

ChatDB – A Tool to Query Database Systems with SQL and NoSQL

[GitHub Code Link](#)

1. Introduction

ChatDB is an interactive tool designed to help users explore, query, and interact with databases using natural language. The primary goal is to simplify database querying for both SQL and NoSQL systems by allowing users to input queries in natural language, receive sample queries, and execute those queries in the respective database systems. Unlike traditional query systems, ChatDB not only interprets user input but also processes and returns database results in a user-friendly interface.

Major features include:

- **Data Exploration:** Users can explore database tables, view attributes, and sample data.
- **Sample Query Generation:** ChatDB suggests various SQL and NoSQL queries, such as `GROUP BY`, `JOIN`, and aggregations, based on templates.
- **Natural Language Processing:** Users can input questions in natural language, and ChatDB matches patterns to generate appropriate database queries.
- **UI:** Queries generated or provided by users will be directly executed in the database, with results displayed in the UI.

Our ChatDB project uses a healthcare dataset stored in MySQL and MongoDB to demonstrate its capabilities. This dataset includes structured SQL tables (patients, hospitals, insurance) and flexible NoSQL collections (medications, exams). Initially, we planned to use separate datasets for SQL and NoSQL, but challenges in scalability led to using a shared dataset.

2. Planned Implementation

From the project proposal, the following steps were planned:

1. **Database Setup:**
 - Configure MySQL for SQL operations and MongoDB for NoSQL operations.
 - Use separate datasets tailored to each database system.
2. **Dataset Collection and Preprocessing:**
 - Gather and preprocess datasets for SQL and NoSQL use cases.
 - Structure data for MySQL tables and flexible MongoDB collections.

3. Query Pattern Mapping:

- Define templates for common queries like **SELECT**, **GROUP BY**, **WHERE**, and **JOIN**.
- Use tokenization and regex to transform natural language input into query syntax.

4. Sample Query Generation:

- Generate sample queries dynamically using query templates, allowing users to explore query types like aggregations and joins.

5. Model Integration:

- Use NLP techniques, such as Named Entity Recognition (NER), to detect query patterns in user input.
- Integrate an RNN model for classification and query generation.

6. User Interface Development:

- Design an interactive Streamlit-based UI for exploring databases, generating queries, and displaying results.

