ChatDB

DSCI 551 – Final Project Report

Mallika Gummuluri

Introduction:

ChatDB is an interactive web application designed to help users learn how to query data in database systems using a conversational, ChatGPT-like interface. It connects to Neo4j, a graph database, allowing users to:

- Explore database schemas and attributes.
- Generate sample queries using patterns like GROUP BY, HAVING, or ORDER BY.
- Execute natural language queries by converting them into Cypher and displaying results.

The project aims to make database learning more intuitive by bridging natural language understanding with backend database systems.

Planned Implementation:

1. Objective:
   - Create an intuitive web application where users can explore database schemas, generate and execute queries in a graph database.
2. Goals:
   - Support for Neo4j as the backend database.
   - Enable natural language querying and dynamic sample query generation.
   - Develop an interactive web-based UI for seamless user interaction.

Architecture Design

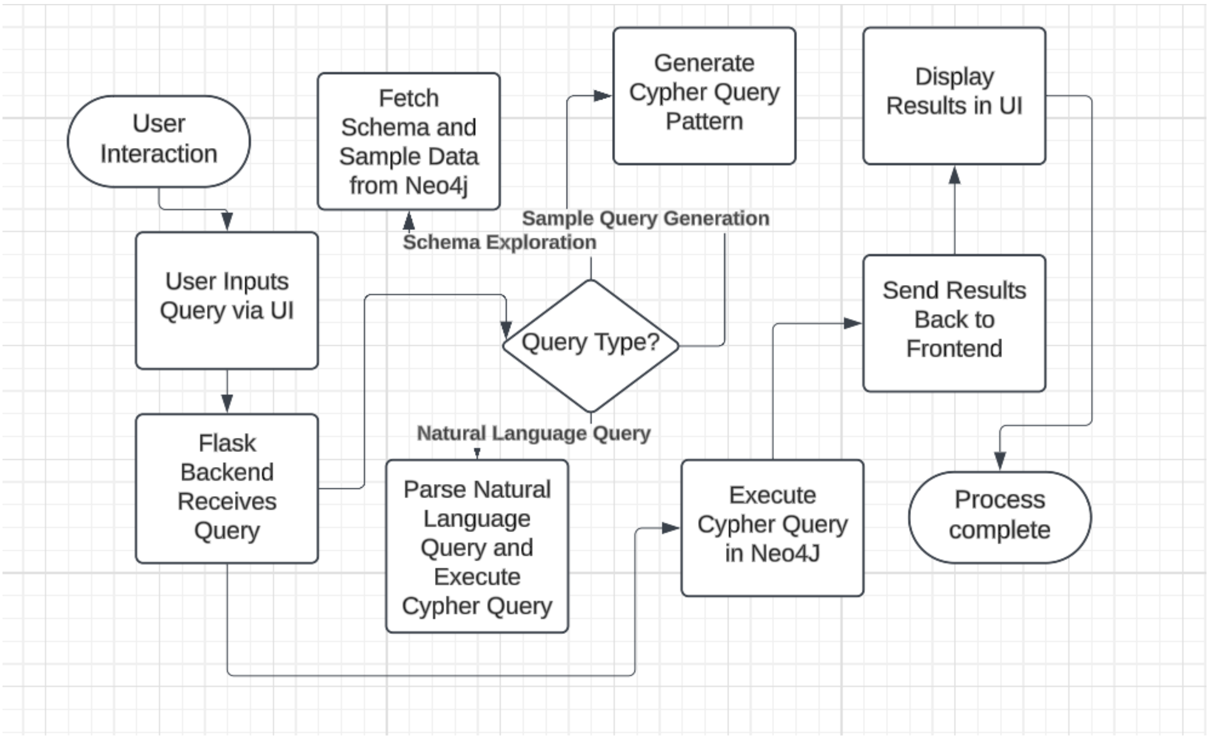
Flow Diagram:

1. Frontend: User inputs (query or database operation) via the web interface.
2. Backend: Flask processes the request, connects to Neo4j, executes the query, and sends results back.
3. Database: Neo4j holds the crime dataset and executes Cypher queries.

Description:

- The frontend sends user input to Flask via REST APIs.

