Stage 4: Checkpoint 2

Project Report:

Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

We originally planned to build three main components: a patient search engine, a graph visualization of symptom-diagnosis relationships, and a Retrieval-Augmented Generation (RAG) based chatbot. However, during implementation, we removed the chatbot component. Instead, we extended the search engine functionality to allow querying the healthcare question-answer dataset directly. To support this change, we also added a new entity to store search logs for question-answer searches. The visualization and patient search engine components remained as planned.

Discuss what you think your application achieved or failed to achieve regarding its usefulness.

The application successfully achieved the goal of helping users explore healthcare data in an intuitive and informative way. The patient search engine and the keyword graph visualization both make symptom-diagnosis trends easier to understand. By allowing direct queries on the question-answer dataset, we preserved a major part of the intended chatbot functionality in a simpler, faster, and more reliable format. One limitation is that, without a chatbot, the application no longer provides natural language dialogue, which could have been more engaging for users.

Discuss if you changed the schema or source of the data for your application

We did not change the sources of the datasets; we still used the Healthcare Documentation Database and the Healthcare NLP Dataset as originally planned. However, we changed the schema by adding a new entity to store logs of the users' question-answer searches. This addition supports tracking and analyzing user interactions with the question-answer dataset.

Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

In the original ER diagram, we had entities for Users, User Chats, Patient Diagnosis Records, Question-Answer Symptoms, and Patient Diagnosis Search Log. In the final design, we removed the User Chats table (since the chatbot feature was dropped) and added a new Search Log table specifically for tracking question-answer queries. This structure better fits the final application because it focuses on user-driven search rather than maintaining long conversations. A suitable design was one that allowed simpler logging and retrieval of user interactions.

Discuss what functionalities you added or removed. Why?

We removed the chatbot functionality because of the complexity involved in implementing Retrieval-Augmented Generation (RAG) and deploying a language model effectively within the project timeline. Instead, we added functionality to search the question-answer symptom dataset directly and store the search history. This change allowed us to still meet the goal of helping users find information based on their symptoms while simplifying the backend infrastructure.

Explain how you think your advanced database programs complement your application.

We implemented advanced database programs like triggers to automatically check if a password is of length > 8 and if the email is of a valid format when a user signs up. Stored procedures were used to find correlations between common symptoms. These programs made the application more efficient and ensured that user interactions were consistently and accurately recorded, complementing the overall goal of providing meaningful search functionality.

Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Aditya Saxena: A major challenge was optimizing search queries on the patient diagnosis records. Initially, searches were slow because the database was scanning too many rows without efficient indexing. After profiling the queries, we decided to add composite indexes on fields like symptom keywords and diagnosis names, which improved response time. Future teams should plan indexing strategies early on if their datasets involve frequent complex searches.

Aryamaan Sen: Another challenge was designing the search logging system to efficiently track user queries without introducing significant database overhead. Initially, logging each search individually led to performance issues when users conducted multiple rapid queries. We resolved this by optimizing the insertion process and minimizing the size of each log entry. Future teams should pay careful attention to the performance impact of logging mechanisms, especially in applications that expect frequent user interactions.

Aditya Raju: Developing the procedure was quite challenging and it initially did not work as planned. Debugging and fixing the queries to ensure that all keywords had accurate and meaningful results was very difficult and time consuming. It would have been helpful to properly map out the desired functionality of the procedure before coding to ensure a smoother development process. Mapping out the relations, relationships, inputs, and outputs clearly would have saved a lot of debugging and development time.

Nischay Singh: While developing the front-end interface for the search modules, a major challenge was ensuring that search results updated dynamically without requiring full page reloads. Handling asynchronous data fetching and updating the UI smoothly required learning new concepts like React hooks and efficient state management. Future teams should plan to allocate time for learning frontend optimization techniques if they want a seamless user experience.

Are there other things that changed comparing the final application with the original proposal?

Yes, besides removing the chatbot and adjusting the database schema, we also slightly redesigned the frontend. Originally, there were three separate pages for each functionality, but in the final application, we streamlined the UI into two main flows - one for patient record search and another for question-answer search - to make navigation cleaner and more user-friendly.

