Name: Kyle Darrow

**Table of Contents**

[Project Direction Overview 2](#_Toc119333351)

[Use Cases and Fields 2](#_Toc119333352)

[Structural Database Rules 2](#_Toc119333353)

[Conceptual Entity-Relationship Diagram 2](#_Toc119333354)

[Full DBMS Physical ERD 2](#_Toc119333355)

[Stored Procedure Execution and Explanations 2](#_Toc119333356)

[Question Identification and Explanations 2](#_Toc119333357)

[Query Executions and Explanations 2](#_Toc119333358)

[Index Identification and Creations 2](#_Toc119333359)

[History Table Demonstration 3](#_Toc119333360)

[Data Visualizations 3](#_Toc119333361)

[Summary and Reflection 3](#_Toc119333362)

# Project Direction Overview

I would like to develop a social media application that uses AI to generate responses from people that would makes sense for that person when asked specific questions called “Parasocial”. This application would function in a way like Instagram or TikTok where users could follow other users and when visiting another user’s page, could type a question and get a verbal response or written response back, like they are talking to the actual person. These responses would be based on questions that the user had answered before that are accessible on the internet, experiences the user would have had which would be inferred by the AI based on location tracking within their devices or photo uploads to the app, as well as the user typing in their own responses to be stored into the database for potential future use by users that visit their profile.

Here is a brief description on how this app would be used. Tom makes an account through the app and allows for the app to track his location. With an empty profile, he decides to upload photos to get started and put in his own answers to generic questions generated by the application to get started. Tom also adds a 1-minute video to his profile introducing himself, so that they AI can use that video and take information based on Tom’s voice to generate an AI Tom for the responses for his followers. He can upload many videos so the app can have more samples to cycle through if he wants to. Once a user visits Tom’s page, they can view his photos (that also contain descriptions) or talk to the AI Tom. When typing a question to Tom, the AI will generate a response based on the information either given by Tom, his photos, or inferred information based on his location or past locations. Any questions that are not in the database will be added, with empty answers if the AI could not answer them and sent to Tom for him to answer personally if he pleases, either through his own voice or typing. This response from Tom will be saved and used for future use on his profile.

Clearly with the use of AI there is a very large programming component for Parasocial, but I plan on focusing on the database design component, which will store profiles, names associated to profiles, locations of users, photos of users, questions to users, user generated answers, AI generated answers, and dates of generated responses. I find this topic very interesting as AI is developing at an exceptional rate and we are likely to see this type of technology in the future. Due to social media applications being used worldwide, I plan for many people to use this app, so I will need to account for each profile within the database as well as all the information within each profile separately.

# Use Cases and Fields

A major aspect of the app Parasocial is account creation as each user would need an account to use the application.

*Account creation Use Case*

1. The user visits the app store to install the application Parasocial.
2. The application asks the user to create an account on initial startup.
3. The user enters their information, like a username and password, and this information is stored in the database along with their account being created.
4. The application asks if the user wants to share their location so they can be tracked on where they go to help better update the database on information based on the user.
5. The user is given the option to create their AI self by making a short video talking to the camera about themselves to help with speech recognition and visual inspection.

The table below covers what fields the use cases will need to store information about an account created by a user.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| AccountName | This stores the name the user wants to associate with themselves on the account. | This is needed so that there can not be users with the same account names and so users can use multiple accounts. |
| FirstName | This stores the users first name tied to the account. | This is necessary when contacting the user of the account through notifications. |
| LastName | This stores the users last name tied to the account. | This is necessary when contacting the user of the account through notifications. |
| DateCreated | This stores the date that the users account was created. | This is used for knowing how long an account has been active. |
| FollowerTotal | This stores the total number of followers that the account has. | This is useful for knowing how much engagement a user is getting towards their account and tracking things like growth or decay of profile visits. |
| VerifiedUser | This stores whether an account is verified or not. | This is needed as there are many fake accounts of people, especially celebrities, on social media. This will help weed out fake accounts or malicious users on the platform. |
| UserLocation | This stores the location of an account. | This is needed to provide the AI with data on locations the user has gone to or currently is, to aid in better user engagement on the account. |

Once an account is created, seeing that the focal point of the application is AI generated responses coming from what looks to be the owner of an account through video as well as their voice overlayed on the response, the user would have to record themselves to get a video for people visiting the account to see. This video would capture the account owner and their voice, which would be played as text to speech when responding to questions.

*AI creation Use Case.*

1. The user accepts to record a video of themselves granting access to their camera and microphone.
2. The user will be given the option to accept or retake the video.
3. Upon completion the user will be given the option to put a filter on their face or voice, depending whether they want to show their face, clear their skin, put on a mask of some sort to keep themselves somewhat anonymous, or sound a different way.
4. The user uploads a video of themselves to their profile, which is stored in the database for use of video and audio data.
5. The user has the option to upload more videos into the database, so that there can be different visual representations of themselves for people visiting their page to look at.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| VideoUsed | This store which videos have been upload to the profile. | This is needed as it will be means of video representation for people visiting the account. Having different videos, the account will cycle through all stored videos in the database to give variety. |
| AudioFilter | This stores the audio information associated with different videos. | This is needed because a user may want a voice changer filter on a specific video and no filter on another, so knowing which filter to use and storing the filters that the user has selected is important to them. |
| DateRecorded | This stores the date in which the videos are recorded. | This is important as some videos may be outdated, or the user may want to know when they last uploaded a recording. |
| FaceFilter | This stores which filter the user chooses for a specific video. | This field is used to display filters that the user has placed over specific videos. |

Another use for the database is to store questions and answers for future use with the AI. The owner of the account will also have the option to view and respond to questions asked to the AI at any point on their account for more accurate responses and less work for the AI.

*AI improvement Use Case*

1. The user selects to improve their AI.
2. The user is given the option to answer random questions about themselves, questions from other users, or to clean up their AI responses.
3. Upon selecting “random questions” The application will have a set of random general questions that the user can respond to verbally or through text, to then be stored in the database for future use.
4. Upon selecting “asked questions” the application will have all the asked questions from other users in which the user can respond verbally or through text, which are then stored in the database.
5. Upon selecting “AI responses” the user can view questions asked to the AI and the AI’s answer.
6. The user can edit these responses to create a more accurate and realistic response for future similar questions.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| VisiterQuestion | This stores questions asked to the AI from another account. | This is needed to help the user see what questions have been asked to their AI so they can contribute answers if they want. |
| UserResponse | This stores a response to a question from the user themselves. | This is needed in case the AI does not answer a question due to an error or if the AI does not answer the question well. The user can fill in their own answer or edit the AI’s if it was close enough. |
| AppQuestions | This stores many basic general questions that you could ask to get to know someone. | This is used to assist the AI in getting some information on the user to use towards its responses without having to wait for other users to ask the questions. |
| AiResponse | This stores the responses of the AI from questions asked to it. | This is needed because it will be used in the database for future responses, and it can also be view by the owner of the account so that they many change some poor responses. |
| WhoAsked | This stores the account name of the user that asked the question to the AI. | This is needed for notifications to questions that have been answered. If a user visits another profile and asks a question that is not answered to their liking or at all and the owner of the account they asked the question on answers their question or edits it, the user who asked the question will get notified of the response. |

Traffic to your profile is important as it helps build the profile and make the AI better overall through a lot of use. There will be a search feature to find people who’s profile you may want to visit.