- Flask parses and processes the query:
  - For schema exploration: It fetches attributes from Neo4j.
  - For sample queries: It dynamically generates Cypher patterns.
  - For user-defined queries: It matches patterns or executes custom Cypher queries.
- Results are displayed in the UI for user review.


User Interaction
User Inputs Query via UI
Flask Backend Receives Query
Fetch Schema and Sample Data from Neo4j
Schema Exploration
Sample Query Generation
Generate Cypher Query Pattern
Query Type?
Natural Language Query
Parse Natural Language Query and Execute Cypher Query
Execute Cypher Query in Neo4J
Send Results Back to Frontend
Display Results in UI
Process complete

Implementation

Dataset Preparation:

- Cleaned and formatted the Chicago Crime dataset using a Jupyter Notebook.
- Imported the dataset into Neo4j as nodes (Crime, Beat) and relationships (HAPPENED_IN).

Backend:

- Flask APIs for:
  - Connecting to Neo4j and switching databases.
  - Executing schema exploration, sample queries, and natural language queries.
- Implemented query patterns like GROUP BY, HAVING, and aggregations.

Frontend:

- Built an interactive UI using HTML, CSS, and JavaScript.
- Enabled dynamic query inputs, result displays, and database selection.

Functionalities

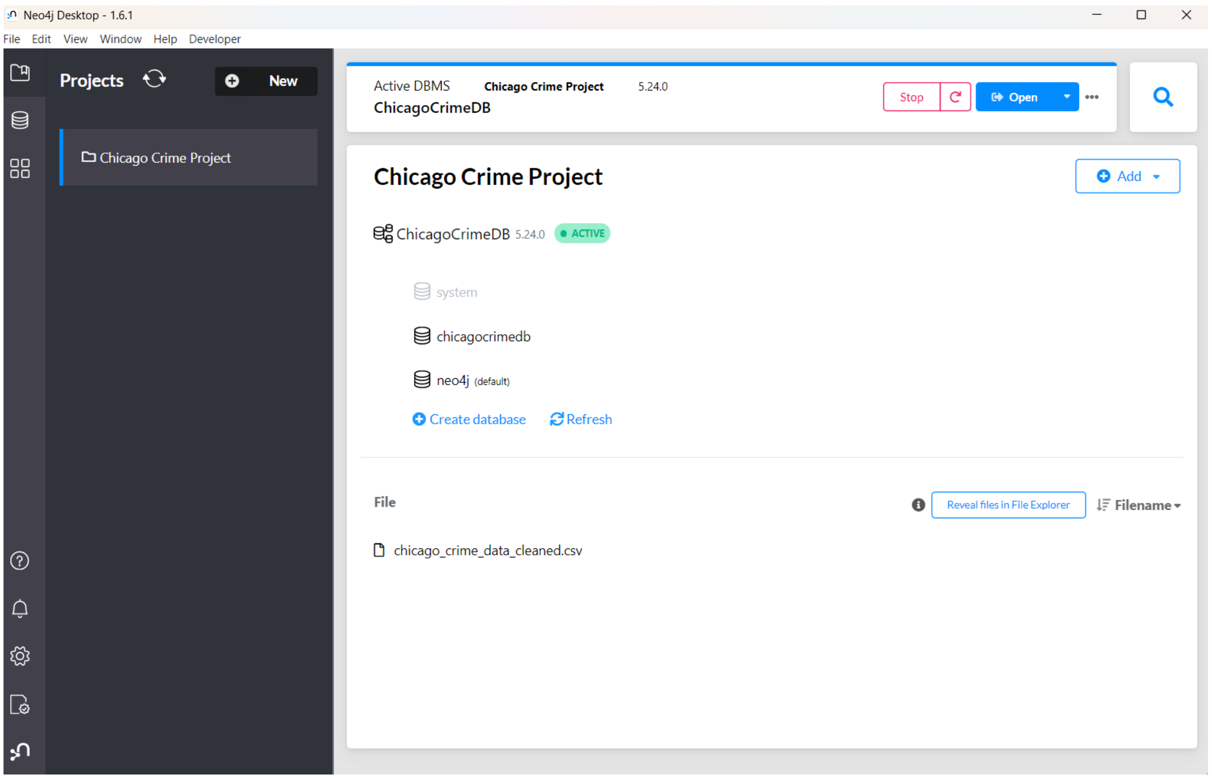
1. Explore databases: Display schemas, tables, attributes, and sample data.
2. Generate sample queries with patterns (e.g., GROUP BY).
3. Execute queries entered as natural language or Cypher.
4. Allow dynamic database switching and customization.

