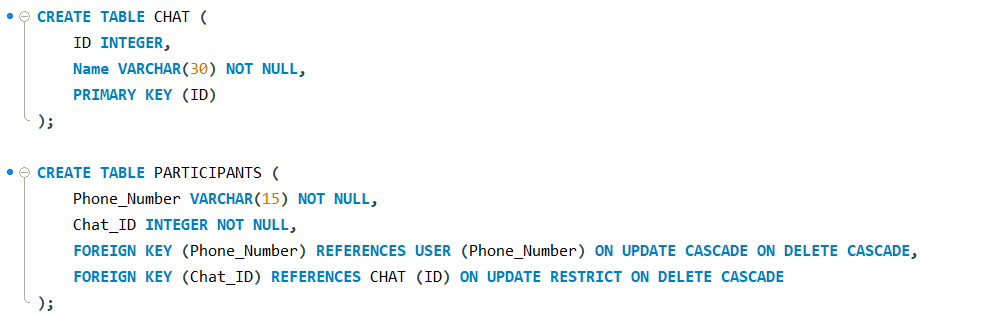
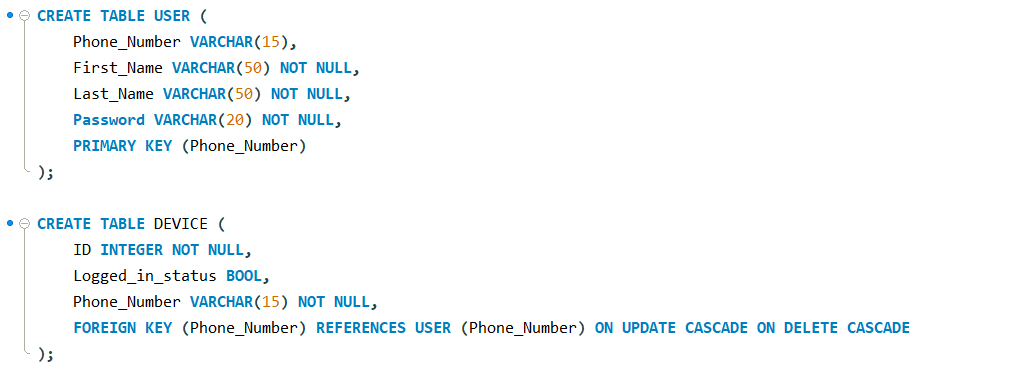
I identified the following functional dependencies:



Device and Participants do not have any functional dependencies as their purpose is to link pairs of foreign keys.

# Explanation and SQL Code for Creating the Database Tables



This SQL script creates six tables; User, Device, Chat, Participants, Message and Media. I used the Relational Schema to establish the order the tables need to be created in. Device references User, Participants and Message reference User and Chat, and Media references Message. Due to these references, I established the order.

## User

**Phone\_Number** – Phone numbers are always unique; therefore, they are a suitable choice for the primary key. “PRIMARY KEY ()” implies “NOT NULL” and “UNIQUE” constraints. The maximum number of digits in a phone number is 15, therefore VARCHAR(15) is sufficient.

**First\_Name** and **Last\_Name** – I concluded that 50 characters would be sufficient for all names, and therefore chose VARCHAR(50) for both name attributes. Messaging applications regularly use users’ names in the interface, therefore these attributes cannot be null.

**Password** – I decided to constrain the passwords to 20 characters, in the interest of not wasting storage space. Every user needs a password so that attribute can’t be null.

## Device

**ID** – Each device needs an individual identifier, however this does not need to be unique in this table as the same device can have access to multiple accounts.

**Logged\_in\_status** – This boolean keeps track of whether or not a user is logged in on a particular device.

**Phone\_Number** – This value is a foreign key and it cannot be null as the purpose of this table is for storing pairs of devices and users. If Phone\_Number is updated in User, this means that the user has changed their phone number, which is their identifier, this value should be updated in Device so that Device can reference the correct User. Therefore, I added “ON UPDATE CASCADE”. If a Phone\_Number is deleted, that means the user’s account has been deleted, and it is no longer necessary to store what Devices are connected to the account. Therefore, I added “ON DELETE CASCADE”.