*People finder Use Case*

1. The user selects a magnifying glass, which acts as a search engine for user in the application.
2. The user is then prompted to type in a name, in which that name is then searched to find a match in the database.
3. The user will be given the exact profile of the name they typed on a new page as well as other profiles with similar names in case they type it incorrectly, giving the user options.
4. The user selects the profile they want to visit and is brought to the selected profile for viewing.
5. The user can then view the profile they searched for or leave the profile.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| AccountName | This stores account names of users who create an account through the application. | This is needed as each account name is unique to an individual, so having unique account names would mean that people would be easier to find or connect to if the name was known. This also allows for multiple accounts. |
| FirstName | This stores the first name of the account owners. | This is needed because if an account owner were to name their account something completely unrelated to their name, there would be no way to find their account through searching their name. Storing a name in the database allows to search for that name without the account name being that exact name. |
| LastName | This stores the last name of the account owners. | This is needed because if an account owner were to name their account something completely unrelated to their name, there would be no way to find their account through searching their name. Storing a name in the database allows to search for that name without the account name being that exact name. |
| FollowerTotal | This stores the total followers of all accounts on the platform. | This is useful when searching for specific people as users with many followers can be favored as they are more than likely the one being searched rather than someone with a similar name. This can also aid in searching for mutual friends as similar followers can be found when searching for an account |
| FollowerName | This stores the account name of the followers for each account. | This is useful as it would help the search engine deduce in who the user may be looking for as some followers may be following the same people, so those people would be favored. |
| DateSearched | This stores the date that a specific account was searched for. | This is useful as the user may want to revisit recent accounts that they have visited and providing the database with this information, the user can be given a list when searching for accounts in order of the date they were searched. |
| UserLocation | This stores the location of an account. | This is needed to provide the AI with data on locations the user has gone to or currently is, to aid in better user engagement on the account. |

How the AI operates is the most vital part of each user’s profile. Each user visiting a profile needs to be able to fully interact the AI with minimal problems giving an immersive experience, like what it would be like to talk to the real person.

*AI usage Use Case*

1. The user would visit another user’s profile.
2. The user would then see the AI already active, but idle, waiting for input.
3. The user typing in a question would then be given a response from the AI, which would be given as though it were a facetime video with the person with a mouth overlay attempting to form mouth patterns similar to what was being said.
4. The user can respond to the response on the right side of the screen with an accuracy rating of how realistic the response sounded on a scale of 1-100.
5. The user can continue asking questions to the AI or leave the profile to view others or leave the application entirely.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| VisiterQuestion | This stores questions asked to the AI from other accounts. | This is needed to generate responses from the AI as well as help make responses better with more use. |
| AiResponse | This stores the answer generate from the AI. | This is needed to aid in accuracy of responses from the AI. |
| VisiterRating | This stores the visitors experience rating of how realist the AI felt based on the questions asked. | This is needed because using a rating comparing questions and answers, the AI will know which answers are more preferred for similar questions in the future. |
| UserResponse | This stores a response from the actual user of the account. | This is needed due to potential inaccuracies from the AI technology. This can always be favored over the AI response as it will be coming from the actual user. It may be empty if the user is very popular, as many questions may be hard to respond to. |
| AskDate | This stores the date in which the question to the AI was asked. | This is needed as some questions and AI responses may be outdated. This would keep responses fresh and up to date. |
| WhoAsked | This stores the account name of the user that asked the question to the AI. | This is needed for notifications to questions that have been answered. If a user visits another profile and asks a question that is not answered to their liking or at all and the owner of the account they asked the question on answers their question or edits it, the user who asked the question will get notified of the response. |

Photos can be useful for storing information on a person regarding where they have been or who they have interacted with. Regarding Parasocial, this can be useful as knowledge of where someone has been or who they have interacted with can give the AI more data to work with. The descriptions on photos can also aid in this as the information comes straight from the user and may contain feelings or emotions towards a thing, people, or place, which the AI could use for better responses towards certain topics.

*Photo section Use Case*

1. The user can choose to view current photos or upload new photos.
2. The user selects to upload a photo from their profile.
3. The user is brought to their photos to select a photo for upload.
4. The user is given the option to add a description for the photo being uploaded.
5. The user can tag the photo to a specific location in which it was taken to help gather more information for the AI.
6. The user is given the option to tag other people in a photo.
7. Once finished with tags and descriptions, the user can confirm the upload, where the photo is then saved to the photo section of their profile.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why It’s Needed** |
| PhotoUsed | This store which photos have been upload to the profile. | This is needed as it will be another form of interacting with an account as well as giving the AI more information to work with when generating responses about a particular user. |
| Description | This stores a custom description about the photo uploaded. | This is needed because a user may want to describe a picture that is on their account. This can also help as it gives more information for the AI to work with. |
| DateUploaded | This stores the date in which the photos are uploaded. | This is important as some photos may become outdated or irrelevant with time. It would also help to have the timeline of photos uploaded so old photos do not display before new photos. |
| LikeNumber | This stores how many likes a particular photo has. | This is important as it can help in finding which photos are more popular on an account and bring more attention to a user who has popular photos. |
| PhotoLocation | This stores the location associated with a photo. | This is important as it can help the AI know where the user has been, the AI can also infer what a user has done or seen based on the location. |
| UserComments | This stores comments from other users on a photo. | This needed as it provides another form of interaction through accounts and stores information on people who comment and what they comment, which could aid in better AI response overall. |

# Structural Database Rules

*Account creation Use Case*

1. The user visits the app store to install the application Parasocial.
2. The application asks the user to create an account on initial startup.
3. The user enters their information, like a username and password, and this information is stored in the database along with their account being created.
4. The application asks if the user wants to share their location so they can be tracked on where they go to help better update the database on information based on the user.
5. The user is given the option to create their AI self by making a short video talking to the camera about themselves to help with speech recognition and visual inspection.

Within this use case there is the application, the user on the application, the database, and the account associated with the user on the application. The entity here is Account as we are not focusing on the user, application or database, but rather the information being stored in that database. The Account entity is the main focus of the first use case in which the other use cases will develop a relationship to.

*AI creation Use Case.*

1. The user accepts to record a video of themselves granting access to their camera and microphone.
2. The user will be given the option to accept or retake the video.
3. Upon completion the user will be given the option to put a filter on their face or voice, depending whether they want to show their face, clear their skin, put on a mask of some sort to keep themselves somewhat anonymous, or sound a different way.
4. The user uploads a video of themselves to their profile, which is stored in the database for use of video and audio data.
5. The user has the option to upload more videos into the database, so that there can be different visual representations of themselves for people visiting their page to look at.

This use case looks at the creation of the account AI, which covers the videos used to portray the user, the filters associated with the videos and the filters associated with the audio. From this I see the 3 entities being, Video, Facial Filter, and Audio Filter. We will need to store which videos are being uploaded as well as the filters associated with those videos.

**1. Each video is contained within an account; each account may contain many videos.**

This rule indicates that every video that is upload must be tied to an account as there is no way to display a video through an account without having an account. It also shows that every account may have none or many videos on it. This is because a user has the option to upload a video to their account to use for the visual immersion aspect of the AI, therefore there can be either no videos on an account or many videos on an account.