Tech Stack

- Frontend: HTML, CSS, JavaScript
- Backend: Python (Flask)
- Database: Neo4j (with Cypher query language)
- Tools: Jupyter Notebook (for data cleaning), dotenv (for managing credentials)

Implementation Screenshots:

Login to Neo4J and Dataset Import



Chicago Crime Project

ChicagoCrimeDB 5.24.0 ACTIVE

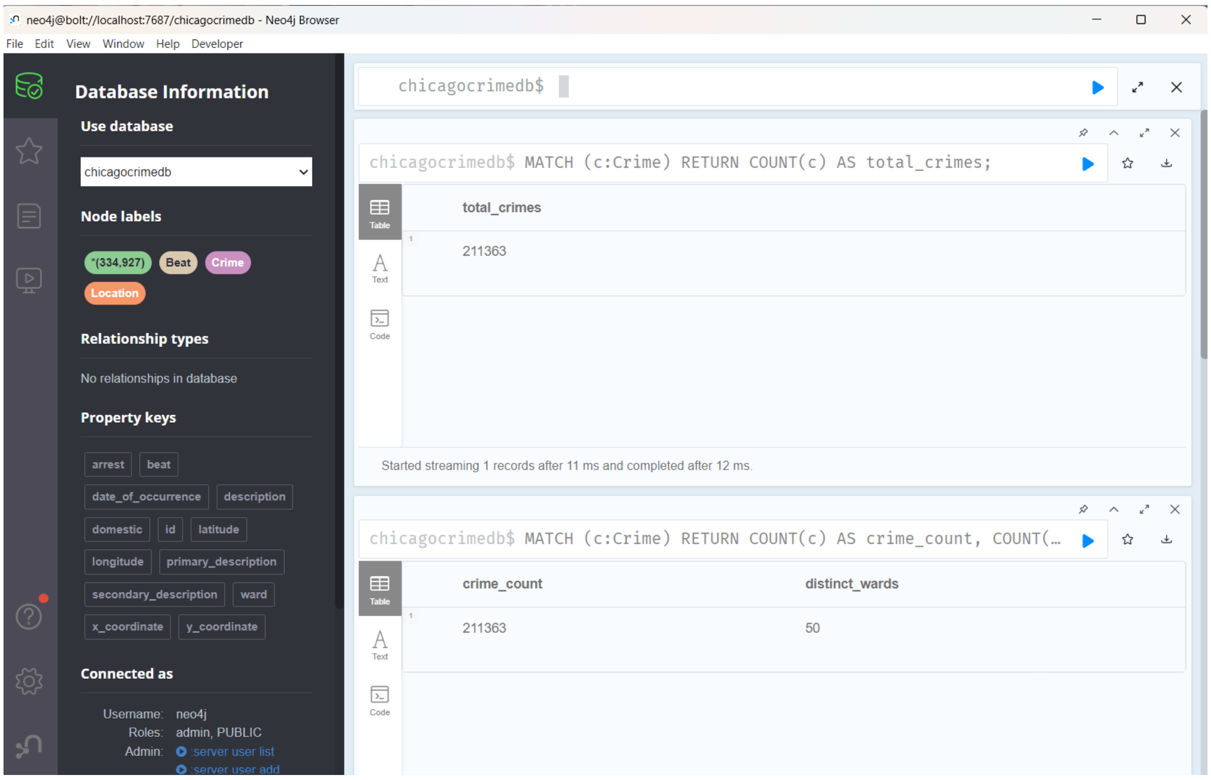
system

chicagocrimedb

neo4j (default)

File

chicago_crime_data_cleaned.csv



```
chicagocrimedb$ MATCH (c:Crime) RETURN COUNT(c) AS total_crimes;
```