## Chat

**ID** – Each chat needs an ID to act as the primary key.

**Name** – Each chat must have a name, and therefore cannot be null. I decided to constrain the name to 30 characters.

## Participants

**Phone\_Number** and **Chat\_ID** - These values are foreign keys and cannot be null as the purpose of this table is for storing pairs of phone numbers and chat IDs. They do not need to be unique as a user can be a participant in multiple chats, and a chat has multiple participants. Updates to a user’s phone number should be reflected in this table, however there is no reason for a chat ID to be updated, so this is restricted. If a user or chat is deleted, the corresponding tuple in Participants needs to be deleted.

## Message

**ID** – Each message needs an ID to act as the primary key.

**DateTime\_Sent** – Stores the date and time the message was sent. I used the “DATETIME” datatype for this as it allows the table to queried based on order the messages are sent. The data and time can be cast from this easily. The attribute does not have to be unique, but it cannot be null, because this would cause problems displaying the sequence of the messages in the chat.

**Text** – The text contained in the message. This is constrained to 500 characters, to avoid wasting space on short messages. This can be null as users can send media without text in a message.

**Sender\_Phone\_Number** and **Chat\_ID** - These values are foreign keys and cannot be null as a message must have a sender and belong to a chat in order to exist. They do not need to be unique as a user can send multiple messages, into multiple chats. Updates to a user’s phone number should be reflected in this table, however there is no reason for a chat ID to be updated, so this is restricted. If a user is deleted, the message should still exist in the chat but the sender should be null. If a chat is deleted, the message is deleted with it.

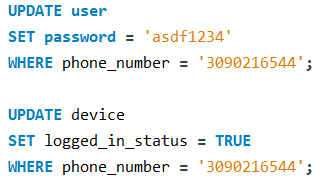
## Media

**ID** – Each media file needs an ID to act as the primary key.

**Directory** – This stores the directory of the media file. The directory should correspond to a bucket the file is stored in. This cannot be null as that would cause application errors.

**Message\_ID** – This is a foreign key linking the media file to the message it was sent in. It cannot be null as files can only be added to the database by being sent in a message. There is no reason that message IDs should be updated, therefore updates are restricted. If the message is deleted, the media is deleted with it.

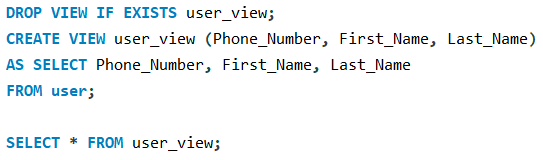
# Explanation and SQL Code for Altering Tables



This code shows some examples of common updates that the app will make to the tables.

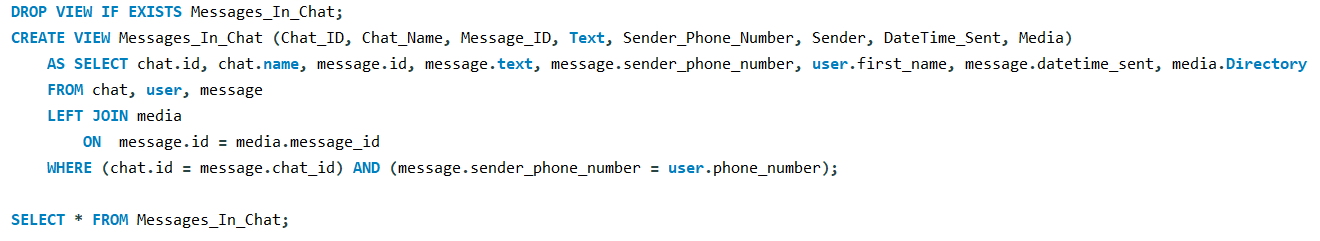
# Explanation and SQL Code for Creating Views