**2. Each video may be overlayed with a filter; each filter may be overlayed on many videos.**

This rule indicates that every video can either have a filter or not overlaying the image/ audio. These filters are interconnected, meaning that if you choose a face filter, there will already be a built-in voice to match it, meaning the user does not get to choose it. Each video would be limited to one filter, but a single filter could be used multiple times on different videos. It is possible that in future updates of the application a single video would be able to cycle between different filters, but as the application stands now, I am only looking at capabilities of videos handling one filter for video/ audio at a time or none at all.

*AI improvement Use Case*

1. The user selects to improve their AI.
2. The user is given the option to answer random questions about themselves, questions from other users, or to clean up their AI responses.
3. Upon selecting “random questions” The application will have a set of random general questions that the user can respond to verbally or through text, to then be stored in the database for future use.
4. Upon selecting “asked questions” the application will have all the asked questions from other users in which the user can respond verbally or through text, which are then stored in the database, randomly selecting 1-20 questions to ask the user.
5. Upon selecting “AI responses” the user can view questions asked to the AI and the AI’s answer.
6. The user can edit these responses to create a more accurate and realistic response for future similar questions.

Looking at the AI improvement use case, I see that there are 2 components, questions and answers. Although the questions may be coming from the application itself as a set of pre-made questions or questions from other users, they can all be categorized as questions even though the sources are different as they are all tied to the same account. This is similar to answers, as with answers, each answer could be coming from the AI generated response or the owner of the account themselves. These answers would all be tied to the account that they are answered on. So, from this use case we get 2 entities being Questions and Answers.

**3. Each account has one to many questions; each question is associated with an account.**

This rule indicates that because a user must have an account to receive a question, a question must be tied to an account. An account can have many questions being asked, but only those specific questions will be associated with the account that they were asked to or on. No matter what, an account will have at least 1 question as a random number of questions will be selected and immediately tied to an account upon account creation to help aid the user in AI response generation. Also, the reason each question is associated to an account and not many accounts is that the same question can be asked to different users, but must be stored that way as well, otherwise one user answering a question would be answering the same question for many others.

**4. Each question may be responded to with an answer; each answer responds to a question.**

This rule indicates that a user must have an account before they are able to answer any questions that are being asked of them. Without an account a user would not be able to answer questions. Answering questions is optional, so a user may have no questions answered or many given how much they want to answer. There is also no way an answer can be given without a question as the user would not be able to fill in an answer for something that does not appear on their account.

*People finder Use Case*

1. The user selects a magnifying glass, which acts as a search engine for user in the application.
2. The user is then prompted to type in a name, in which that name is then searched to find a match in the database.
3. The user will be given the exact profile of the name they typed on a new page as well as other profiles with similar names in case they type it incorrectly, giving the user options.
4. The user selects the profile they want to visit and is brought to the selected profile for viewing.
5. The user can then view the profile they searched for or leave the profile.

When looking at the use case for the people finder, although not explicitly stated within the use case, a location tracker comes into play when searching for other users. When dealing with locations, many users may be within a single location at a given time or within a certain area. This aids the algorithm in searching for nearby users when using the search function. Many users may also come up as a result of a search due to similarities regarding account naming. This creates 2 more entities being Location and Search.

**5. Each account may be associated with many locations; each location is tracked by one-to-many accounts.**

This rule indicates that locations can have different accounts within them as users will be asked if their location can be shared while using the application. If many users happened to live within the town of Canton, MA, someone searching for a user within their town may also get results for users with similar names to the user that they are searching for because they are contained within the same location. If a location is in the database it is mandatory for that location to be tied to an account in some way. If the location was not associated with an account, it could not exist within the database unless GPS locations were stored in the database to select through, which is not implemented currently, so for this case we are only looking at user selected locations.

**6. Each search may be saved to many accounts; each account may save many searches.**

It is also indicated that an account name being searched may yield results of the account the user wants to access, but it could also yield no results if the account name was incorrect or no account name similar to the one being searched is within the database. A user can also search for an account and get many results for a searched account name. An account can be searched by many people if it happens to be popular enough. This is also beneficial for a searching algorithm as it will make it easier to find accounts that are more likely to be desired by users.

*AI usage Use Case*

1. The user would visit another user’s profile.
2. The user would then see the AI already active, but idle, waiting for input.
3. The user typing in a question would then be given a response from the AI, which would be given as though it were a facetime video with the person with a mouth overlay attempting to form mouth patterns similar to what was being said.
4. The user can respond to the response on the right side of the screen with an accuracy rating of how realistic the response sounded on a scale of 1-100.
5. The user can continue asking questions to the AI or leave the profile to view others or leave the application entirely.

When using the AI of an account being visited some significant data must be stored, like the interactions between the user and the AI. The user realistically could interact with an AI and do nothing, meaning the AI would just sit there on a video loop, but usually there will be some sort of interaction between the user and the AI. These different interactions should be stored within the entity Interactions itself as the interactions are the most important part of this use case, but not what the specific interactions are.

**7. Each account may create many interactions; each interaction is created by an account.**

This rule indicates that one users AI could have many interactions take place during 1 visit, like asking many questions or rating the AI performance, or none at all if the visitor did nothing but just look at the AI and then leave. If an interaction is registered within the database, there must be an account tied to it as the users are able to interact on the platform without an account. If no interaction occurs the visit will no be stored in the database because storing the data may interfere with potential algorithms in the future that recommend accounts that a user may be interested in. The reason it will not be stored is because visiting another user’s account could be due to a misclick on the screen, where the user hits the wrong name when searching for another user and immediately backs out of the account. If future updates introduce recommendations for users, only real interactions of users will be stored as that is where the user would spend most of their time. Another important aspect of this rule is to remember that users do not have to upload a video to have the AI working, the application will default to using a text based AI if the user does not upload a video initially, which is why this rule is tied to the Account entity and not the Video entity.

*Photo section Use Case*

1. The user can choose to view current photos or upload new photos.
2. The user selects to upload a photo from their profile.
3. The user is brought to their photos to select a photo for upload.
4. The user is given the option to add a description for the photo being uploaded.
5. The user can tag the photo to a specific location in which it was taken to help gather more information for the AI.
6. The user is given the option to tag other people in a photo.
7. Once finished with tags and descriptions, the user can confirm the upload, where the photo is then saved to the photo section of their profile.

Within the Photo use case we can see that important data we want stored is with the photos. These photos will contain tagged locations and people, but people tagged are also only going to be users that can be tagged, which means they also must have an account through the application. From this use case I see 3 entities, that being Photo, Location, and Account.

**8. Each photo is stored in an account; each account may store many photos.**

This rule indicates that a photo on an account may be tied to many different accounts as multiple people can be tagged if they are in the picture. Users also have the option to not upload photos to their account. If a photo is to exist at all within the database, it must be tied to an account of some kind even if no one is tagged.