7. Query Execution and Testing:

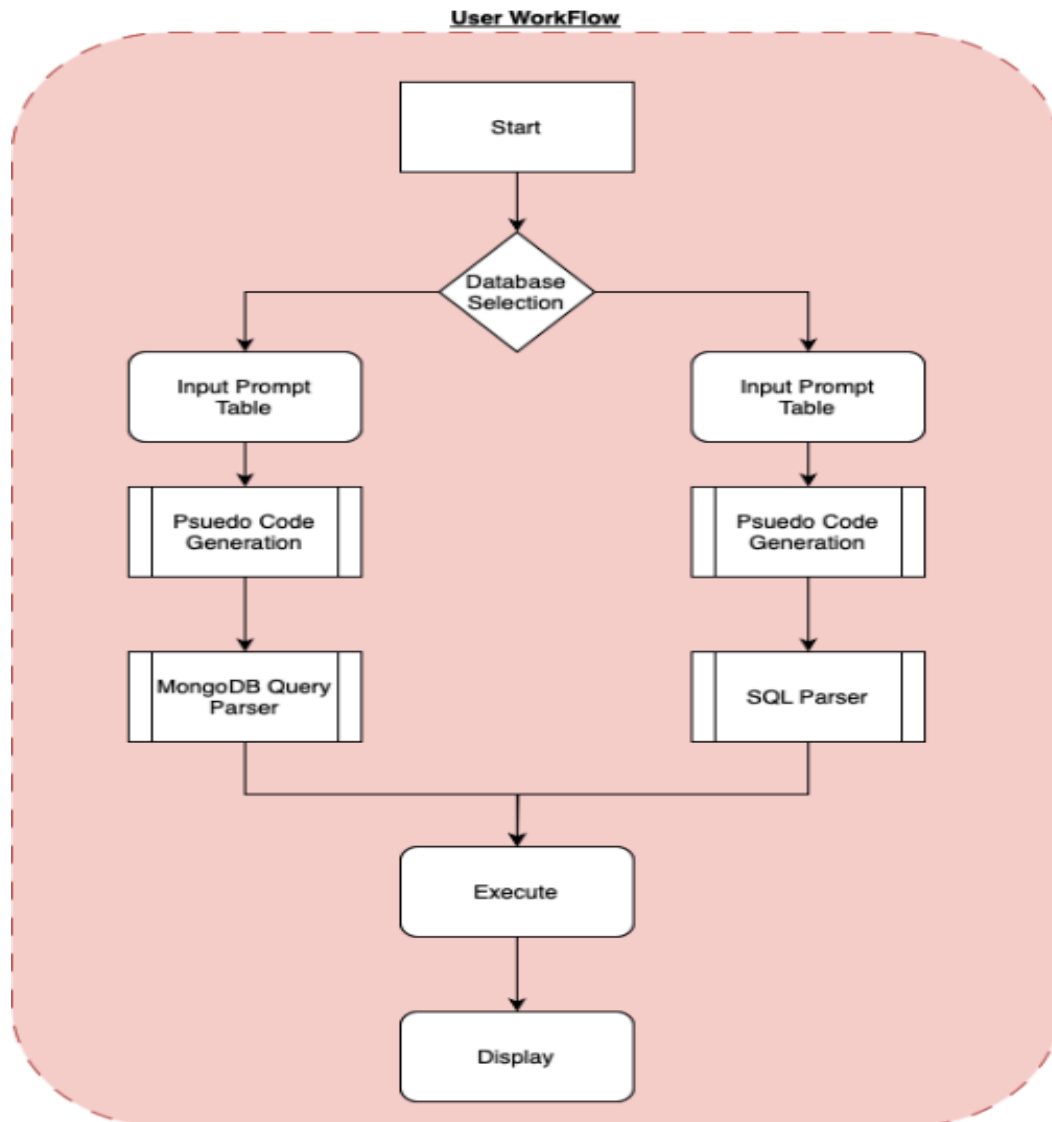
- Test SQL and NoSQL queries across three datasets, ensuring compatibility and accuracy.

3. Architecture Design

How It Works:

