Name:

**Table of Contents**

[Project Direction Overview 2](#_Toc73971270)

[Use Cases and Fields 2](#_Toc73971271)

[Structural Database Rules 7](#_Toc73971272)

[Conceptual Entity-Relationship Diagram 9](#_Toc73971273)

[Full DBMS Physical ERD 13](#_Toc73971274)

[Stored Procedure Execution and Explanations 13](#_Toc73971275)

[Question Identification and Explanations 16](#_Toc73971276)

[Query Executions and Explanations 16](#_Toc73971277)

[Index Identification and Creations 19](#_Toc73971278)

[History Table Demonstration 22](#_Toc73971279)

[Data Visualizations 24](#_Toc73971280)

[Summary and Reflection 25](#_Toc73971281)

# Project Direction Overview

I would like to develop a mobile application called InstaCare. The primary purpose of this app and database is to connect the people seeking psychological help with a Psychiatrist. This app and database will help the people who are in severe need of help, because every year, thousands of people end up losing their lives to depression and other mental health issues just because they do not receive timely access to healthcare.

It will contain patient’s information, it’s medical history, information regarding different doctors that are available etc. To get started, patients will input their profile details within the app when they sign up, such as their name, their age, as well as location. After signing up, patients will be able to search and schedule an appointment with a doctor for a 30 minute consultation.

Here is an example of how someone will use this app. Jake has been feeling depression and anxiety for a while but now, it has increased so much that he is unable to even step outside his home. He downloads the app and creates an account on it. He is then able to look up different doctors that are available at that time with an open availability. He schedules an appointment to see the doctor right away. He meets the doctor online, tells the doctor about his issue, and the doctor evaluates him, prescribes him some medicines, and asks him to follow up.

I am interested in this project because I have been dealing with mental health issues for around a decade now. I was diagnosed with OCD and anxiety. I was privileged enough to receive timely consultation from a doctor, but not everyone is able to do so, and I want to change that. This is why, with InstaCare, people would not have to worry about calling each office to check if they have appointments available, or if they accept uninsured people. This service would also be free for everyone, and the app would only allow doctors to offer their service voluntarily so that people who are not able to afford the service can also seek mental health

# Use Cases and Fields

One usage of the database related to this app is when a person installs the application and signs up for a new account.

1. **Account Signup/Installation Use Case :**

1. The patient (user) installs the app via the InstaCare website or the app store.

2. The application asks them to create an account when its first run.

3. The patient enters their information and the account is created in the database.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| PatientID | This field stores the customer id which is unique and given to each patient when they sign up. | It’s needed so that each patient can be uniquely identified. |
| First\_Name | This is the first name of the account holder. | This field is necessary so that if the doctor needs to prescribe medicines, he or she would know what name to put on the prescription. |
| Last\_Name | This is the last name of the account holder. | This field is necessary so that if the doctor needs to prescribe medicines, he or she would know what name to put on the prescription. |
| Gender | This is the gender of the account holder | This field is needed to determine the gender of the account holder. |
| Patient\_DOB | This field will store the date of birth of the patient. | This field is needed so that the doctor knows the age of the patient that is being seen. |

Another usage of the database related to this app is when a patient (user) searches for a doctor in the app.

1. **Doctor Lookup :**

1. The person searches for a particular doctor in the search bar.

2. Multiple doctors with the same name show up.

3. The patient selects a doctor to read more details about the doctor such as average ratings etc.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| DrID | This field stores the doctor id which is unique and given to each provider when they sign up to provide their services through the app. | It’s needed so that each healthcare provider can be uniquely identified. |
| Dr\_First\_Name | This field stores the first name of the doctor. | This field is necessary so that the person can look up doctors in the app. |
| Dr\_Last\_Name | This field stores the last name of the doctor. | This field is necessary so that the person can look up doctors in the app. |
| Dr\_Gender | This field stores the gender of the doctor | This field is needed in case the patient wants to see a doctor of a specific gender only. |
| Dr\_Race | This field stores the race of the doctor | This field is needed in case the patient does or does not want to see doctors who are the same race as him, out of embarrassment. |
| Languages | This field stores the name of the languages spoken by the doctor | This field is necessary if the patient wants to see a doctor who speaks a certain language. |
| Years\_Exp | This field stores the years of experience the doctor has in the medical field. | This field is needed in case the patient wants to decide which doctor they want to be seen by, based on the years of experience. |

Another usage of the database related to this app is when a patient schedules an appointment with the doctor.