**9. Each photo may tag a location; each location may be tagged in many photos.**

This rule indicates that while a photo may exist, it does not have to be associated with a location as the user may not tag the location. If a location is to be tagged, there can only be one location tagged per photo. If the user is to tag the location, then an association is formed between the photo and the location. The same location may be tagged by many different users, an example being people posting a picture at a concert, where that concert would have many tags associated with it, but each picture can only have that concert location tagged, and no other location tags. A user could also upload many pictures within the same location to their account which also supports the rule in stating that a location can be associated with many photos.

**Specialization Generalization**

*Verified Account Use Case (NEW)*

1. The user selects account verification option through settings.

2. The user uploads a photo of their driver’s license, which is kept private.

3. The user also is asked to provide an e-mail in which they also must confirm through.

4. After being checked carefully, a user account will then appear with a check next to the account name indicating that the user is now verified.

Accounts are personalized to everyone and must also protect the safety of users. Malicious users may create fake accounts of other users to try and portray them to be something they are not. To combat this issue, account verification will be implemented, and staff will check from information provided whether or not that information is coming directly from the real user. From this use case I see a new database rule being derived.

**10. An account is a normal account or verified account.**

*AI improvement Use Case*

1. The user selects to improve their AI.
2. The user is given the option to answer random questions about themselves, questions from other users, or to clean up their AI responses.
3. Upon selecting “random questions” The application will have a set of random general questions that the user can respond to verbally or through text, to then be stored in the database for future use.
4. Upon selecting “asked questions” the application will have all the asked questions from other users in which the user can respond verbally or through text, which are then stored in the database, randomly selecting 1-20 questions to ask the user.
5. Upon selecting “AI responses” the user can view questions asked to the AI and the AI’s answer.
6. The user can edit these responses to create a more accurate and realistic response for future similar questions.

If we were to look at this use case again, we see that within #3 a user has the option to respond to a question either verbally, in which the verbal response will be read by our program, put into text, and then recorded into the database; or through text in which the user would type their response and enter it into the database that way. From this we get another database rule.

**11. An answer must be verbal, through text, or through AI.**

Looking at this use case again, we can also see that there are different ways questions can be asked to a user on an account, either application generated, or user asked questions. This creates another database rule.

**12. A question can generated by the application or another user.**

**(NEW)20. Each AI response may have many changes; each answer changed is related to an AI response.**

*AI usage Use Case*

1. The user would visit another user’s profile.
2. The user would then see the AI already active, but idle, waiting for input.
3. The user typing in a question would then be given a response from the AI, which would be given as though it were a facetime video with the person with a mouth overlay attempting to form mouth patterns similar to what was being said.
4. The user can respond to the response on the right side of the screen with an accuracy rating of how realistic the response sounded on a scale of 1-100.
5. The user can continue asking questions to the AI or leave the profile to view others or leave the application entirely.

Within the AI usage use case, we can see another database rule as it gives the user the option to rate another users AI responses. These responses will be stored in the database on a scale from 1-100, based on user experience. The higher scores will be favored over lower scores in AI responses, so if a user’s experience rates high, the AI is likely to use similar responses to similar questions for the future for other users. We also see the entity Rating here as well.

**13. AI performance can either be rated by users or not rated.**

**14. A rating is associated to a video; a video may be associated with a rating.**

The reason we see a rule and an entity here is because we can have a rating or no rating at all, which is the rule. The entity created shares a relationship the video entity as each video being generated to try and make a realistic AI is the thing that is being rated. A user may choose to rate the AI generated video or not based on rule, so we see that a rating will always be associated with a video, but a video may not always be associated with a rating.

Seeing that passwords need to be secured, they must be stored in their own separate table. Users can create these passwords upon account creation.

**15. A password is associated with an account; an account is tied to a password.**

Out of the current use cases, the important entities have created 15 structural database rules.

**1. Each video is contained within an account; each account may contain many videos.**

**2. Each video may be overlayed with a filter; each filter may be overlayed on many videos.**

**3. Each account has one to many questions; each question is associated with an account.**

**4. Each question may be responded to with an answer; each answer responds to a question.**

**5. Each account may be associated with many locations; each location is tracked by one-to-many accounts.**

**6. Each search may be saved to many accounts; each account may save many searches.**

**7. Each account may create many interactions; each interaction is created by an account.**

**8. Each photo is stored in an account; each account may store many photos.**

**9. Each photo may tag a location; each location may be tagged in many photos.**

**10. An account is a normal account or verified account.**

**11. An answer must be verbal, through text, or through AI.**

**12. A question can generated by the application or another user.**

**13. AI performance can either be rated by users or not rated.**

**14. A rating is associated to a video; a video may be associated with a rating.**

**15. A password is associated with an account; an account is tied to a password.**

# Conceptual Entity-Relationship Diagram

With the structural database rules I created and listed below, I was able to add to my original ERD.

**1. An account is a normal account or verified account.**

**2. An answer must be verbal, through text, or through AI.**

**3. A question can generated by the application or another user.**

**4. AI performance can either be rated by users or not rated.**

Diagram

Description automatically generated

There is now a Rating entity connected to the Video entity. This Rating entity will contain all rating from videos and their associated id numbers. Due to the user having the option to rate something, it is expected to have some values in the data to be NULL. The Account entity also has 2 subtypes associated with it as well, being Verified or Normal. We see from the graph that the lines connecting the entities are labeled “{Optional, or}”, meaning that an account can choose between wanting to be verified or not, but the user can not have both at once. The Question entity has 2 subtypes, ApplicationQ and UserQ. These entities will cover questions generated from the application itself or the users who ask questions to a particular account. This is labeled as {Mandatory, and} because while the users have the option of asking questions to other users, so it is possible to not have questions from other users, every account in the database has an immediately generated set of questions to help the users get started in improving their AI. This shows that no matter what there will be a question within the Question entity as it will come from ApplicationQ, which makes the relationship mandatory. The relationship also contains “and” as users can also choose to ask questions to another user, in which both application questions and user questions will be contained within the Question entity. Finally, the Answer entity has 3 subtypes as there can be 3 different ways to respond to users. A verbal response, a text response, and the AI response all share the {Mandatory, and} relationship with the Answer entity because no matter what an AI response will be generated even if a question can’t be answered. The AI may respond with “I’m sorry I do not know the answer to that.”, which would be stored in the database. The other responses would be stored in the database as well which gives it the “and” relationship, even though these answers are optional. Without AI responses, the relationship between these entities would be {Optional, and}.