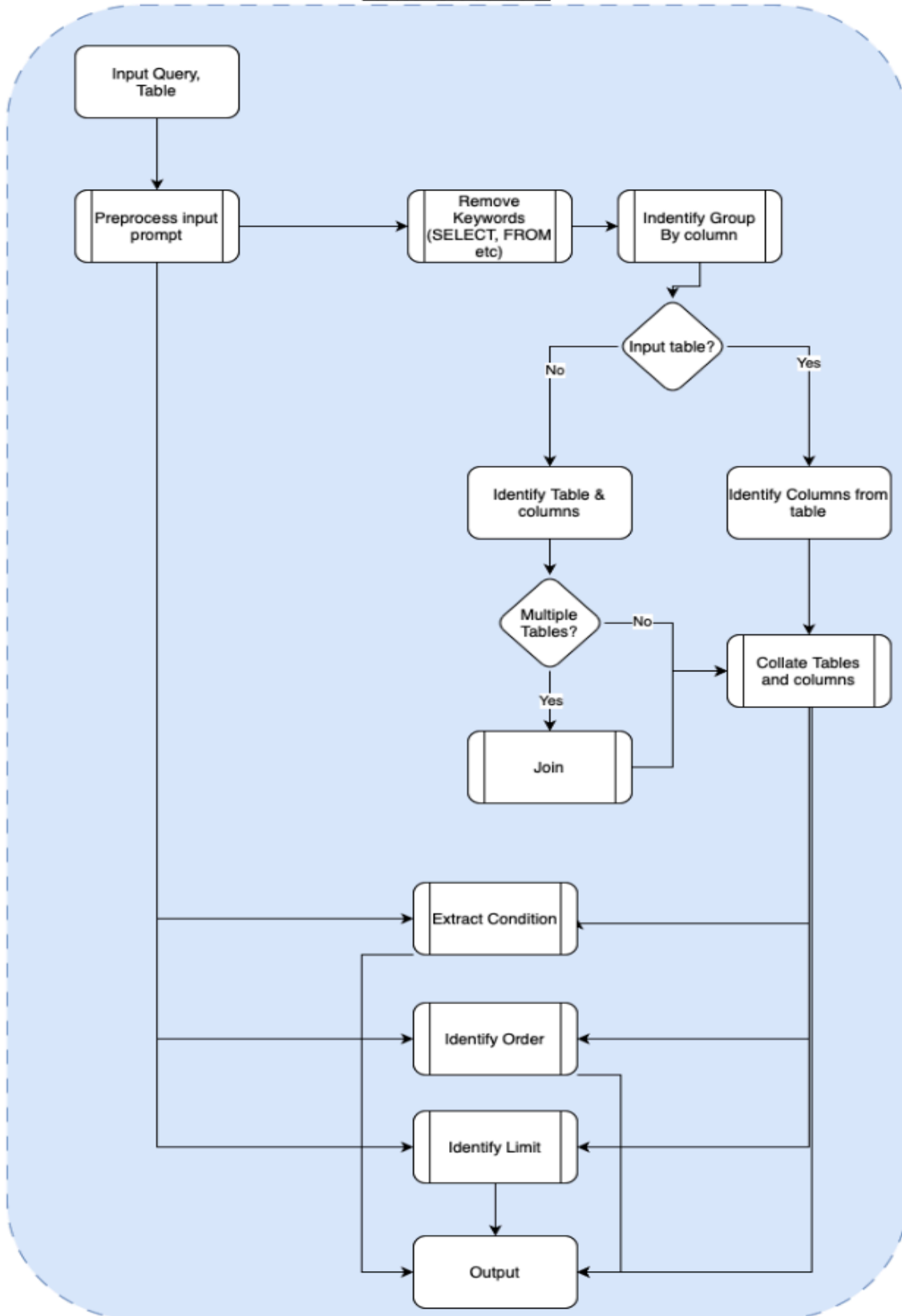
1. The user types a query in natural language.
2. ChatDB tokenizes the input and maps it to predefined patterns or templates.
3. The system converts the input into SQL or MongoDB queries.
4. Suggested sample queries are displayed to guide exploration.
5. The system executes the query and displays results.

This flow ensures flexibility, usability, and compatibility with both structured and flexible databases.



Pseudo Code Generation Flowchart can be found below

Pseudo Code Generator



4. Implementation

Actual Implementation:

- 1. Database Setup:**
 - Installed and configured MySQL and MongoDB.
 - Used a shared healthcare dataset for both databases, including patients, hospitals, and insurance data.
- 2. Dataset Collection and Adaptation:**
 - Adapted the dataset to structured tables in MySQL and collections in MongoDB, ensuring compatibility across systems.
- 3. Query Pattern Mapping:**
 - Created templates for common queries, enabling dynamic generation based on user input.
- 4. Sample Query Generation:**
 - Suggested queries based on user input and query patterns, covering constructs like joins, aggregations, and filters.
- 5. Model Integration:**
 - Integrated tokenization and NER to preprocess queries.
 - Used an RNN model to map natural language to database-specific query syntax.
- 6. User Interface:**
 - Built a Streamlit UI for database exploration, query generation, and query execution.
- 7. Query Execution and Testing:**
 - Successfully executed SQL and MongoDB queries for operations such as filtering, grouping, and joins.