1. **Scheduling an Appointment Use Case :**

1. The patient selects the doctor which they want to see.

2. The patient selects the date and time for the appointment.

3. The patient selects the appointment medium, which could be either audio appointment or video appointment.

4. Once the time, date and the appointment medium is selected, the patient selects “OK” and the appointment is scheduled.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| DrID | This field stores the doctor id which is unique and given to each provider when they sign up to provide their services through the app. | It’s needed so that each healthcare provider can be uniquely identified. |
| Dr\_First\_Name | This field stores the first name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Dr\_Last\_Name | This field stores the last name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Appointment\_Date | This field stores the date of the appointment | This field is necessary so that the app can send a reminder to the patient for the appointment |
| Appointment\_Time | This field stores the time for the appointment | This field is needed so that the app can send a reminder to patient for the appointment. |
| Audio\_length | This field stores the length of the audio appointment | This field is needed to keep track of the time duration for appointments. |
| Video\_length | This field stores the length of the video appointment | This field is needed to keep track of the time duration for appointments. |

Another usage of the database related to this app is when a patient looks up his/her previous medical consultations that have taken place through the app to look up the previous medications etc.

1. **Consultation History Use Case:**

1. The patient looks up the details of the previous medical consultations he has had through the app by selecting the tab “My Historical Consultations”.

2. The app provides the patient with the list of all the previous consultations.

3. The patient selects a previous consultation and reads the prescription provided by the doctor during that consultation.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| DrID | This field stores the doctor id which is unique and given to each provider when they sign up to provide their services through the app. | It’s needed so that each healthcare provider can be uniquely identified. |
| Dr\_First\_Name | This field stores the first name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Dr\_Last\_Name | This field stores the last name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Appointment\_Date | This field stores the date of the appointment | This field is needed so that the patient and doctor both know what day the appointment is. |
| Appointment\_Time | This field stores the time of the appointment | This field is needed so that the patient and doctor both know what time the appointment is. |

Another usage of the database related to this app is when a patient writes a review for a particular doctor that he/she has had medical consultation from.

1. **Leaving a Review UseCase:**

1. . The patient looks up the previous medical consultations he has had through the app by selecting the tab “My Historical Consultations”.

2. The a pp provides the patient with the list of all the previous consultations.

3. The patient selects a previous consultation and clicks on “Write a Review”, and submits the review.

|  |  |  |
| --- | --- | --- |
| **Field** | **What it Stores** | **Why it’s Needed** |
| DrID | This field stores the doctor id which is unique and given to each provider when they sign up to provide their services through the app. | It’s needed so that each healthcare provider can be uniquely identified. |
| Dr\_First\_Name | This field stores the first name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Dr\_Last\_Name | This field stores the last name of the doctor. | This field is necessary so that the patient can look up doctors in the app. |
| Review\_Note | This field stores the review for a particular doctor, given by the patient. | This field is necessary because it stores the reviews for different doctors which are accessible by the different patients (users) which helps them in deciding if they want to book an appointment with a particular doctor or not. |
| Review\_Date | This field stores the date when a particular review was given. | This field is necessary so that the patient can be made aware of how new or how old a particular review is. |

# Structural Database Rules

Update your list of structural database rules to describe the new history table.

1. Each **account** must be associated with only one **patient** (user); Each patient may have multiple accounts.

This rule indicates that every patient may have many accounts associated with it, since a patient is free to make as many accounts as possible. At the same time, each account must belong to only one patient, since the account holds private data such as medications and diagnosis. I made it mandatory for an account to be associated with a patient because in order for a patient to use the app, the patient must have an account.

1. Each patient may schedule more than one appointments; Each **appointment** must only belong to one patient.

This rule indicates that every patient may schedule more than one appointment. At the same time, each appointment must belong to only one patient, since an appointment cannot be divided to see more than one patient.

1. Each doctor may have more than one appointment; Each appointment must only belong to one doctor.

This rule indicates that every doctor may have more than one appointment, since a doctor may work for multiple hours in a day, so he/she may have multiple appointments scheduled by different patients. At the same time, each appointment must only have one doctor, since an appointment cannot be divided between more than one doctor.

1. Each **doctor** may have more than one **review**. Each review must only belong to one doctor.

This rule indicates that every doctor may have more than one review associated with him/her, since any patient is eligible to leave a review, provided the patient was seen by the doctor via the app. At the same time, each review must only belong to one doctor. I made it mandatory for each review to only be associated with one doctor because the app does not allow the option to automatically apply the review to another doctor, provided the patient saw two different doctors through the app. The patient would need to go to the second doctor’s profile and manually type the review again, or copy the review from the first doctor’s profile, and paste it into the second doctor’s profile.

1. Each patient may have more than one **prescriptions**; Each prescription must only belong to one patient.

This rule indicates that every patient may have more than one prescription provided to him/her. This is because the patient may see the same doctor twice for two different issues, and the doctor may write more than one prescription as a result. Another reason for this is that the doctor may see two different doctors and they each write a prescription for the patient, in which case the patient will have more than one prescription. At the same time, each prescription must belong to only one patient. I made it mandatory for a prescription to be associated with only one patient because a prescription is non-transferable, and cannot be used by a different patient.