# Full DBMS Physical ERD

Before anything I now need to create attributes along with datatypes. These are all listed in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** | **Example Data** |
| City | CityName | VARCHAR(140) | All locations should be considered when a user wants to tag a location, so we must account for the length of the longest named location in the world. This location has 85 characters in the name, so given that it could be a location someone tags, it must be accounted for along any state, country, or continent within that same description. | Boston |
| Landmarks | LandmarkName | VARCHAR(140) | Along with locations come landmarks to give more specific information on given locations. | Faneuil Hall |
| Photo | PhotoDescription | VARCHAR(1000) | User on social media, especially today, like expressing themselves. Rather than having the picture say everything, people will make very long picture descriptions to tell a story of how the picture came to be, giving background and context to the image. | Got T-boned yesterday and must spend a couple days in the hospital until I get better, I am ok, but I just wanted to let everyone know that. Thank you for all your support! |
| Photo | UploadDate | DATE | Pictures should have a date tied to them as it would be helpful for users to know the date they uploaded their photos and it would also help the UI of the application as more recent photos would show up before older photos. | 10/12/2022 |
| Photo | Comments | VARCHAR(144) | Comments are for users who want to comment on a photo. The comments are limited to 144 characters to avoid responses that are not necessary. | Wow! Looks like a fun time! |
| Photo | Likes | DECIMAL(12) | Likes are to see the amount of user interaction on a particular photo and whether or not a certain photo is more popular than the others. Having 12 digits should be plenty of space for likes as it covers such a large range of people. | 1,322,682 |
| Tags | TaggedUser | VARCHAR(50) | User should have the ability to tag their friends or others in photos. This attribute is limited to 50 characters because account names are also limited to 50 characters. | Kdarrow1 |
| Account | AccountName | VARCHAR(50) | Users will have to have unique account names, which will be limited to 50 characters. This should be plenty of space for each user to create a unique username. | Livefitabs |
| Account | AccountType | CHAR(1) | There will be two types of accounts, one being a normal account and another being verified. Within the database there will either be a “V” for verified or “N” for normal. | V |
| Account | FirstName | VARCHAR(255) | This will be the first name of the person who owns the account. They will have plenty of space to enter their name even if it is long given the 255-character limit. | Kyle |
| Account | LastName | VARCHAR(255) | This will be the last name of the person who owns the account. They will have plenty of space to enter their name even if it is long given the 255-character limit. | Darrow |
| Account | AccountDate | DATE | This will contain the date the account was created. | 12/15/2020 |
| Account | AccountPassword | VARCHAR(255) | Each account will need a password to create/log-in. These passwords will be encrypted and then stored as the encrypted text. | nzltywmfr |
| Account | Email | VARCHAR(255) | This will be the email address the owner of each account signs up with. | kdarrow1@gmail.com |
| Passwords | Password | VARCHAR(50) | Users will have the option to create their passwords, which will be stored here. | Password123! |
| Question | QuestionType | VARCHAR(4) | Some questions will be automatically generated on account creation, and some can be asked by users, therefore there are two possible question types, those being App or User. | App |
| Question | QuestionDate | DATE | Having the date that a question was asked is useful for users as they may not want to answer questions from too long ago and would rather focus on more recent questions. | 1/24/2022 |
| ApplicationQ | Question | VARCHAR(255) | Questions are needed to improve the AI and to develop each users account. This will store the question being asked from other users within 255 characters. | What do you like to do on the weekends? |
| UserQ | Question | VARCHAR(255) | Questions are needed to improve the AI and to develop each users account. This will store the question being asked from other users within 255 characters. | Do you ever go to the gym? |
| Askers | AskingUser | VARCHAR(50) | This will contain the account name of the user who is asking the question. This is 50 characters long being that is the limit for account names. | JeffdaBest63 |
| Answer | AnswerType | VARCHAR(6) | There are three possible ways answers can be generated, either the user giving a verbal response, a text response, or the AI responding. These will be specified as verbal, text, or AI. | text |
| Answer | AnswerDate | DATE | This will give the date that a user gave an answer, which can also help the AI work in the future as it can favor more recent answers over older answers. | 05/24/2022 |
| AIResponsechange | AnswerChangeID | DECIMAL(12) | This is the primary key for the history table that represents when an answer from the AI that is altered on an account by the owner. Giving this DECIMAL(12) allows for many values of changes. | 15 |
| AIResponsechange | OldAnswer | VARCHAR(1000) | This is the previous answer AI or not, that was stored in the database, that the user has changed, but it will be displayed as what it originally was. | I think it would be fun to travel out of the country and see new sights, I’ve always wanted to! |
| AIResponsechange | NewAnswer | VARCHAR(1000) | This is the answer after a user updates a previous answer. Giving this VARCHAR(1000) is plenty of space to revise an answer or give a long-winded answer. | I think I would be too nervous to take a trip out of the country, plane rides of that length freak me out! |
| AIResponsechange | AnswerID | DECIMAL(12) | This is a foreign key to the Answer table, which will be a reference to the ID of an answer pulled from the sub-type “AIResponse” of the Answer table. | 4 |
| AIResponsechange | DateChanged | DATE | This is the date that the answer change occurred on with the DATE datatype. | 07/23/2022 |
| VerbalResponse | Answer | VARCHAR(1000) | All answers will be written out with a limit of 1000 characters as responses are most likely not going to exceed this limit | I enjoy going out with my friends on the weekend, usually playing sports. |
| TextResponse | Answer | VARCHAR(1000) | All answers will be written out with a limit of 1000 characters as responses are most likely not going to exceed this limit | I go to the gym 4 times a week and mostly like focusing on cardio. |
| AIResponse | Answer | VARCHAR(1000) | All answers will be written out with a limit of 1000 characters as responses are most likely not going to exceed this limit | Tacos are probably my favorite food, but it’s hard to choose, I like so many things. |
| Interaction | InteractionType | VARCHAR(20) | There are different types of interactions a user can do when on their account, like giving answers, asking questions, or posting pictures. These will defined as “answer”, “question”, “post”, “comment”, or “like”. | comment |
| Interaction | InteractionDate | DATE | This will contain the date for interactions on an account. | 07/08/2022 |
| Video | UploadDate | DATE | This will contain the data a video was uploaded. | 10/15/2022 |
| Rating | VideoRating | DECIMAL(3) | Ratings will be based on a scale from (0-100) on the performance of the AI. | 74 |
| Rating | DateRated | DATE | This will contain the date a rating was given to an AI. | 09/15/22 |
| Rating | UserRated | VARCHAR(50) | Users that rate the AI responses will have their names recorded in the database. | Touchdown935 |
| Search | SearchDate | DATE | This will contain the date a user was searched for. This will help with ordering searches from recent to old for the user. | 07/13/2022 |
| Search | SearchName | VARCHAR(50) | This will contain the exact information the user searched for, which is also why it is limited to 50 characters. This name may be incorrect to what they desire, but the program itself can return what it thinks the user is looking for. | Tom Bredy |

Diagram, schematic

Description automatically generated

**Normalization**

Given some redundancy in my physical ERD having to do with Location and location names being repeated due to different users posting the same location I have normalized the location information by splitting the tag of the location into two parts, the city, and the landmarks within those cities. Assigning each city an ID and the landmarks their own ID, these ID’s can be referenced to pull the specific names of the locations being addressed without repeating the data.

Another issue with redundancy came with questions being asked as one user has the potential to ask as many questions as they want. To help deal with repeated users, I normalized the user question information by the asking users ID’s. This allows for there to be a unique ID for each asking user. These ID’s are tied to the asking users username in an attribute AskingUser. The Askers entity can be referenced to pull the username of a user asking a question without repeating that name.

The final redundancy issue I ran into had to do with the Photo entity containing TaggedUser, which could appear many times as the same user could be tagged in many different photos. Rather than have that I created an entity Tag that stores each tagged user with an associated ID, which makes each user unique when being referenced from other tables.

A picture containing diagram

Description automatically generated

After normalization there are four new entities City, Landmarks, Askers, and Tag. Creating the City and Landmark entities allows me to split the Location entity up into segments to allow for each city, and landmark within cities that are being referenced, to be unique. This is now also the case for UserQ, as users will not repeat within this entity, but instead will be referenced using Askers, which creates a unique AskID as well as having the username of that asker. Tag is similar as each user has a tag ID in which they can be referenced now.