## User\_View



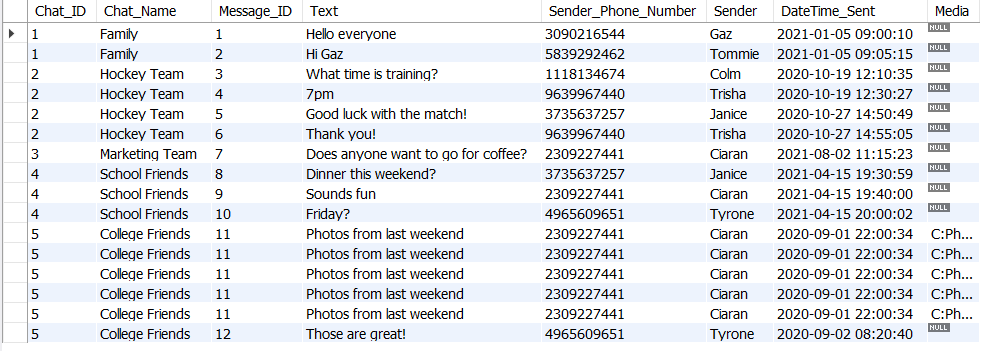
This view is for security reasons. Users and developers of the application should not be able to read passwords saved in the User table. Therefore they can be granted access to this view instead, which does not include the passwords.

## Messages\_In\_Chat View



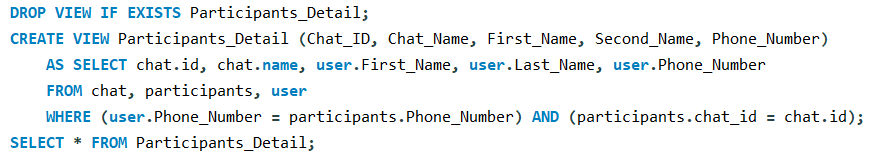
This view lists all of the messages, and adds further detail such as the chat’s name and ID, the sender’s name, and the directory of any media included in the message from other tables. I used a LEFT JOIN to add the media directory attribute because many messages do not have media, and therefore should only be added to the resulting table when there is media. Multiple media files create duplicates of the message in the table.

Here are the results of selecting this view:



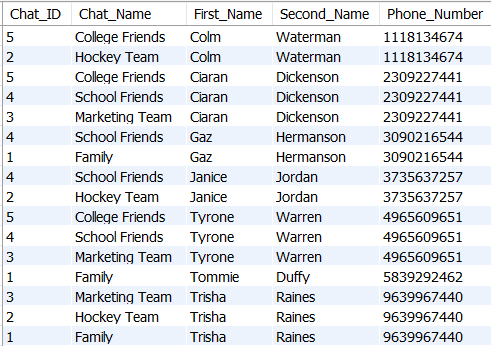
This view is useful for querying the messages in a chat using the procedure described in *Explanation of Additional SQL Features.*