1. Each patient may provide more than one review; Each review must only be provided by one patient.

This rule indicates that every patient may provide more than one review. I made this to be optional because a patient may see more than one doctor. At the same time, each review must only be provided by one patient. I made it mandatory for each review to only be provided by one patient because a patient is only eligible to write a review if he/she scheduled an appointment and was seen by the doctor. Since an appointment cannot be divided in order to see more than one patient as per rule no. 2, the review must only be provided by the patient who was seen by the doctor.

1. An appointment is an audio appointment or a video appointment.

This rule indicates that each

appointment which is scheduled must be an audio appointment or a video appointment. I have made it mandatory for each appointment to be an audio or a video appointment because other than audio or video, there is no third way in the app to provide consultation to the patients. The relationship is short but to the point, and is complete. Since the appointment is either an audio or a video appointment, it is disjoint.

1. Each doctor may see more than one patient; A patient may see more than one doctor.

This rule indicates that a doctor can see more than one patient in the app. I made this to be optional because it is possible that the doctor may not receive any more appointments. Similarly, a patient can see more than one doctor if they wish to. I made this to be optional because a patient may feel satisfied with the first doctor that they see, and may decide to only consult with that particular doctor for all their issues.

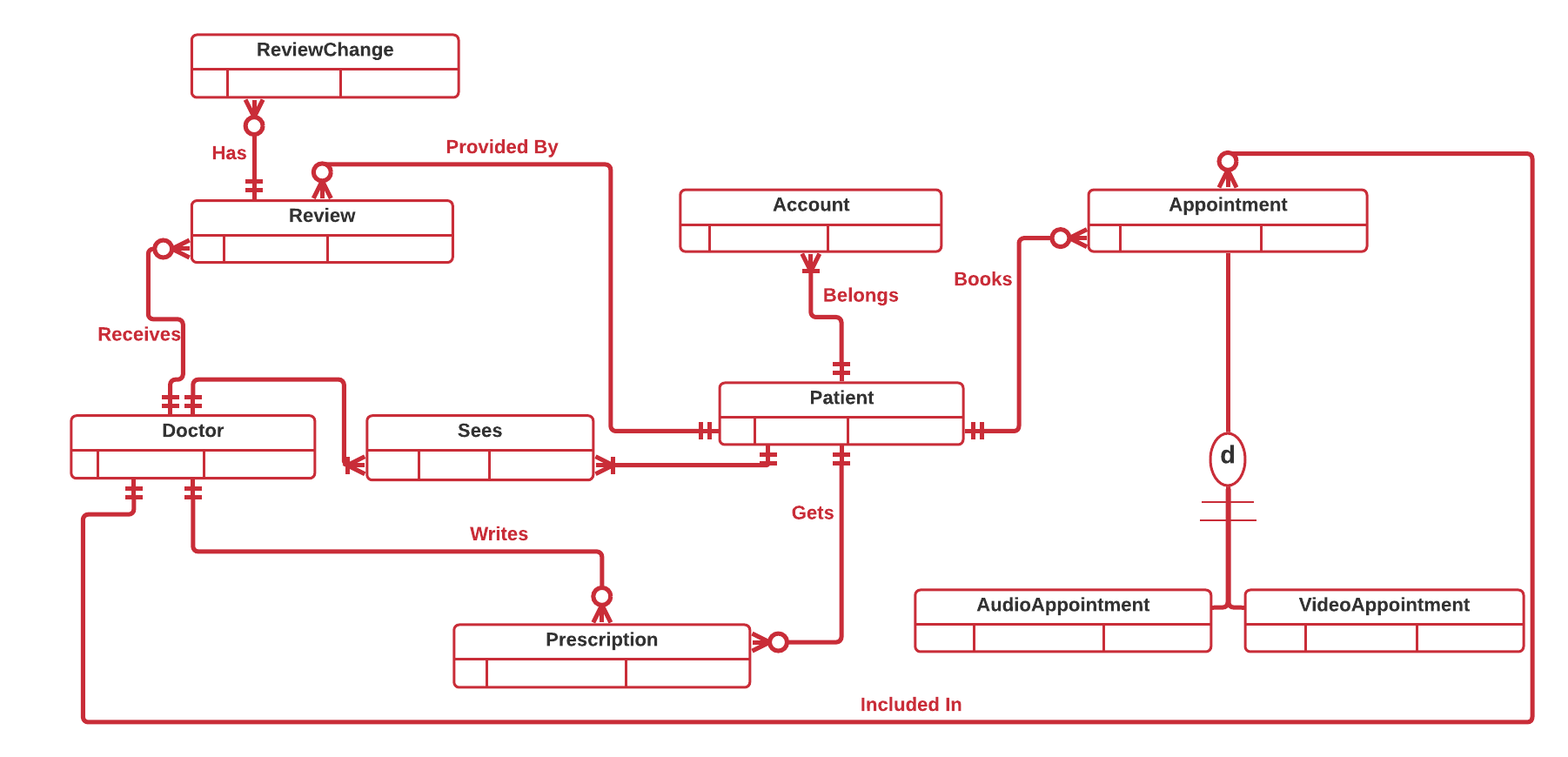
1. A doctor may write more than one prescription; A prescription must only be written by one doctor.