While the Asker, Landmarks, and Tag tables are normalized to BCNF, the City table is not, as this table could be broken down even more to contain addresses, street names, street numbers, or even put through a broader scope like looking at states. For the sake of this project, I would like to specifically have the location tags be a city and the associated landmark within the city. I think using state as an entity would give too broad of a scope in terms of location and given addresses would be way too specific and could cause for security problems of the users. I do not see these additions benefiting the database all that much and think the City entity accomplishes the goal of what I wanted tags to locations to look like in this project.

Below is a list of my structural database rules I have created so far. New entities are also listed below in bold.

1. Each video is contained within an account; each account may contain many videos.

2. Each video may be overlayed with a filter; each filter may be overlayed on many videos.

3. Each account has one to many questions; each question is associated with an account.

4. Each question may be responded to with an answer; each answer responds to a question.

5. Each account may be associated with many locations; each location is tracked by one-to-many accounts.

6. Each search may be saved to many accounts; each account may save many searches.

7. Each account may create many interactions; each interaction is created by an account.

8. Each photo is stored in an account; each account may store many photos.

9. Each photo may tag a location; each location may be tagged in many photos.

10. An account is a normal account or verified account.

11. An answer must be verbal, through text, or through AI.

12. A question can generated by the application or another user.

13. AI performance can either be rated by users or not rated.

14. A rating is associated to a video; a video may be associated with a rating.

15. A password is associated with an account; an account is tied to a password.

16. Each location may contain many cities; each city is associated with a location.

17. Each city may have many landmarks; each landmark is in a city.

18. Each user question is created by an asker; Each asker may create many user questions.

19. Each photo may have many tags; Each tag is associated with a photo.

**(NEW)20. Each AI response may have many changes; each answer changed is related to an AI response.**

Looking at the new rules add we see that that 16, 17 and 19 are self-explanatory while 18 is a little more complicated. Within these rules there is a difference between a question and a user question. A question encompasses all questions being asked, while a user question is a sub-type of the question entity, just like application, which stands for application questions. When 18 states that each user question is created by an asker, it is differentiating user questions from application questions, as users can not generate those. Therefore, each user question is created by an asker (person asking a question) and each asker may generate many user questions. Hopefully that clears up some confusion as 3 and 18 may seem like similar rules but are different entirely.

Below is a quick display of my conceptual ERD to show the new additions that were made.

Diagram, schematic

Description automatically generated

This conceptual ERD shows the new entity added being the history table named AIResponseChange as well as the other entities added after normalization. In the future there will be implementations to allow users to change their text based or verbal based answers, but for now I am only focusing on AI answer changes because it is the only factor right now that is out of the users control, so adding this history table allows the user to control what the AI says to some regard by adapting answers that may be incorrect.

# Stored Procedure Execution and Explanations

Update, if necessary, your screenshots and explanations of defining and executing your stored procedures to transactionally add data to your database.

After adding in AIResponseChange as a history table, there are some things we must also add within our stored procedures. To get an answer, a user could either answer an application question or a user question that was asked to them. Even though our database already has some application questions within it, I wanted to also add some user questions. To do this, we create a stored procedure called AddUserQ, which will add data to the tables Question, Asker, and UserQ.

Graphical user interface, text, application

Description automatically generated

Using this stored procedure, we can do just that. Seeing that this stored procedure is focused on inserted user questions, we can hardcode “User” as the QuestionType and use GETDATE() which will get the date the question was asked. We still need to know the account ID that is receiving the question, the question itself, and the user that is asking the question, which are the values we will insert through our execution statements. The remainder of the procedure includes sequences which get the next value for the QuestionID and the AskID. With this procedure in place, we can now execute the procedure with some values so that we can fill our database.

Graphical user interface, text, application, email

Description automatically generated

After execution of the procedure, we now have some questions from user that we can answer going forward.

Table

Description automatically generated

With questions now in place and ready to be answered, I can now create a stored procedure that will help populate the Answer table as well as the AIResponse table since that is the table we are looking to get the history from.

Graphical user interface, text, application

Description automatically generated

Using this stored procedure, we can now add data to the database through execution statements.

Graphical user interface, text, application, email

Description automatically generated

With the executed data, we now have answers to questions that have the QuestionID 6 and 7. This can be shown below.

Table

Description automatically generated

# Question Identification and Explanations

**First Query**

**Which normal accounts have a filter on their videos?**

This question checks which users use filters on their videos. Not only would help us know which filters are more popular, but we could also see whether Normal or Verified accounts use filters more. This could help us also find users that do not use filters and if it happened that many users on the platform did not use filters, we could find ways to make filters easier to manage or implement. We could also use this information to focus on improving filters that are less used, so that they get more traction.

**Second Query**

**Which verified accounts have requested an application question?**

This question is important as it helps filter out all Normal accounts, which are going to be the majority, from the verified accounts. It will also help see which questions are being given to these users from the application and may provide some insight on whether verified users even care about application question as they already may be getting enough from other users given their popularity.

**Third Query**

**Which users created an account after the date 12/31/2022 that use the filter “animal”?**

This question is great as it allows for tracking how popular certain filters within the application are. We could compare the amount of user currently using a filter within the database to the number of new users after the given date to help chart growth or popularity of a filter, possibly after an update.

# Query Executions and Explanations

**First Query**

**Which normal accounts have a filter on their videos?**

Graphical user interface, text, application

Description automatically generated

This is a great demonstration that can be useful for many reasons. If implementation filters were low for the total number of users in the database, then we could put more focus on making seamless transition for adding a filter onto a video. This could also help show the popularity of some filters with further filtering of the data, which could give information on what people enjoy using to pull more attention to the application. Within this query we are pulling the account name, account type, and the filter type for each account that does have a filter. We can get rid of all options that do not have a filter, but using a WHERE clause, and checking to see all options where the filter type is not equal to “None”. We would not be able to pull the information of filter types without JOIN because there is no relationship between Account and Filter, only Video and Filter. We first join Video and then join Filter by the FilterID’s as those are shared between Video and Filter. This then allows us to develop a connection between all tables included to get the results we want. The reason we do not see verified users in this example is because we did not join with the verified table, as we are not looking for those users.

**Second Query**

**Which verified accounts have requested an application question?**

**Graphical user interface, application

Description automatically generated**

This is a simple demonstration of how we can pull which verified accounts have requested an application question. This is important as verified account would tend to be celebrities or people of higher status in society and they would already have all sorts of traffic to their profiles. Seeing that users can ask questions, the more traffic to your page, the more users asking questions to you, the less likely you will need to answer generated questions from the application itself. This is important for potential algorithms in the future for the application. We could implement a system where users with less traffic to their profile get more notifications to request questions from the application to help improve their AI. If there is a very low ratio of verified accounts within the database to verified accounts in the database that request questions from the application, then it would be known that verified accounts do not need much focus or direction with getting generated questions and the application could be more forward to normal users who need assistance in building up their AI. Notice that I did not pull data from all questions requested from verified users as this would cause a skew in the data. In this example MrMogie and MrsMogie appear as the 2 verified users that have requested an application question, but when looking at all of the data, MrMogie has actually requested 2 questions. The reason his name does not appear twice is because of DISTINCT. This makes it so there are no repeats within the data being displayed, as a repeat in this case, where there are 2 verified people in the database means that there would be 3/2 verified users requesting questions, which is impossible. The rest of the query pulls information from the Question table and the ApplicationQ table using JOIN’s, filtering out any normal accounts using a WHERE clause that looks for account types matching “V”.