Return average age of patients per gender



Converted into query

```
patients.aggregate([{'$match': {}}, {'$group': {'_id': {'gender': '$gender'}, 'age':
```

Now executing

	_id	age	gender
0	{"gender": "Male"}	51.5003	Male
1	{"gender": "Female"}	51.5787	Female

Query executed successfully. Here are the results:

Download Results



Get patients names and hospital doctors where carelevel is 'Emergency'



Converted into query

```
hospitals.aggregate([{'$lookup': {'from': 'patients', 'localField': 'patientid', 'foreignField': 'id', 'as': 'patients'}}
```

Now executing

	_id	patientid	doctor	carelevel	patients
1	2,003	3	TiffanyMitchell	Emergency	{"patientname":"DaNnYsMith"}
2	2,007	7	KellyOlson	Emergency	{"patientname":"edwArDEDWaRDs"}
3	2,008	8	SuzanneThomas	Emergency	{"patientname":"CHrisTInAMARTinez"}
4	2,013	13	KennethFletcher	Emergency	{"patientname":"connORHAnSEn"}
5	2,025	25	StephanieKramer	Emergency	{"patientname":"PAUlHEndERsOn"}
6	2,028	28	JamesEllis	Emergency	{"patientname":"mr.KenNEthMoORE"}
7	2,029	29	JaredBruceJr.	Emergency	{"patientname":"MaRyhUNter"}
8	2,035	35	LynnYoung	Emergency	{"patientname":"NicOlERodriGUEz"}
9	2,039	39	ChristineJohnson	Emergency	{"patientname":"NicOlELUCeRO"}
10	2,041	41	WilliamReynolds	Emergency	{"patientname":"chRiSTOPHerLEe"}
11	2,046	46	DanielMurphy	Emergency	{"patientname":"lYnnMaRTinez"}

Query executed successfully. Here are the results:

Download Results

Activity Monitor



Retrieve the doctors name that attended carelevel equal to 'Urgent'



Converted into query

```
hospitals.find({'carelevel': {'$eq': 'Urgent'}} , {'carelevel': 1, 'doctor': 1})
```


Now executing

	_id	doctor	carelevel
0	2,001	MatthewSmith	Urgent
1	2,005	KathleenHanna	Urgent
2	2,006	TaylorNewton	Urgent
3	2,011	JohnDuncan	Urgent
4	2,012	DouglasMayo	Urgent
5	2,014	TheresaFreeman	Urgent
6	2,015	RobertaStewart	Urgent
7	2,018	JustinKim	Urgent
8	2,022	GregorySmith	Urgent
9	2,023	VanessaNewton	Urgent

Query executed successfully. Here are the results:

Download Results

Ask about the database... >

 show all patients where age greater than 60

 Converted into query

```
SELECT patients.age
FROM patients
WHERE age > 60;
```

Now executing

	age
0	62
1	76
2	82
3	72
4	75
5	68
6	63
7	63
8	67
9	72

Query executed successfully. Here are the results:

Ask about the database...



Give the count of patients grouped by bloodtype.



Converted into query

```
SELECT COUNT(patients.patientid), patients.bloodtype
FROM patients
GROUP BY patients.bloodtype;
```

Now executing

	COUNT(patients.patientid)	bloodtype
0	6,944	B-
1	6,956	A+
2	6,969	A-
3	6,917	O+
4	6,947	AB+
5	6,945	AB-
6	6,945	B+
7	6,877	O-

Query executed successfully. Here are the results:



List all hospital names and patient names.



Converted into query

```
SELECT hospitals.hospitalname, hospitals.patientid, patients.patientname
FROM hospitals
JOIN patients ON hospitals.patientid=patients.patientid;
```

Now executing

	hospitalname	patientid	patientname
4	White-White	5	adriENNEbEll
5	Nunez-Humphrey	6	EMILYJOHNSOn
6	GroupMiddleton	7	edwArDEDWaRDs
7	PowellRobinsonandValdez,	8	CHrisTInAMARTinez
8	SonsRichand	9	JASmiNeaGullaR
9	Padilla-Walker	10	ChRISTopherBerG
10	Schaefer-Porter	11	mlchElLedaniELs
11	Lyons-Blair	12	aaRonMARTiNeZ
12	PowersMiller,andFlores	13	connORHANsEn
13	Rivera-Gutierrez	14	rObErtbAuer

Query executed successfully. Here are the results:

5. Functionalities

- **Explore Databases:** View schema, attributes, and sample data.
- **Sample Query Suggestions:** Generate dynamic queries based on user input.
- **Query Execution:** Run queries in MySQL or MongoDB, with results displayed in the UI.
- **Data Insertion:** Allow users to insert new data into both databases.