This rule indicates that a doctor may write more than one prescription. I have made this to be optional because it is possible that the doctor may not need to write any prescription or only one prescription. On the other side, a prescription must only be written by one doctor because on the prescription, there must only be one doctor’s name. I have made it mandatory because a prescription cannot be provided without a doctor’s name.

1. A review can have many changes; A change must only belong to one review

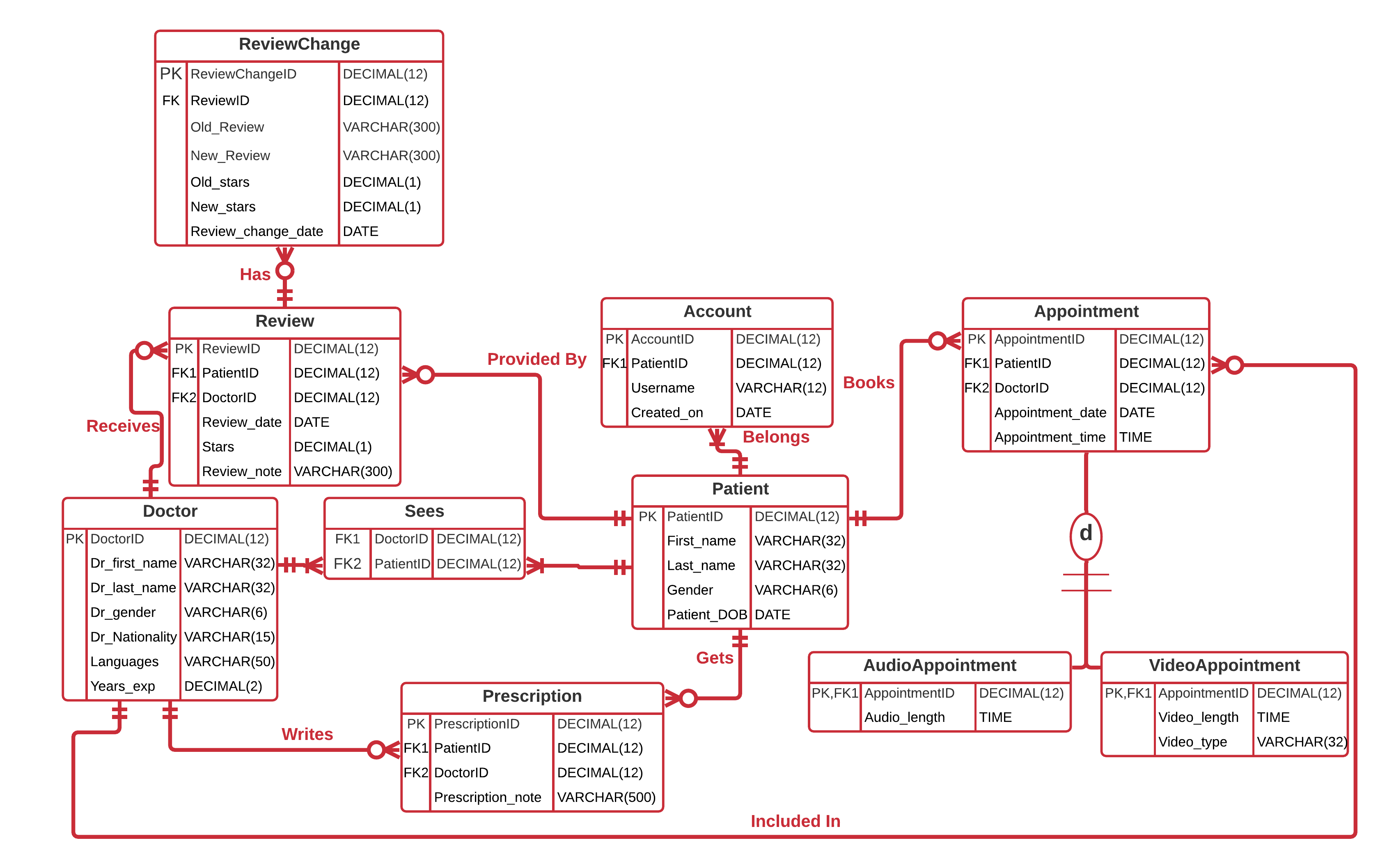
This rule indicates that a review may have no change or as many changes as the user wishes. I have made this optional because not all reviews need to be updated. On the other side, a review change must only belong to one review and it is mandatory for it to only belong to one review because app does not provide a functionality to update multiple reviews at the same time or apply the same update to multiple review within the app.