## Participants\_Detail View



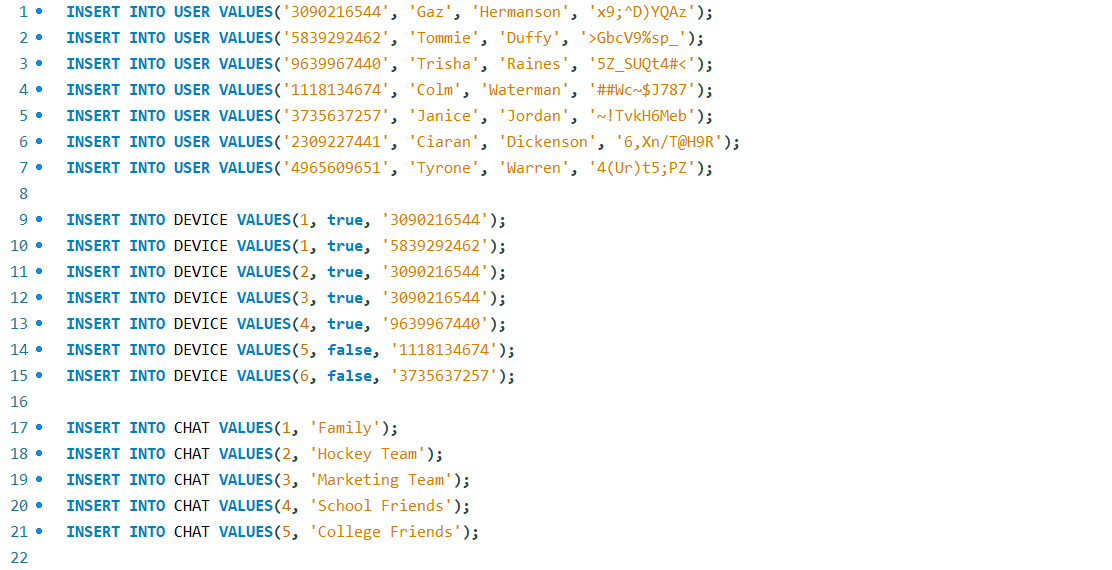
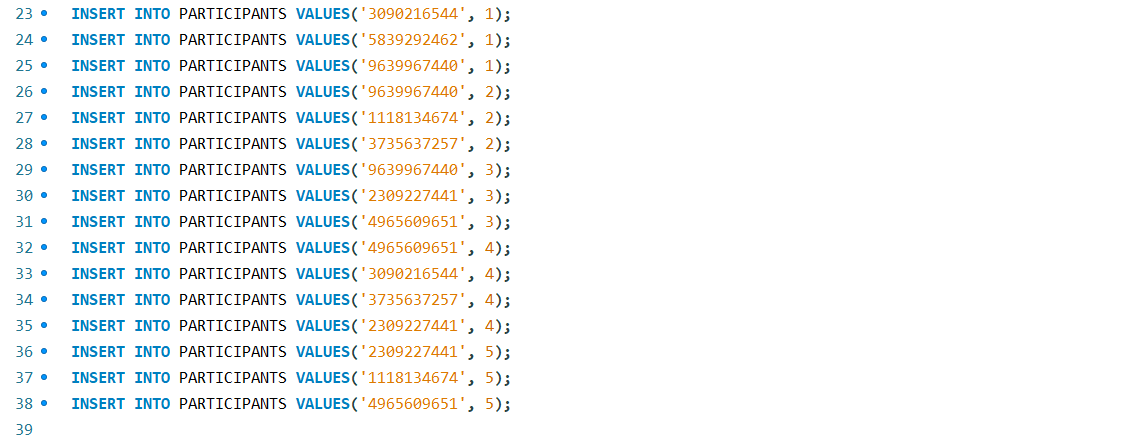
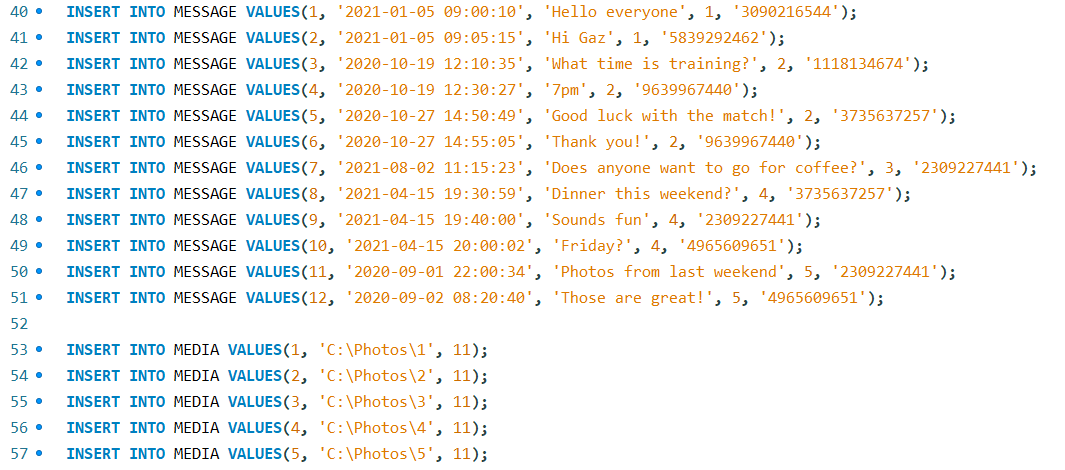
This view lists the tuples in Participants, and adds more detail; the chat name and the user’s first and last name. The tables are joined using the phone number and chat ID attributes in the participant table.