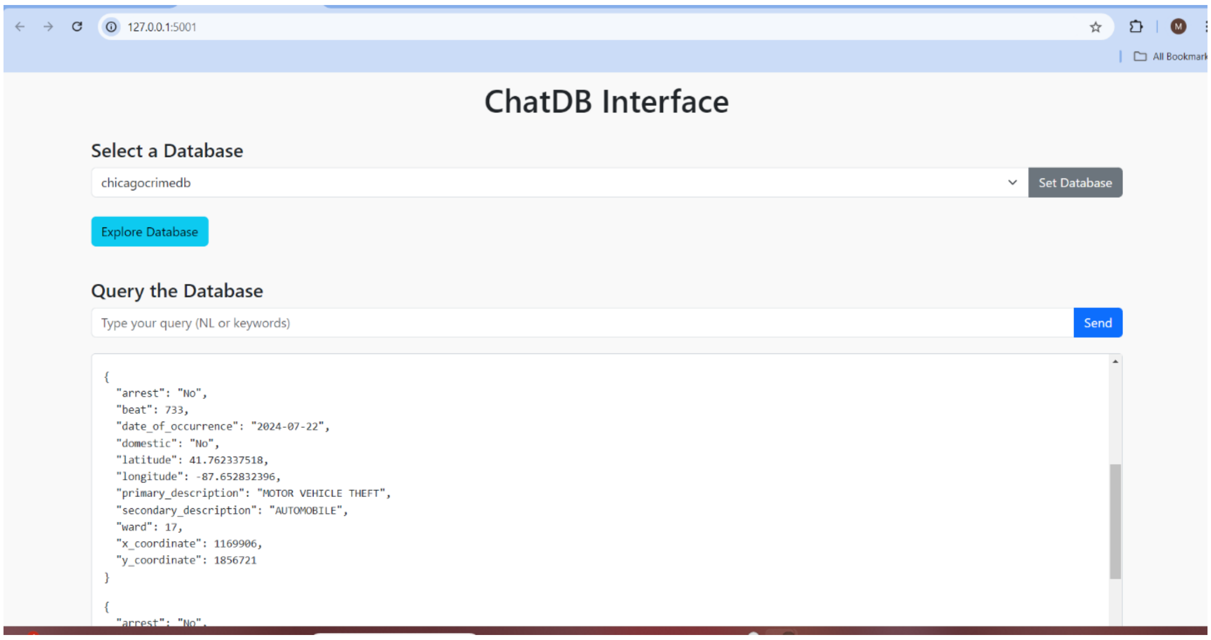
	total_crimes
1	211363

Started streaming 1 records after 11 ms and completed after 12 ms.

```
chicagocrimedb$ MATCH (c:Crime) RETURN COUNT(c) AS crime_count, COUNT(…
```