# Conceptual Entity-Relationship Diagram



|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** |
| Account | Username | VARCHAR (12) | Username is what patients use to sign in to the app. It can be 12 characters max. |
| Account | Created\_On | DATE | This attribute determines when an account was created. |
| Patient | First\_name | VARCHAR(32) | This is the first name of the patient, and must not go above 32 characters, which seems reasonable. |
| Patient | Last\_name | VARCHAR(32) | This is the last name of the patient, and must not go above 32 characters, which seems reasonable. |
| Patient | Gender | VARCHAR(6) | This is the gender of the patient, and cannot be greater than 6 characters. |
| Patient | Patient\_DOB | DATE | This is the date of birth of the patient, and DATE data type helps in inserting the values accurately. |
| Prescription | Prescription\_note | VARCHAR(500) | This attribute stores all the notes and prescriptions written by the doctor in each appointment. The maximum characters it can hold is 500, which seems reasonable. |
| Sees | DoctorID | DECIMAL(12) | “Sees” is a bridging entity between the tables “Patient” and “Doctor”. DoctorID is a foreign key for Patient table. |
| Sees | PatientID | DECIMAL(12) | “Sees” is a bridging entity between the tables “Patient” and “Doctor”. PatientID is a foreign key for Doctor table. |
| Doctor | Dr\_First\_name | VARCHAR(32) | This is the first name of the doctor, and cannot be more than 32 characters, which seems reasonable. |
| Doctor | Dr\_Last\_name | VARCHAR(32) | This is the last name of the doctor, and cannot be more than 32 characters, which seems reasonable. |
| Doctor | Dr\_Gender | VARCHAR(6) | This is the gender of the doctor and it cannot be more than 6 characters. This attribute is important because some patients only feel comfortable talking to a doctor of a specific gender. |
| Doctor | Dr\_Nationality | VARCHAR(15) | This is the nationality of the doctor. This attribute is important because some patients only feel comfortable talking to a doctor of their own nationality, or sometimes, they only want to talk to a doctor who is not from the same country as them. |
| Doctor | Languages | VARCHAR(50) | These are the names of the languages the doctor can speak, and the maximum characters it can hold is 50, which seems reasonable. This attribute is important incase someone wants to only speak to a doctor who speaks a certain language. |
| Doctor | Years\_exp | DECIMAL(2) | This is the year(s) of experience the doctor has. It can only hold up to 2 digits. |
| Review | Review\_date | DATE | This is the date when the review was given. DATE data type helps in inserting the values accurately. |
| Review | Stars | DECIMAL(1) | This attribute holds the number of stars a doctor received. Maximum stars than can be given are 5. |
| Review | Review\_note | VARCHAR(300) | This attribute holds the review given by the patient. It can hold up to 300 characters, which seems reasonable. |
| Appointment | Appointment\_Date | DATE | This attribute holds the date of the appointment, so that the patient can be sent a reminder a day before the appointment date. DATE data type helps in inserting the values accurately. |
| Appointment | Appointment\_time | TIME | This attribute holds the time of the appointment, so that the patient can be sent a reminder a day before the appointment date. TIME data type helps in inserting the values accurately. |
| AudioAppointment | Audio\_length | TIME | This attribute holds the duration of an audio appointment. It is a subtype of the table “Appointment”. TIME data type helps in inserting the values accurately. |
| VideoAppointment | Video\_length | TIME | This attribute holds the duration of an audio appointment. It is a subtype of the table “Appointment”. TIME data type helps in inserting the values accurately. |
| VideoAppointment | Video\_type | VARCHAR(32) | This attribute holds the type of video software to be used for the video appointment. It can carry up to 32 characters. |
| ReviewChange | Old\_Review | VARCHAR(300) | This attribute holds the old review provided by the patient and can carry up to 300 characters. |
| ReviewChange | New\_Review | VARCHAR(300) | This attribute holds the new review provided by the patient and can carry up to 300 characters. |
| ReviewChange | Old\_Stars | DECIMAL(1) | This attribute holds the old rating and can only be 1 character. |
| ReviewChange | New\_Stars | DECIMAL(1) | This attribute holds the new rating and can only be 1 character. |
| ReviewChange | Review\_Change\_Date | DATE | This attribute holds the date when the review was updated. DATE data type helps in inserting the values accurately. |

# Full DBMS Physical ERD



All of the tables are normalized because each column had only one value for each row in the table which met the requirement for 1NF, every non-prime attribute of the relation is dependent on the whole of every candidate key which met the requirement for 2NF, none of the non key attribute in the tables is transitively dependent on the primary key, which meets the requirement for 3NF, and super key is usually only primary key or combination of primary key and foreign key, which is there in the ERD as well, which meets the requirements for BCNF. The additional entities under Appointment are AudioAppointment and VideoAppointment, each of

which have a primary and foreign key of AppointmentID which reference the primary key of Appointment.