6. Tech Stack

- **Programming Language:** Python
 - **Databases:** MySQL, MongoDB
 - **Libraries:** `pymysql`, `pymongo`, `nltk`, `regex`, `numpy`, `pandas`
 - **UI Framework:** Streamlit
 - **Development Platforms:** GitHub, Visual Studio Code, Sublime, Google Colab
-

7. Learning Outcomes

- Gained experience with natural language processing for query generation.
 - Improved skills in designing schemas and building user interfaces.
 - Learned how to debug real-world projects effectively.
 - Effectively implement Advanced Data structures and algorithms concepts in this project
-

8. Challenges Faced

1. Adapting the healthcare dataset for both SQL and NoSQL systems.
 2. Generating complex join queries for SQL and MongoDB.
 3. Identifying Conditions with correct comparative operator, columns and tables
 4. Handling aggregations, such as **SUM**, **COUNT**, in NoSQL.
 5. Managing regex patterns for query generation, which were prone to errors.
 6. Required domain knowledge to address similarities in patterns.
-

9. Individual Contributions

- **Gleice:**
 - Designed the database structure and schema.
 - Created flowcharts and tested the system.
 - Set up databases.
 - **Vedant:**
 - Developed query generation and UI functionalities.
 - Designed regex patterns for input mapping.
 - Debugged and tested the system.
-

10. Suggestions Implemented after demo

- **Execution of Sample Queries Directly**



sample queries for group by



Sample Queries based on group_by:

```
SELECT benefit, SUM(billingcost) FROM insurance GROUP BY benefit;
```

	benefit	SUM(billingcost)
0	Senior	350,330,647.64
1	Standard	355,661,281.0786
2	Premium	356,534,836.4398
3	Family	354,905,278.1674

Query executed successfully. Here are the results:



sample queries for join



Sample Queries based on Join:

```
SELECT h.*, p.patientname FROM hospitals h JOIN patients p ON h.patientid = p.patientid;
```

	admissionid	patientid	insuranceid	doctor	hospitalname	intakedate	dischargedate	roomnumber	carelevel	testresults	patientname
0	2,001	1	1,001	MatthewSmith	SonsandMiller	01/31/2024	02/02/2024	328	Urgent	Normal	BobbyJac
1	2,002	2	1,002	SamanthaDavies	KimInc	08/20/2019	08/26/2019	265	Emergency	Inconclusive	LesLieTer
2	2,003	3	1,003	TiffanyMitchell	CookPLC	09/22/2022	10/07/2022	205	Emergency	Normal	DaNnYsM
3	2,004	4	1,004	KevinWells	HernandezRogersandVang,	11/18/2020	12/18/2020	450	Elective	Abnormal	andrEww
4	2,005	5	1,005	KathleenHanna	White-White	09/19/2022	10/09/2022	458	Urgent	Abnormal	adriENNE
5	2,006	6	1,006	TaylorNewton	Nunez-Humphrey	12/20/2023	12/24/2023	389	Urgent	Normal	EMILYJOH
6	2,007	7	1,007	KellyOlson	GroupMiddleton	11/03/2020	11/15/2020	389	Emergency	Inconclusive	edwArDE
7	2,008	8	1,008	SuzanneThomas	PowellRobinsonandValdez,	12/28/2021	01/07/2022	277	Emergency	Inconclusive	CHRISTIn
8	2,009	9	1,009	DanielFerguson	SonsRichand	07/01/2020	07/14/2020	316	Elective	Abnormal	JASmiNe
9	2,010	10	1,010	HeatherDay	Padilla-Walker	05/23/2021	06/22/2021	249	Elective	Inconclusive	ChRISTop

Query executed successfully. Here are the results:

sample queries using filter

Sample Queries based on Filter:

```
db.hospitals.find({'roomnumber': {'$lt': 105}})
```