	crime_count	distinct_wards
1	211363	50

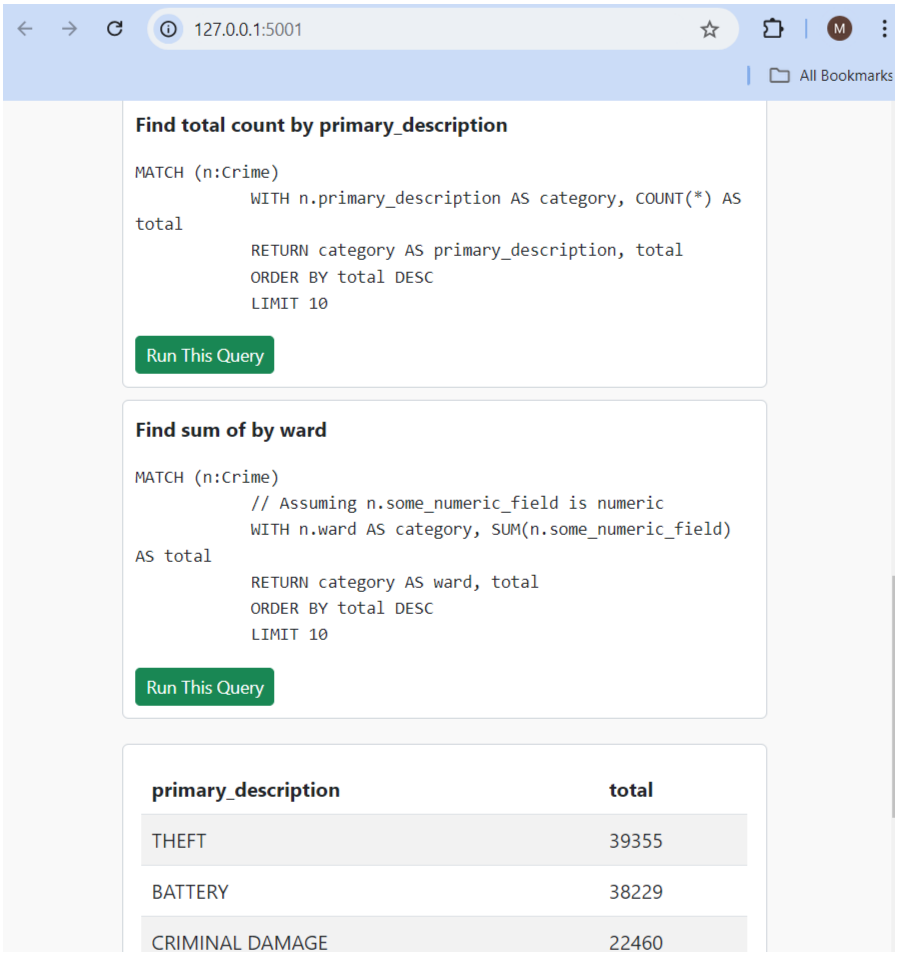
Explore Database Functionality

5. 

```
{
  "arrest": "No",
  "beat": 733,
  "date_of_occurrence": "2024-07-22",
  "domestic": "No",
  "latitude": 41.762337518,
  "longitude": -87.652832396,
  "primary_description": "MOTOR VEHICLE THEFT",
  "secondary_description": "AUTOMOBILE",
  "ward": 17,
  "x_coordinate": 1169906,
  "y_coordinate": 1856721
}
{
  "arrest": "No",
```

Sample Query Generation

1. **Screenshot of the UI generating a GROUP BY query.**



Find total count by primary_description

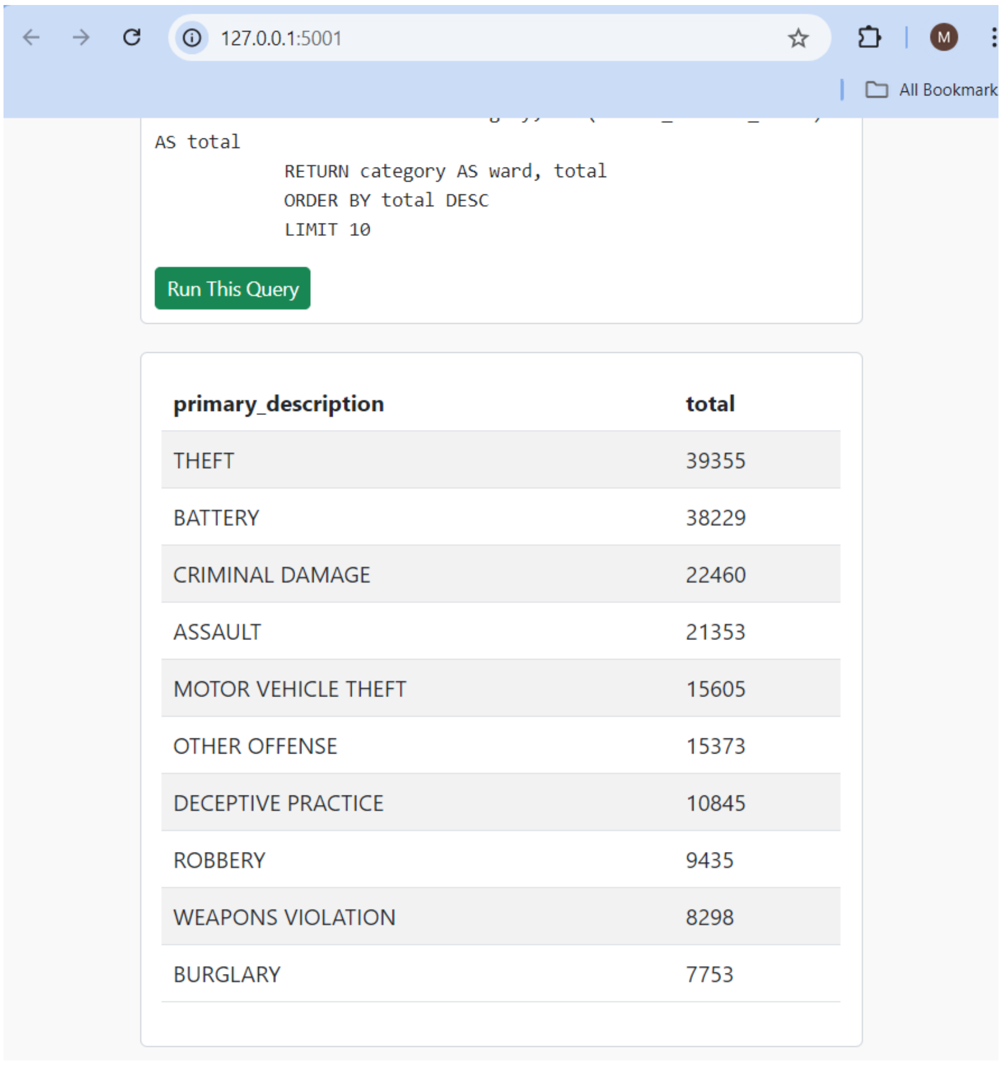
```
MATCH (n:Crime)
            WITH n.primary_description AS category, COUNT(*) AS
total
            RETURN category AS primary_description, total
            ORDER BY total DESC
            LIMIT 10
```