Here are the results of selecting this view:



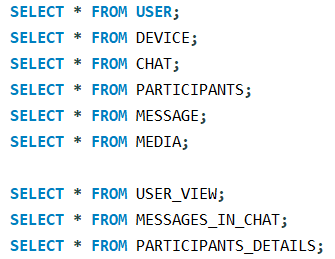
This view is useful for querying the participants in a chat using the procedure described in *Explanation of Additional SQL Features.*

# Explanation and SQL Code for Populating the Tables.

This code inserts sample data into the table. It is performed in the same order as the creation of tables as when a user creates an account, they create data in this order, Device, then Chat, then Participants, then Message, then Media.

# Explanation and SQL Code for Retrieving Information from the Database



These commands each return all attributes in the corresponding table or view.

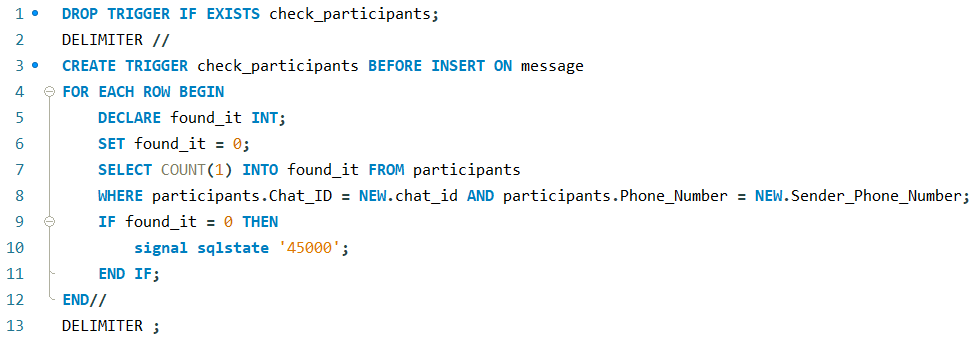


This command returns the name of the user with the phone number 4965609651.



This command returns the chats the user, with the phone number 4965609651, is a participant in.

# Explanation and SQL Code for Triggers



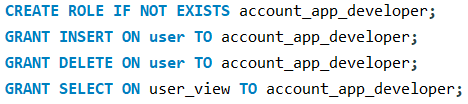
A message can only be sent into a chat that the sender is a participant in. This trigger is to enforce that. Before a message is added to the table, this trigger checks if there is a tuple in participants that includes the chat ID and phone number in the new message. If it does not find such a tuple, an error is signalled and the message cannot be added to the database.