The Review entity has two foreign keys. FK1 is referencing the primary key of Patient, and FK2 is referencing the primary key of Doctor entity.

With these additional mappings, this DBMS physical now has all of the relationships in the conceptual ERD.

# Stored Procedure Execution and Explanations

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**EXPLAINATION:** The two stored procedures I created are about booking an appointment and leaving a review. For both of them, I wrote my parameters for all columns. Then I, begin my transaction for this procedure and executed it by providing the values for some as a query which I wanted both the procedures to insert.

# Question Identification and Explanations

**Q1**. How many times was the medicine “Fluvoxamine” prescribed to patients?

**EXPLAINATION:** This question is useful because company earns a small commission every time it prescribes this medicine to any of the patients, which helps in keeping the app functional, so keeping track of how many times this medicine was prescribed is important. The doctors are not made aware of this to prevent overuse of “Fluvoxamine” as a prescribed medicine.

**Q2.** Provide the list of the first name, last name and id of doctors whose consultation appointments ran less than 5 minutes, along with the review left by the patient.

**EXPLAINATION:** Customer Service Dept. noticed a review about a doctor that she rushed through the entire audio appointment in under 5 minutes and did not pay attention to what the patient was asking. Management wants to know the name of the doctor who provided consultation so that they can notify the doctor of this complaint. Management also wants the id of the doctor to be included in case there is more than one doctor of the same name. It would be possible to return many doctors of the same name, but we would be able to narrow it down by the patient id and the number of stars received because company’s policy is that all doctors who receive less than 4 stars are to be notified, to make customer service better.

**Q3.** Create a view which provides the count of the new accounts that are created in the last 30 days.

**EXPLAINATION:** This view is needed so that the management knows how popular the application is, and how many new users is it attracting.

# Query Executions and Explanations

**Query 1.**

select count(doctor.doctorid) as medicine\_count

from patient

join sees

on sees.patientid = patient.patientid

join doctor

on sees.doctorid = doctor.doctorid

join appointment

on appointment.patientid = patient.patientid

join prescription

on patient.patientid = prescription.patientid

where prescription\_note like '%Fluvoxamine%;

Graphical user interface, application, Word

Description automatically generated

**EXPLAINATION:** Query found that the medicine “Fluvoxamine” was prescribed three times in the last 30 days. The result was found by joining the table “Patient” with “Doctor” through the bridging table “Sees”. It was then joined with the table “Appointment” in order to filter for the last 30 days using the column “appointment\_date”. The query was further joined with the table “Prescription” so we can filter the data using the column “Prescription\_note” which contains the medicine name “Fluvoxamine” anywhere in the “Prescription\_note” column.

**Query 2:**

select doctor.doctorid, dr\_first\_name, dr\_last\_name, audio\_length

from doctor

join appointment

on doctor.doctorid = appointment.doctorid

join audioAppointment

on appointment.appointmentid = audioAppointment.appointmentid

where audio\_length < '00:5:00';

Graphical user interface, text, application, email

Description automatically generated

**EXPLAINATION:** The output shows that there was 1 doctor who seemed to have “rushed” through her appointments twice because according to the patient, the audio appointment lasted even less than 5 minutes.

ID of the doctor, name, and audio appointment duration were queried. The tables “Doctor”, super type “Appointment”, as well as the sub type “Audioappointment” were joined together to retrieve the results. WHERE clause was use to filter the data to only include the details where the audio appointment duration or video appointment duration was less than 5 minutes.

**Query 3.**

create view newaccounts as

select count(accountid) as new\_accounts\_count

from account

where created\_on >= DATEADD(day, -30, getdate());

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**EXPLAINATION:** The output shows there were 15 new accounts created in the last 30 days. We got this output by creating a view and naming it as “newaccounts”. Within the view, we did a count on the column “accountid” because all the account ids are unique, and named it “new\_accounts\_count”. In the WHERE clause, we added a condition which would filter to only provide us data for the last 30 days.