Find sum of by ward

```
MATCH (n:Crime)
            // Assuming n.some_numeric_field is numeric
            WITH n.ward AS category, SUM(n.some_numeric_field)
AS total
            RETURN category AS ward, total
            ORDER BY total DESC
            LIMIT 10
```

primary_description	total
THEFT	39355
BATTERY	38229
CRIMINAL DAMAGE	22460

2. **Corresponding results displayed on the UI.**

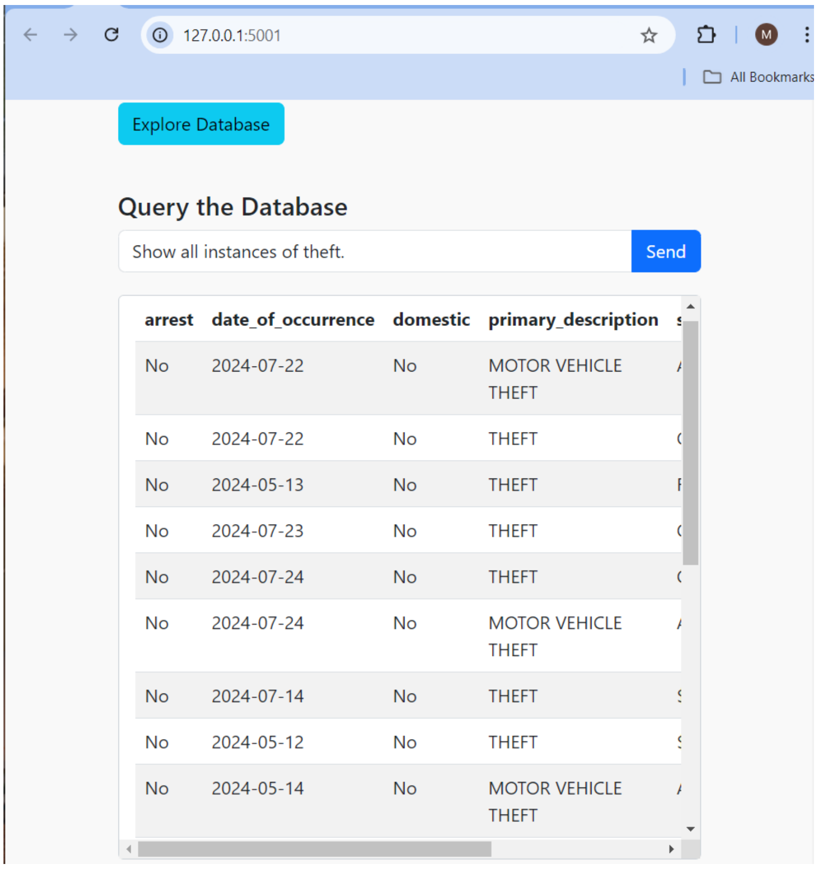


```
AS total
            RETURN category AS ward, total
            ORDER BY total DESC
            LIMIT 10
```

primary_description	total
THEFT	39355
BATTERY	38229
CRIMINAL DAMAGE	22460
ASSAULT	21353
MOTOR VEHICLE THEFT	15605
OTHER OFFENSE	15373
DECEPTIVE PRACTICE	10845
ROBBERY	9435
WEAPONS VIOLATION	8298
BURGLARY	7753

