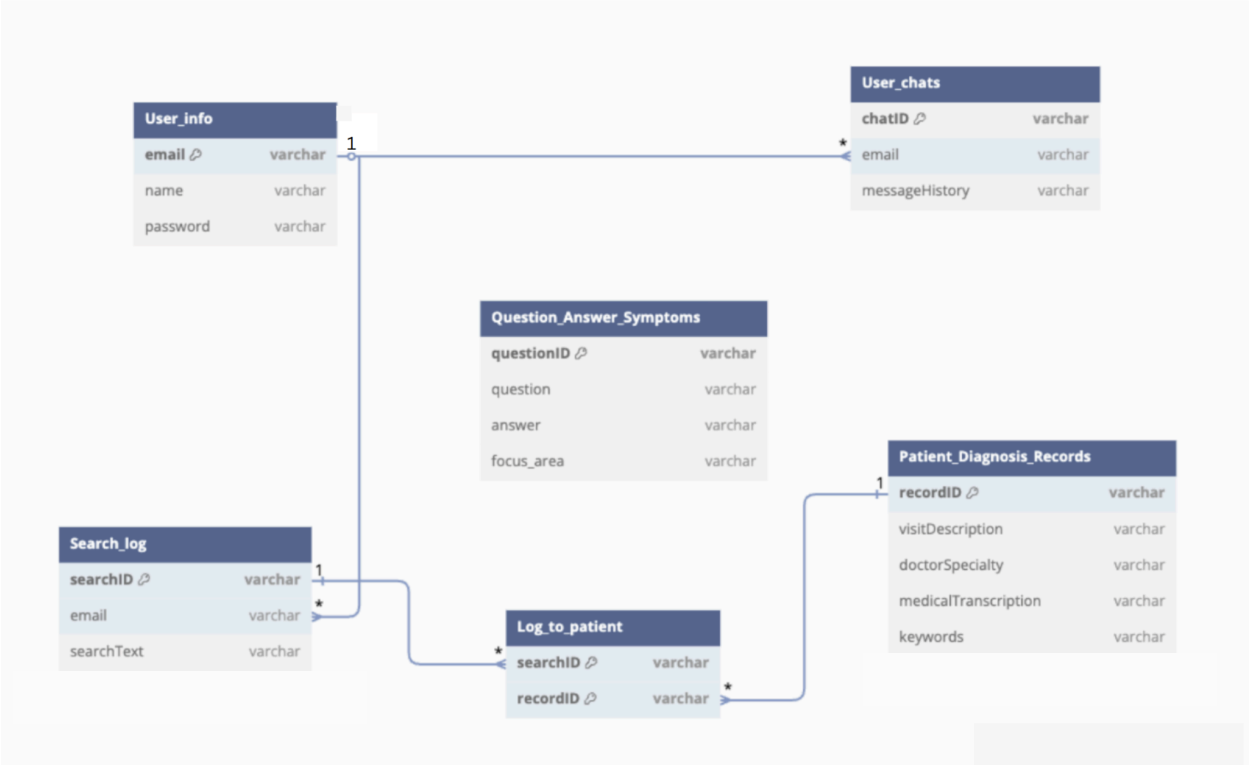
Stage 2: Conceptual and Logical Database Design

Aditya Saxena (saxena11), Aryamaan Sen (as202), Aditya Raju (raju7), Nischay Singh (nischay2)



Entities

1. **User_Info** (User Account Data)

- Assumption: Each user has a unique account that stores their interaction history.
- Why a separate entity?
    - A user will have many data attributes linked to their account such as their email, password, linked chats, queries, etc.
- Attributes:
    - Email (Primary Key) - VARCHAR
    - Name - VARCHAR
    - Password - VARCHAR (Salted and Hashed)

2. **User_Chats** (Stores Chat Sessions)

- Assumption: Each chat session is independent and stored separately.
- Why a separate entity?
    - A user will have one personal chat (1-1 relationship).
    - Storing chat sessions separately allows us users to revisit old conversations
- Attributes:
    - ChatID (Primary Key) - VARCHAR
    - Email (Foreign Key -> Users_Info) - VARCHAR
    - MessageHistory -> VARCHAR list of message and responses

3. **Patient_Diagnosis_Records** (Medical Records from Dataset)

- Assumption: Each record represents a patient's diagnosis history.
- Why a separate entity?
    - The dataset contains structured patient records, so it's essential to store them in an entity.
    - Many users may do multiple queries to multiple patient records for symptom analysis (M-M).
- Attributes:
    - RecordID (Primary Key) - VARCHAR
    - VisitDescription - VARCHAR
    - DoctorSpecialty - VARCHAR
    - MedicalTranscription - VARCHAR
    - Keywords - VARCHAR