# Index Identification and Creations

|  |  |  |
| --- | --- | --- |
| **Primary Keys Columns** | **Unique/Non-Unique** | **Description** |
| Account.AccountID | Unique | This is the primary key of the table “Account” |
| Patient.PatientID | Unique | This is the primary key of the table “Patient” |
| Prescription.PrescriptionID | Unique | This is the primary key of the table “Prescription” |
| Review.ReviewID | Unique | This is the primary key of the table “Review” |
| Appointment.AppointmentID | Unique | This is the primary key of the table “Appointment” |
| Doctor.DoctorID | Unique | This is the primary key of the table “Doctor” |
| AudioAppointment.AppointmentID | Unique | This is the primary key of the table “AudioAppointment” |
| VideoAppointment.AppointmentID | Unique | This is the primary key of the table “VideoAppointment” |

|  |  |  |
| --- | --- | --- |
| **Foreign Keys Columns** | **Unique/Non-Unique** | **Description** |
| Account.PatientID | Non-Unique | This foreign key in the Account table references the Patient table. The index is non-unique since many accounts can be created by the same patient. |
| Sees.PatientID | Non-Unique | This foreign key in the Sees table references the Patient table. The index is non-unique since a patient can be seen many times. |
| Sees.DoctorID | Non-Unique | This foreign key in the Sees table references the Doctor table. The index is non-unique since a doctor can see the same patient many times. |
| Review.PatientID | Non-Unique | This foreign key in the Review table references the Patient table. The index is non-unique since many reviews can be provided by the same patient. |
| Review.DoctorID | Non-Unique | This foreign key in the Review table references the Doctor table. This index is non-unique since many reviews can be provided to the same doctor. |
| Prescription.PatientID | Non-Unique | This foreign key in the Prescription table references the Patient table. This index is non-unique since many prescriptions can be provided to the same patient |
| Prescription.DoctorID | Non-Unique | This foreign key in the Prescription table references the Doctor table. This index is non-unique since many prescriptions can be provided by the same doctor. |
| Appointment.PatientID | Non-Unique | This foreign key in the Appointment table references the Doctor table. This index is non-unique since many appointments can be scheduled by the same patient. |
| Appointment.DoctorID | Non-Unique | This foreign key in the Appointment table references the Doctor table. This index is non-unique since many appointments can be scheduled for the same doctor. |

|  |  |  |
| --- | --- | --- |
| **Query Driven Columns** | **Unique/Non-Unique** | **Description** |
| Account.Created\_on | Non-Unique | This is a field with DATE data type. Fields with DATE or TIME data type such as DOB, Appointment time or date, date an account was created on is commonly used by analysts to limit their result in order to carry out their analysis and reports. It is non-unique because many accounts can be created in a day. |
| Review.Stars | Non-Unique | Ratings of a doctor can be used in WHERE clause to only look for doctors who have high ratings etc. It is non-unique because many doctors can have the same ratings. |
| Appointment.Appointment\_date | Non-Unique | This is a field with DATE data type. Fields with DATE or TIME data type such as DOB, Appointment time or date, Appointment\_date are commonly used by analysts to limit their result in order to carry out their analysis and reports. It is non-unique because many appointments can be created on the same date. |

**Query 1.** Following is a single table query that retrieves the date related to accounts that were created in the month of September.

Select accountid, username

from account

where account.created\_on >= '2020-09-01' and account.created\_on <= '2021-09-30';

Two columns are accessed in this query – Account id and the username– but only created\_on is in the WHERE clause. Therefore we would index the created\_on column. We would create this index as a non‐unique index.

**Query 2.** The following query retrieves the doctor id, doctor’s name who received less than 4 stars.

select doctor.doctorid, dr\_first\_name,dr\_last\_name

from doctor

join review

on doctor.doctorid = review.doctorid

where stars < 4

Three columns are accessed in this query – Account id, doctor’s first name, and his last name– but only stars is in the WHERE clause. Therefore we would index the stars column. We would create this index as a non‐unique index.

**Query 3.** The following query retrieves the patient id, patient’s first name and last name of all the patients who booked an appointment for the month of September.

Select patient.patientid, first\_name, last\_name

from patient

join appointment

on patient.patientid = appointment.patientid

where appointment\_date between '2021-09-01' and '2021-09-30';

Three columns are accessed in this query – patientid, first\_name, and last\_name– but only appointment\_date is in the WHERE clause. Therefore we would index the appointment\_date column. We would create this index as a non‐unique index.

# History Table Demonstration

Company’s customer service dept. forwards the complains and bad reviews to the specific doctors who in turn follow up with the patients to improve the experience that they had. Patient has the choice to update his review if they have already provided one. If they update their review, the history is captured in the history table called Reviewchange. It captures the data from the review table anytime a review is updated.

It contains the reviewchangeid as the primary key. The reviewed which is the id of the original review. Another column is Old\_stars which captures the previous ratings. The column new\_stars captures the updated ratings. Old\_review lists out the previous review, and new\_review lists out the new review. Review\_change\_date provides the date when the review was updated.