Natural Language Query Execution

1. **Screenshot of the input box with a natural language query (e.g., "Find total crimes by ward").**



Query the Database

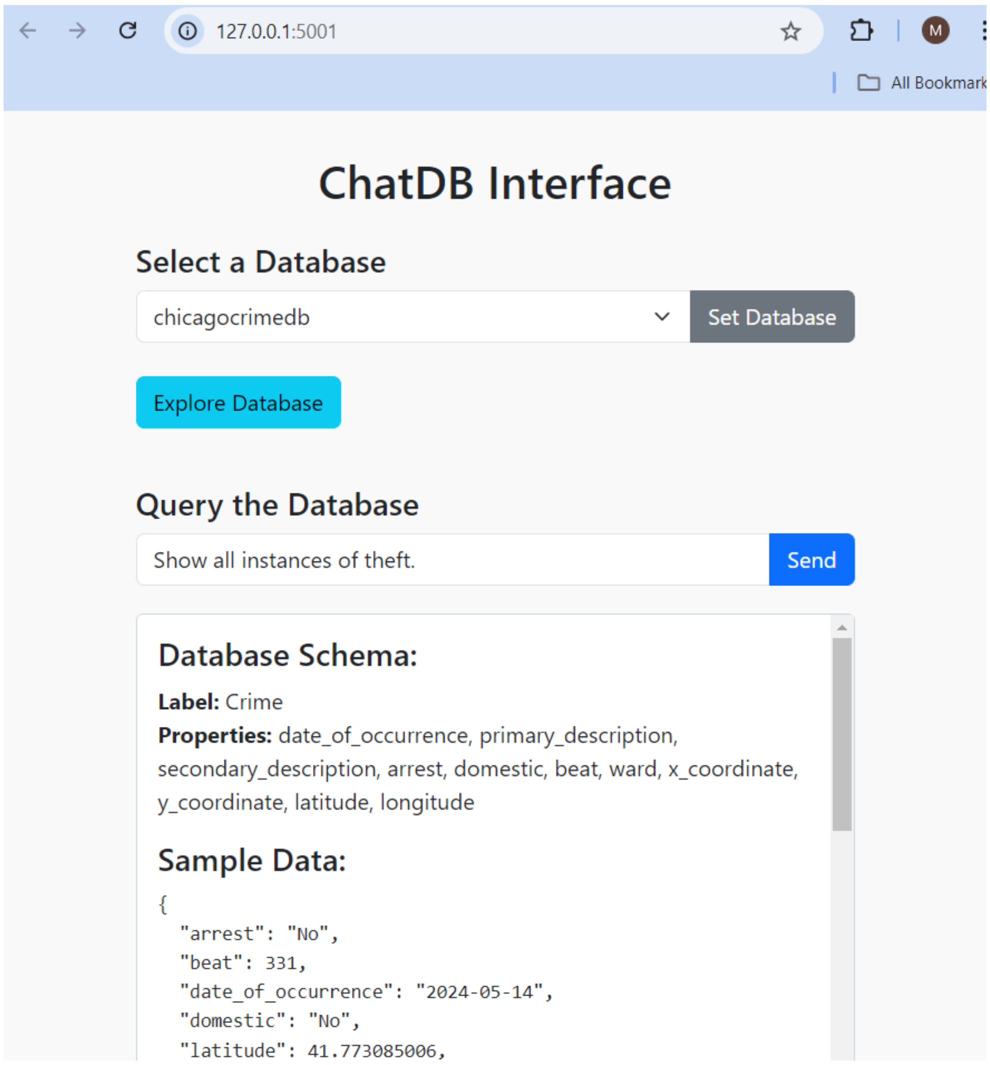
Show all instances of theft.