**Third Query**

**Which users created an account after the date 12/31/2022 that use the filter “animal”?**

Graphical user interface

Description automatically generated

Application

Description automatically generated with low confidence

This query is useful and would be used a lot for a popular social media application that constantly generates new filters for users, almost like Snapchat! With new filters constantly being created and tested, it would be important to know which filters fall flat and others that that succeed. This query does exactly that by return a list of users that used a filter after a certain date. This would be important information if the application wanted to know how many users have used their newly added filter from a week ago, which could then be compared to other filters that also had their usage recorded after a week of release. In this query I created a view called After\_2022 that pulls the AccountName from the table Filter that has a date after 12/31/2022 and has used the filter “Animal”. The Filter table alone does not have all of the data that I need, so I must JOIN the Video table and the Account Table. The Video table is a necessary JOIN because the Filter table and the Account table do not share a relationship, but the Video table shares a relationship with the Filter table and the Video table also shares a relationship with the Account table. This helps us pull all of the information needed to get the AccountName from filter that is associated with the constraints put in place by the WHERE clause. After specific AccountNames I am looking for are assigned to After\_2022, I then call that view within another SELECT statement along with the first name, last name, and date the account was created. I must also JOIN the Account table again because After\_2022 is only assigned the names of the users accounts that I wanted, but not the other information being requested. After the JOIN and running the script, I now get the requested data with the help of the view.

**(NEW) Forth Query**

**How many normal vs. verified users use Parasocial?**

It is important to know the users of your application. Having celebrities use your application may influence others to also download the app and make accounts. For this reason, I find it important to know the ratio of users that have normal accounts vs users that have verified accounts. Verified users are more likely to be people who many others are interested in and have the potential to bring more users to the application. It would be helpful to know the number of normal users and the number of verified users so that the application could maybe market towards celebrities to help with growth as they could generate more users for the application.

Graphical user interface, text, application

Description automatically generated

Although simple, this query provides important information for the development of the application and for the growth that will be desired. This query pulls the total number of normal accounts and verified account contained with the application. Using this query, we can display it through data visualization to get useful information regarding users of the application.

**(NEW) Fifth Query**

**What is the number of questions each user currently has on their account?**

This query is important as it can tell us a lot about how users are interacting with the application and which users may be more popular than others. Many questions tied to an account means that a user either has a lot of other users going to their account to ask questions or the user is interacting with their account and trying to make their AI better through requesting questions. Either way, the higher the number, the better because it means a lot of user interaction either way.

Graphical user interface, text, application, email

Description automatically generated

Using this query, we can now see the representation of how many questions are tied to a specific account. This can give us important information about users, whether they are getting traffic to their accounts or if they are trying to improve their AI on their account. This query creates a great foundation to build and get more specific answers from. This query can also be represented visually to give a better image of the user account interactions on the application.

# Index Identification and Creations

|  |  |
| --- | --- |
| Primary Key Column | Description |
| City.CityID | This is the primary key of the City table. |
| Landmarks.LandkmarkID | This is the primary key of the Landmarks table. |
| Location.LocationID | This is the primary key of the Location table. |
| Tag.TagID | This is the primary key of the Tag table. |
| Photo.PhotoID | This is the primary key of the Photo table. |
| Account.AccountID | This is the primary key of the Account table. |
| Video.VideoID | This is the primary key of the Video table. |
| Filter.FilterID | This is the primary key of the Filter table. |
| Interaction.InteractionID | This is the primary key of the Interaction table. |
| Question.QuestionID | This is the primary key of the Question table. |
| Answer.AnswerID | This is the primary key of the Answer table. |
| Rating.RatingID | This is the primary key of the Rating table. |
| Asker.AskID | This is the primary key of the Asker table. |
| Search.SearchID | This is the primary key of the Search table. |

Due to all primary keys being unique, all primary keys that are not surrogate keys are listed here.

|  |  |
| --- | --- |
| Foreign Key Column | Description |
| City.LandmarkID | This foreign key in the City table references the Landmarks table and is non-unique as there can be many landmarks in a City. |
| Location.CityID | This foreign key in the Location table references the City table and is non-unique as there can be many cities in a Location. |
| AccountLocation.AccountID | This foreign key in the AccountLocation table references the Account table and is non-unique as there can be many account within AccountLocation. |
| AccountLocation.LocationID | This foreign key in the AccountLocation table references the Location table and is non-unique as there can be many locations within AccountLocation. |
| Photo.LocationID | This foreign key in the Photo table references the Location table and is non-unique as there can be many locations taken in a location. |
| Photo.AccountID | This foreign key in the Photo table references the Account table and is non-unique as there can be many photos added to an account. |
| Photo.TagID | This foreign key in the Photo table references the Tag table and is non-unique as there can be many tags on many photos. |
| Verified.AccountID | This foreign key in the Verified table references the Account table and is non-unique as there can be many verified accounts. |
| Normal.AccountID | This foreign key in the Normal table references the Account table and is non-unique as there can be many normal accounts. |
| Passwords.AccountID | This foreign key in the Passwords table references the Account table and is non-unique as different account owners can have the same password. |
| Question.AccountID | This foreign key in the Question table references the Account table and is non-unique as there can be many questions per account. |
| ApplicationQ.QuestionID | This foreign key in the ApplicationQ table references the Question table and is non-unique as there can be many application questions. |
| UserQ.QuestionID | This foreign key in the UserQ table references the Question table and is non-unique as there can be many user questions. |
| Answer.QuestionID | This foreign key in the Answer table references the Question table and is non-unique as there can be many answers to questions. |
| VerbalResponse.AnswerID | This foreign key in the VerbalReponse table references the Answer table and is non-unique as there can be many responses as answers.. |
| TextReponse.AnswerID | This foreign key in the TextReponse table references the Answer table and is non-unique as there can be many responses as answers.. |
| AIReponse.AnswerID | This foreign key in the AIResponse table references the Answer table and is non-unique as there can be many responses as answers. |
| Interaction.AccountID | This foreign key in the Interaction table references the Account table and is non-unique as there can be many interactions on an account. |
| Video.FilterID | This foreign key in the Video table references the Filter table and is non-unique as different videos can have the same filters. |
| Video.AccountID | This foreign key in the Video table references the Account table and is non-unique as there can be many videos per account. |
| Rating.VideoID | This foreign key in the Rating table references the Video table and is non-unique as there can be many ratings per video. |
| AccountSearch.SearchID | This foreign key in the AccountSearch table references the Search table and is non-unique as there can be many searches stored in total account searches. |
| AccountSearch.AccountID | This foreign key in the AccountSearch table references the Account table and is non-unique as there can be many searches per account. |

For the first query driven index I selected Account.AccountName. This would be used to help find account that use a particular name, if a user happened to be reported, then it would be easy to find this user given they have an AccountName. This also avoids any possible confusion between users and data. This will be a unique index because all account must have different names.