I create a trigger called trigger\_insert\_review on the review table. I declare the parameters for

Patientid, doctorid and old review and provide them the data type. Then I set the columns old\_stars and old review = to the values which were deleted by SQL when the update occurred, if they were deleted. If not, I set old stars and review equal to the stars and review where the ids are equal to the parameter defined earlier. Then I use sequence to make sure every time there is a new record in reviewchange table, there is a new review sequence id which must be used, in order to keep the field unique. I use the getdate() function to update the review change date field which fetches the current date.

Afterwards, I update the review for reviewed 9, and then take screenshots of the review table and the history table to demonstrate that the update occurred successfully.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# Data Visualizations

The query that I created to use for the first data visualization using a bar chart is as follows:

select review.stars, count(\*) as total\_stars

from review

group by stars;

The data visualization tells us that the company has mostly received positive reviews from the users. Of all the reviews, it is only twice that the user provided only 1 star, whereas of all the 21 reviews, 18 are either 4 or 5 star reviews. Interestingly, no user (patient) provided 2 star as a review, which is good for the company since a 2 star review is considered to be more on the negative side. This also tells me that there are in total 3 users who are unsatisfied with the services that they received, so the company must dig further into the experience these customers had to see what caused them to have a bad experience.

Chart

Description automatically generated

# 

**2.** For the 2nd visualization, I use the following query:

select datename(month, review\_change\_date) as month\_name, count(1) as reviews\_updated

from reviewchange

group by datename(month, review\_change\_date)

This query provides me the info that how many people updated their rating each month from their original rating. The data visualization and the increasing height of the bar chart clearly shows that the users have increasingly updated their reviews each month.

Chart

Description automatically generated

# Summary and Reflection

The primary purpose of this app is to provide help to those people who are dealing with mental health issues, and do not know where to go to seek help. When I started working on this iteration, I came to this realization that there are so many different things that go into making an application, no matter how simple or complex the app is. I am still unable to grasp the scale of this application and how many more fields and how many tables it would require, since I am just starting my work on this project.

Four weeks ago, I had made up my mind on what kind of application I want to develop and I had an idea on the kind of data that will go into it. This week, after the professor’s presentation of 2nd iteration, and after having a one on one meeting with my facilitator, I was able to come up with the accurate and precise Business Structural Rules which are vital to any database development. From those rules, I was able to derive the entities which I would need to come up with an ERD. Once I was able to finalize the structural rules, I was able to create an ERD which provided me with some sort of visualization on how my tables would connect with one another once my database goes live.

After the 3rd project iteration overview which was given by the professor, which consisted of understanding what specialization-generalization relationship is and how to map it in our ERD, I came to the realization that I am unable to include a specialization-generalization relationship to my ERD because none of the use cases that I have written over the previous weeks had the material from which I could derive or develop a business rule which could cover any aspect of the specialization-generalization relationship. In order to fix this, I edited the use case # 3 to add another field “AppointmentMedium” which would keep track of each appointment made by the patient in regards to whether it is an audio only appointment or a video appointment. From here, I developed a business rule that an appointment must either be an audio appointment or a video appointment, and then I added this specialization-generalization relationship to my ERD. Once I was able to finish my ERD, I added the appropriate primary and foreign key(s) to the entities. I still think that as we go forward and the project develops further, there might be some changes that would need to be made, such as addition of more tables, use cases etc.

For 4th week, I had to normalize my tables upto BCNF. When I sat down to normalize all the tables, I came to the conclusion that all of the tables are normalized because each column had only one value for each row in the table which met the requirement for 1NF, every non-prime attribute of the relation is dependent on the whole of every candidate key which met the requirement for 2NF, none of the non key attribute in the tables is transitively dependent on the primary key, which meets the requirement for 3NF, and super key is usually only primary key or combination of primary key and foreign key, which is there in the ERD as well, which meets the requirements for BCNF. I decided what attributes I want my database tables to have. From my understanding, the attributes which I have chosen and displayed in my full DBMS physical ERD are to the point and are essential to make this database workable and functional. At the end, I successfully created the tables and sequences in SQL server, following the guidelines provided by the professor.

For 5th week, I had to create and execute at least 2 stored procedure. I created the first stored procedure about booking an appointment through an app, and the 2nd stored procedure was about leaving a review for the doctor. Then I created business driven queries, took the screenshot of the query, as well as copied the queries into the design document. I also created index, which was something I had never done before, but with the help of the explanation document, it became very easy to not only decided which fields to create index on, but also how to create them in SQL.

For this week, I gave final touches to my project, and created a history table, a trigger to maintain the history table, as well as some data visualization. Overall, the final week was not so difficult because every task in this week was already done before in labs.

I hope that this database will not only serve as a mock-up for a full-fledged application which will help the people in need, but also create awareness about the mental health crisis this country is facing.