Describe future work that you think, other than the interface, that the application can improve on

In the future, we could integrate machine learning techniques to recommend relevant symptoms or questions to users based on their previous search history. Another improvement could be adding statistical summaries (e.g., most common symptoms

searched) and providing filters based on different demographics like age or gender to make the tool even more useful for health researchers.

Describe the final division of labor and how well you managed teamwork.

Aditya Saxena: Developed and optimized the patient search engine and backend query functionalities.

Aryamaan Sen: Managed database schema design, built the search logging system, and worked on backend optimizations.

Aditya Raju: Worked on the backend primarily and SQL developments such as the Trigger, advanced queries, and procedures.

Nischay Singh: Helped develop the front-end interfaces for both the patient search and the question-answer search modules.

We managed teamwork well by holding regular check-ins, communicating through group chats, and dividing the work based on each member's strengths. Everyone completed their assigned tasks on time, and we helped each other debug and test critical parts.

Advanced SQL Queries With Isolation Level

1. SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED

   START TRANSACTION READ ONLY

   SELECT DoctorSpecialty, COUNT(*) AS NumRecords

   FROM Patient_Diagnosis_Records

   WHERE RecordID IN (

   SELECT RecordID

   FROM Patient_Diagnosis_Records

   WHERE MedicalTranscription LIKE %s OR Keywords LIKE %s

)

```
        GROUP BY DoctorSpecialty
        HAVING COUNT(*) >= 2
        ORDER BY NumRecords DESC
    COMMIT
```

2.
```
    SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED
    START TRANSACTION READ ONLY
    SELECT TRIM(FocusArea) AS FocusArea,
            COUNT(*) AS NumQuestions,
            MIN(TRIM(Question)) AS ExampleQuestion
        FROM Question_Answer_Symptoms
        WHERE TRIM(FocusArea) IS NOT NULL AND TRIM(FocusArea) != ''
        GROUP BY TRIM(FocusArea)
        HAVING COUNT(*) > 1
        ORDER BY NumQuestions DESC
        LIMIT 10;
    COMMIT
```

Constraints -

We have 2 foreign key constraints in our Patient_Search_Log and QA_Search_Log, which have the user email as a foreign key.

Trigger to check if Email is of valid format and Password is more than 8 characters.

```
DELIMITER //
CREATE TRIGGER validate_user_info_before_insert
```

```
BEFORE INSERT ON User_Info

FOR EACH ROW

BEGIN

   IF NEW.Email NOT LIKE '%@%' OR NEW.Email NOT LIKE '%.com%' THEN

      SET NEW.Email = NULL;

   END IF;

   IF CHAR_LENGTH(NEW.Password) <= 8 THEN

      SET NEW.Password = NULL;

   END IF;

END//

DELIMITER ;
```

Procedure code

```
DELIMITER //

CREATE PROCEDURE GetKeywordCountsForKeyword(IN input_question TEXT)
BEGIN
        DECLARE focus_area TEXT;

        SELECT FocusArea
        INTO focus_area
        FROM Question_Answer_Symptoms
        WHERE QuestionID = (
        SELECT QuestionID
        FROM Question_Answer_Symptoms
        WHERE Question LIKE CONCAT('%', input_question, '%')
        LIMIT 1
        )
        LIMIT 1;
```

```
        SELECT
        k.Keyword,
        COUNT(DISTINCT pdr.RecordID) AS MatchCount,
        COUNT(*) AS TotalOccurrences
        FROM
        (
        SELECT 'fever' AS Keyword UNION ALL
        SELECT 'cough' UNION ALL
        SELECT 'headache' UNION ALL
        SELECT 'nausea' UNION ALL
        SELECT 'fatigue' UNION ALL
        SELECT 'pain' UNION ALL
        SELECT 'dizziness' UNION ALL
        SELECT 'rash' UNION ALL
        SELECT 'vomiting' UNION ALL
        SELECT 'diarrhea'
        ) AS k
        JOIN
        Patient_Diagnosis_Records pdr
        ON
        pdr.Keywords LIKE CONCAT('%', k.Keyword, '%')
        WHERE
        pdr.Keywords LIKE CONCAT('%', focus_area, '%')
        GROUP BY
        k.Keyword
        ORDER BY
        MatchCount DESC;
END //

DELIMITER ;
```