	_id	admissionid	patientid	insuranceid	doctor	hospitalname	intakedate	dischargedate	roomnumber	carelevel	testresults
0	2,077	2,077	77	1,077	JulieRamirez	LinThompsonWells,and	10/11/2023	10/27/2023	102	Urgent	Normal
1	2,131	2,131	131	1,131	TraceySpence	LewisLivingston,Loweand	10/13/2019	10/16/2019	104	Emergency	Normal
2	2,176	2,176	176	1,176	JenniferHall	GroupRodriguez	05/16/2019	05/31/2019	103	Urgent	Inconclusive
3	2,200	2,200	200	1,200	JohnHansen	HumphreyandBrowningFitzgerald,	07/09/2023	07/15/2023	104	Emergency	Abnormal
4	2,237	2,237	237	1,237	MissValerieSaundersMD	Rodriguez-Zimmerman	07/26/2019	08/17/2019	102	Emergency	Abnormal
5	2,269	2,269	269	1,269	TonyaWade	PLCDixon	03/13/2022	03/26/2022	101	Emergency	Inconclusive
6	2,353	2,353	353	1,353	JohnHansen	Koch-Alvarado	05/07/2021	05/31/2021	102	Urgent	Abnormal
7	2,564	2,564	564	1,564	OliviaNichols	Morgan-Young	04/12/2020	05/04/2020	103	Urgent	Normal
8	2,565	2,565	565	1,565	ChristieVargas	GarciaandNguyen,Leonard	11/20/2020	12/15/2020	101	Urgent	Inconclusive
9	2,637	2,637	637	1,637	JulieWest	Vincent-Hill	01/20/2022	02/06/2022	102	Emergency	Inconclusive

Query executed successfully. Here are the results:

sample queries using sort

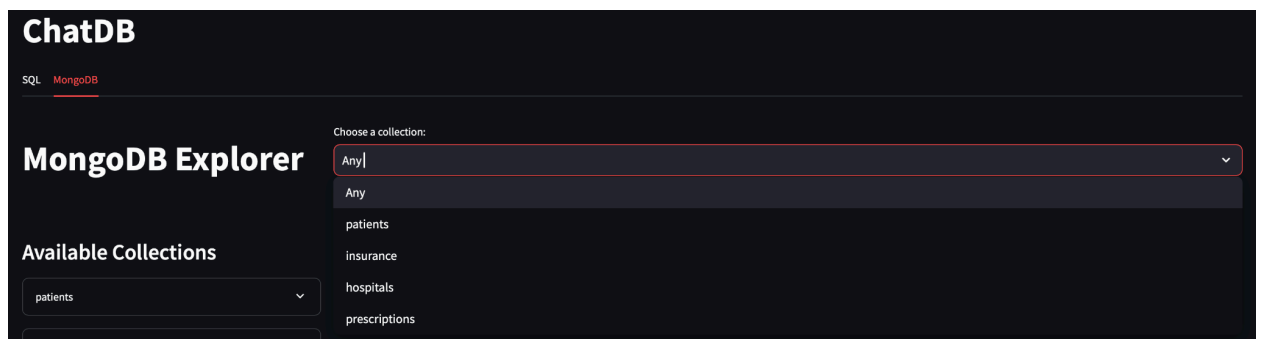
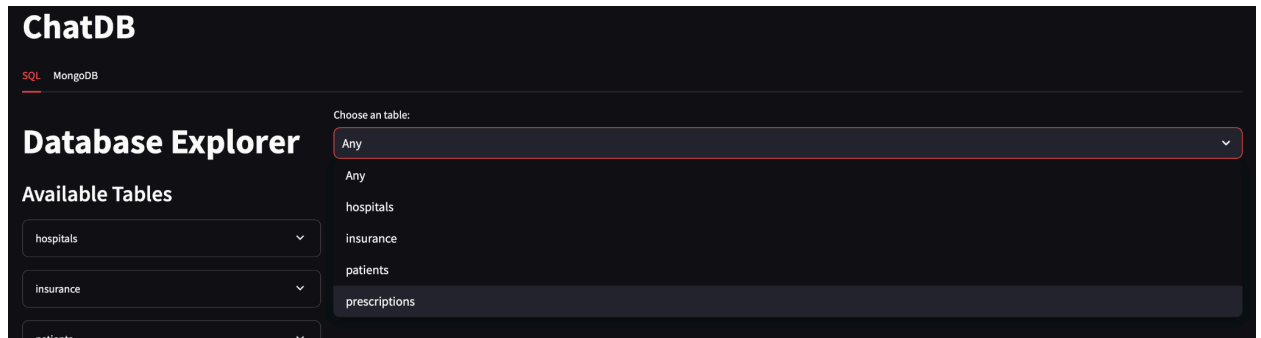
Sample Queries based on sort:

```
db.hospitals.find({}).sort({'intakedate': -1})
```