The second query driven index I selected is UserQ.Question. This would be used to find common questions that people are being asked, which could be helpful in updating the application questions as well. This could also help us filter out any inappropriate questions or put censorship on certain words or phrases. This would be non-unique as the same questions could be asked to different verified users.

The third query driven index I selected is Account.AccountDate. This could be used for many things, not only find when users created an account, but to also monitor how often the application is getting new users. There could be spikes or declines based on updates through the application, so knowing that data would be important as a decline may signify that developers would need to work on and implement something soon. This is also non-unique because many accounts created on the same day.

Graphical user interface, text, application

Description automatically generated

I named the first index “AccountNameIdx” to show that it is taking the account name. This index is referencing the unique column AccountName in the table Account.

The second index is named “UserQuestionIdx” to show that it is taking the User asked questions. This index is referencing the non-unique column Question in the table UserQ.

The third index is named “AccountCreationIdx” to show that it is taking the date an account was created. This index is referencing the non-unique column AccountDate from the Account table.

# History Table Demonstration

Explain the specifics of your history table, including how the trigger works, and demonstrate that the history table captures changes.

Graphical user interface, text, application

Description automatically generated

The image above shows the creation of my history table, which is named AIResponse change. The reason for this table is to track any changes made from the user to questions answered by the AI. A user may find that questions from the AI are not to their liking or accurate, so any change that a user implements will appear in this table. Although the user should have the option to change any subtype of the Answer table, those being VerbalResponse, TextResponse, and AIResponse, for this example I decided to only focus on the AIResponse as that is the only response that will be out of the user’s control as the user could answer exactly how they wanted through a text response or verbal response because it is coming directly from them initially. The TextResponse and VerbalReponse tables are also less likely to be populated with as much data as the AIResponse table because the entire purpose of the application is to rely on the AI to respond for the user, so there are likely to be many more response from the AI than any other sub-type. This is why it is important we focus on tracking AIResponse changes for now because while they may be put in the database as the AI’s response, the user will have the option to change those responses to their liking, in which the user response will replace the previous AIResponse.

With the focus on AIReponse changes we can now create a trigger that will notify us whenever there is a change to the AIResponse table.

Graphical user interface, application

Description automatically generated

Breaking the trigger down to its components, I first create the trigger name AIResponseChangeTrigger on the Answer table. This trigger will occur after an update and will capture both the old and new responses through accessing the Answer table and its related AIResponse table under the conditions that the answer type is “AI”, meaning the answer had originally come from the AI. This answer is then checked to see if the new answer matches the old answer, if it does, then nothing is updated, but if the answers are different, then the insert statement is run, filling out the AIResponseChange table with its AnswerChangeID (primary key), the old answer, the new answer, the AnswerID (foreign key), and the date the change occurred on.

After creating all the stored procedures and executions associated with Answers and Questions we can now put the trigger to the test. To test this trigger, we can change one of the responses the AI gave to the users.

Graphical user interface, application, Word

Description automatically generated

We can see from this screenshot that when updating data 4 total rows were affected. There are 2 rows in the AIResponse table that are affected and 2 rows in the AIResponseChange table. Using this information we can break down exactly what happened with our data as well as check if the trigger worked correctly. To check this, I first look at my AIResponse table.

Graphical user interface, text, application, email

Description automatically generated

When looking at this table previously there were only 2 rows, and we still have 2 rows, but notice that the first row has its column “Answer” changed from what it originally was in prior examples. When updating the data from the previous screenshot, 2 of the affected rows occurred in this table, but separately occurred on row 1. We are just seeing the result of the last update we made. The other 2 rows affected have to do with the AIResponseChange table, which is displayed below.

Graphical user interface, application

Description automatically generated

From this we can see that our trigger worked and recorded the proper data that we wanted. It shows exactly what happened to the AIResponse table and what the old answer was replaced with, along with the ID of that answer and the date it was changed. This is all we need for confirmation that our trigger and history table are functioning how we intended them to.

# Data Visualizations

Include and explain data visualizations and stories that tell effective stories about the information in a way that people quickly and accurately understand it.

With our fourth query being answered in a previous example, we can now display our results in a way that is easier to read and view through data visualization.

**Forth Query**

**How many normal vs. verified users use Parasocial?**

Graphical user interface, text, application

Description automatically generated

Seeing that this a simple comparison of the number of users that have a normal account and the number of users that have a verified account, I created a bar chart to represent the results.

This chart tells us the number of verified users is lower than the number of normal users, which would be expected as there are more people in the world who aren’t celebrities or public figures than there are average people in the world. With such a small population sample, the data may be slightly misrepresented, but with a larger sample size we could expect to see the ‘N’ bar be much larger than the ‘V’ bar. If we were to use data from outside resources that said that every 1/1000 people is a celebrity, then we could use our data to compare to see if the comparison matches. If the number of verified users was lower than the desired outcome, we could use this data and potentially market towards celebrities with some sort of incentive to get them to join the platform, which will in turn grow the normal user’s numbers as well, creating a hopefully endless cycle of growth as new people may become celebrities, join the platform, and then bring more users.

With our fifth query being answered in a previous example, we can now display our results in a way that is easier to read and view through data visualization.

**Fifth Query**

**What is the number of questions each user currently has on their account?**

Graphical user interface, text, application, email

Description automatically generated

This bar chart shows us the number of questions that each user currently has on their account whether or not they are answered. This information could be great for estimating the amount of application usage on an account or across accounts. The X-axis lists all the users currently within the database and the Y-axis lists all the questions tired to their account. The more questions, the more account interaction either by popularity, where other users are visiting their account and asking question to them, or by wanting to improve AI, where they are requesting application questions to help make their AI more robust. With lower questions on an account, we could potentially send notifications toward users that will hopefully spark interest in either visiting other users and asking questions or answering application questions to make their own AI better and hopefully get more traffic to their account.

# Summary and Reflection

My database is for a mobile application called Parasocial, which uses user data to create AI generated responses and characters for everyone. This application will be the next step in social media as it delves even deeper into what technology can do as well as what is beginning to trend within media while keeping everything we know and love though social media today. This database will be useful for users as they make their own AI robust and unique to themselves acting as them when they are not even there.

From last week, I added my DROP TABLE’s to my SQL file to make sure it is not missing again.This week I created more store procedures to populate my table more with data consisting of questions from users. Using this new data, I created a history table which tracked any changes made from the owner of an account based on the responses that their AI generated. These changes were saved in the history table using a trigger that I also created. I then tested my trigger using real data to confirm it works as intended. Finally, I added 2 new queries to my database, which allowed me to answer questions that would be helpful to developers and created charts to help visualize the data, making the data easier to understand for any person wanting to understand.

I am really proud of everything I have learned in the course and how each week prepared me for the next. This project was a huge task and really made me struggle at times when dealing with creating all my tables and their relationships. It was a breath of fresh air after finishing though, as the ERD’s set me up well for creating my database in SQL. I ran into minimal problems because my foundation was solid, and the last few weeks felt very smooth in the transition to SQL. I look forward to continuing to build on my idea and use this database to represent my skills and what I can accomplish given dedication and continued hard work.