4. **Question_Answer_Symptoms** (Medical Q&A Dataset)

- Assumption: This entity contains pre-recorded symptom-related questions and answers from a medical database.
- Why a separate entity?
    - Each question-answer pair is predefined and independent of user interactions.
    - A user’s search may match multiple Q&A pairs.
- Attributes:
    - QuestionID (Primary Key)- VARCHAR
    - Question - VARCHAR
    - Answer - VARCHAR
    - FocusArea - VARCHAR

5. **Search_Log** (User-Submitted Search Queries for Patient Data)

- Assumption: Each time a user enters a symptom to search the datasets, it gets logged.
- Why a separate entity?
    - This ensures we track each user’s input separately from each other.
    - Queries can match multiple Q&A symptom entries (M-M).
- Attributes:
    - SearchID (Primary Key) - INTEGER

    - Email (Foreign Key -> User_Info) - VARCHAR
    - SearchText - VARCHAR

6. **Log_To_Patient** (Many to Many Relation)

- Assumption: This entity is used to manage our many-to-many relationship between Search_Log and Patient_Diagnosis_Records.
- Why a separate entity?
    - We require a separate entity for a many-to-many relationship.
- Attributes:
    - SearchID - VARCHAR
    - RecordID - VARCHAR
    - Primary Key is (SearchID, RecordID)

Relationships and Cardinality

Here's how each entity is connected:

1. **User_Info -> User_Chats**
    - 1 User -> Many Chat Instances (1 to Many)
    - A user will be able to make multiple chat sessions.
2. **Email -> Search_Log**
    - 1 User -> Many Queries (1 to Many)
    - Each user has a log of multiple queries they have searched.
3. **Search_log -> Patient_Diagnosis_Records**
    - Multiple Search Queries -> Multiple Patient Records
    - Each query can return multiple records and each record can be associated with multiple queries.
    - This relationship is handled through the entity.

Relational Schema

User_info(Email : VARCHAR(255) [PK], Name: VARCHAR(255), Password: VARCHAR(255))

User_Chats(ChatID: VARCHAR(255) [PK], Email: VARCHAR(20) [FK to User_info.Email], MessageHistory: TEXT(10000))

Patient_Diagnosis_Records(RecordID: VARCHAR(255) [PK], VisitDescription : TEXT(1000), DoctorSpecialty: VARCHAR(255), MedicalTranscription: TEXT(10000), Keywords: TEXT(1000))

Question_Answer_Symptoms(QuestionID: VARCHAR(255) [PK], Question: TEXT(1000), Answer: TEXT(10000), FocusArea: VARCHAR(255))

Search_Log(SearchID: VARCHAR(255), Email: VARCHAR(255) [FK to Users_info.Email], Search_text: TEXT(10000))

Log_to_Patient(SearchID: VARCHAR(255) [FK to Search_Log.SearchID], RecordID: VARCHAR(255) [FK to Patient_Diagnosis_Records.RecordID])

Normalizing Database

We have the following functional dependencies -

1. Email -> Name, Password
2. ChatID -> Email, MessageHistory
3. RecordID -> VisitDescribes, DoctorSpecialty, MedicalTranscription, KeyWords
4. QuestionID -> Question, Answer, FocusArea
5. SearchID -> Email, SearchText

In each of the schemas, the super key on the left is the only column that can uniquely determine the columns on the right.

- In user_info, the name cannot give the password and vice versa since people can have the same password and name as well.
- In User_chats, the email cannot give MessageHistory and vice versa since a single user can have many chats which can have the same message history.
- In Patient_Diagnosis_Records, VisitDescribes, DoctorSpecialty, MedicalTranscription, and keywords cannot uniquely give each other as there can be multiple values for each of them with different values in other columns.
- In Question_Answer_Symptoms, Question, Answer, and FocusArea can all be repeated values so only QuestionID is a superkey for this table.
- In SearchLog, one email can have multiple searches associated with it and the SearchText is also not necessarily unique, so only SearchID is a superkey.
- The functional dependencies involving the table Log_to_Patient are trivial as its only purpose is to facilitate a many-to-many relationship.

Every functional dependency has a super key on the left, which fulfills the requirements for BCNF. Thus, every table in the schema follows BCNF, meaning we do not need to break our tables down any further. This design helps avoid duplicate data and keeps everything accurate while clearly showing how users, chats, medical records, Q&A pairs, and search logs are connected.