	_id	admissionid	patientid	insuranceid	doctor	hospitalname	intakedate	dischargedate	roomnumber	carelevel	testresults
0	42,426	42,426	40,426	41,426	BryanRivera	GroupHenderson	12/31/2023	01/07/2024	249	Urgent	Normal
1	52,373	52,373	50,373	51,373	GregoryWolf	Bryant,andPetersonEvans	12/31/2023	01/13/2024	467	Urgent	Normal
2	47,892	47,892	45,892	46,892	BeckyClay	ContrerasandSons	12/31/2023	01/29/2024	352	Emergency	Abnormal
3	5,209	5,209	3,209	4,209	RobertBurns	SmithandNealJones,	12/31/2023	01/09/2024	150	Elective	Inconclusive
4	2,981	2,981	981	1,981	TracyHoward	andRhodesTrujillo,Booth	12/31/2023	01/27/2024	326	Emergency	Inconclusive
5	43,044	43,044	41,044	42,044	CodyGentry	CampbellandLamFuentes,	12/31/2023	01/12/2024	300	Urgent	Normal
6	19,540	19,540	17,540	18,540	GlendaJoseph	Dillon-White	12/31/2023	01/25/2024	281	Urgent	Abnormal
7	5,693	5,693	3,693	4,693	KevinShaw	Brown,andFosterGordon	12/31/2023	01/05/2024	215	Elective	Abnormal
8	51,079	51,079	49,079	50,079	LauraScott	StevensPLC	12/31/2023	01/15/2024	205	Elective	Inconclusive
9	36,716	36,716	34,716	35,716	FrancisBrown	Douglas-Curry	12/31/2023	01/14/2024	342	Emergency	Normal

Query executed successfully. Here are the results:

- Option of user to select table



11. Conclusion

ChatDB shows how natural language can make database queries easier, but building such a system presented several challenges. One major issue was using regex for pattern matching. Regex patterns are powerful for identifying and converting user inputs into database queries, but they can be hard to understand and maintain, especially when the schema/dataset is long or complex. Small mistakes, like typos or syntax errors, can cause bugs or incorrect query results.

Another challenge was ensuring the accuracy of natural language queries, especially when we changed the phrasing or pattern of the inputs. For example, small differences in how a question is asked caused the system to fail in identifying the correct pattern and generating the right query. This is a realistic limitation of natural language. They rely heavily on predefined patterns or templates. Making the system more flexible would require advanced models.

This project also showed the importance of having a multidisciplinary team with different skills. One team member focused on designing a clear and efficient database schema, while another worked on writing regex patterns and debugging the system. This reflects how real-world projects often rely on collaborative teams with different areas of expertise to achieve success.

While ChatDB achieved the basic goals of interpreting and executing queries in SQL and NoSQL databases, it also showed limitations of data science projects in the real-world. The system worked well with the shared healthcare dataset, but scaling it to other datasets or industries would require significant changes. This demonstrates the need for realistic goals in projects like this.

11. Future Scope

1. **Integrate LLM models:** Integrate LLM models making it more robust and dynamic in nature.
2. **Expand Database Support:** Add more databases like PostgreSQL and Cassandra to increase usability for diverse projects.
3. **Enhance NLP:** Improve the model to handle more complex queries, such as joining more than two tables, using subqueries, and advanced filtering.
4. **Enhance UI:** Integrate Data upload into UI and Make the interface more interactive with better error handling and improved data visualization.
5. **Optimize Performance:** Scale the system to handle larger datasets and improve the efficiency of query processing.