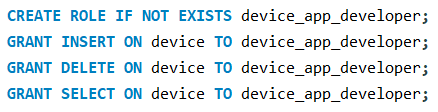
# Explanation and SQL Code for Security

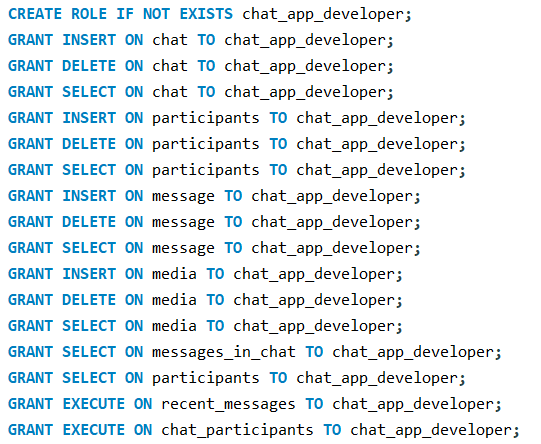
For the purposes of security, I have created multiple roles for different categories of people working on the application and database. I have assigned them privileges accordingly.

## App Developers

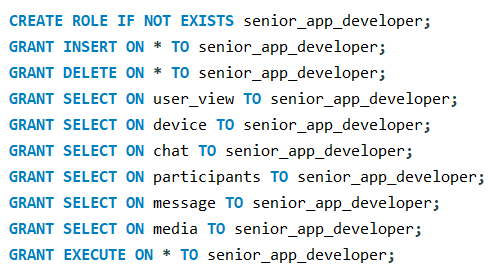
There are three application development teams working on user accounts, how devices connect to user accounts, and the actual messaging component of the app. Each need access to different tables. None of these employees can select the User table, as this includes users’ passwords. Instead, they can select User\_View.





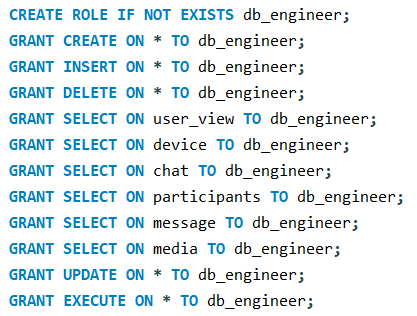


The chat\_app\_developer role has the additional privileges of being allowed to use the procedures that I have created to facilitate retrieving content that needs to be displayed in the application. These procedures are explained in *Explanation of Additional SQL Features*.



The senior application developers have access to everything that the three app developer teams have access to.

## Database Engineers



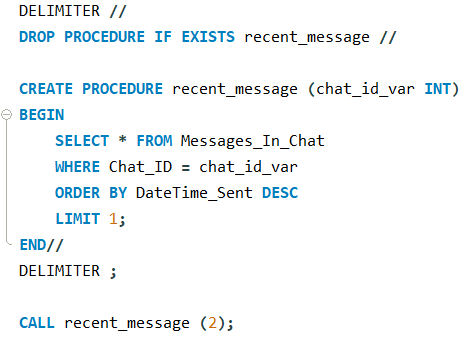
The db\_engineer role allows the user to do most things. They do not have the privilege to drop tables, as minimal people should have access to do this as it could break the application. They do not have SELECT privileges for user, just user\_view, to avoid sharing passwords.



The database administrator has all privileges.

# Explanation of Additional SQL Features

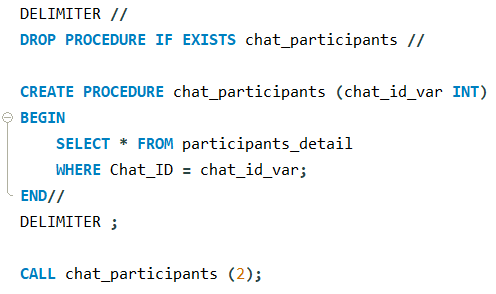
## Recent\_Message Procedure



This procedure takes a chat ID as a parameter, and selects the most recent message from that chat. This procedure is important for the messaging application as the home page should display all the chats the user is in and their most recent message.

I used a procedure to do this as a view cannot take an input variable, and a function cannot return a table in MySQL.

## Chat\_Participants Procedure



This procedure takes a chat ID as a parameter, and selects all the participants in that chat. This procedure is important for the messaging application as users should be able to access a page where they can see who is in the chat they have clicked on.