Send

arrest	date_of_occurrence	domestic	primary_description
No	2024-07-22	No	MOTOR VEHICLE THEFT
No	2024-07-22	No	THEFT
No	2024-05-13	No	THEFT
No	2024-07-23	No	THEFT
No	2024-07-24	No	THEFT
No	2024-07-24	No	MOTOR VEHICLE THEFT
No	2024-07-14	No	THEFT
No	2024-05-12	No	THEFT
No	2024-05-14	No	MOTOR VEHICLE THEFT

? Dynamic Database Switching

1. **Screenshot of the dropdown to switch between databases.**


127.0.0.1:5001

All Bookmark

ChatDB Interface

Select a Database

chicagocrimedb

Set Database

Explore Database

Query the Database

Show all instances of theft.

Send

Database Schema:

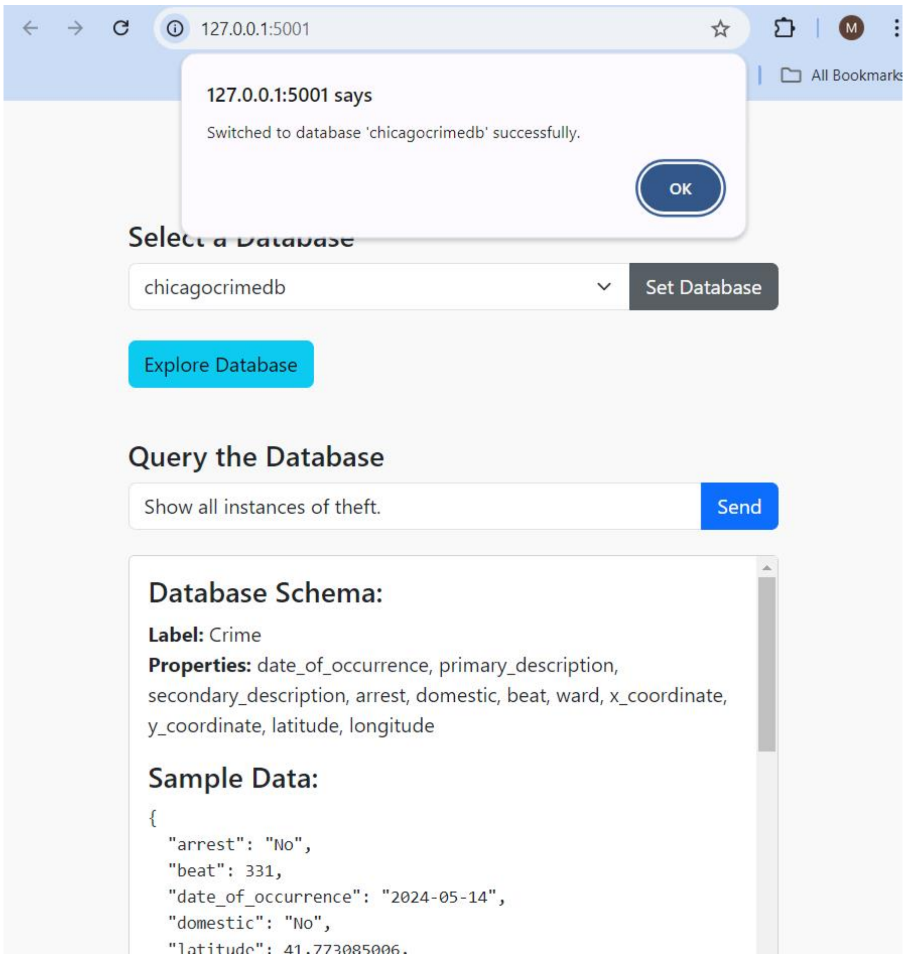
Label: Crime

Properties: date_of_occurrence, primary_description, secondary_description, arrest, domestic, beat, ward, x_coordinate, y_coordinate, latitude, longitude

Sample Data:

```
{
  "arrest": "No",
  "beat": 331,
  "date_of_occurrence": "2024-05-14",
  "domestic": "No",
  "latitude": 41.773085006,
```

2. **Screenshot showing confirmation message after database switch.**



127.0.0.1:5001

127.0.0.1:5001 says

Switched to database 'chicagocrimedb' successfully.

Select a Database

chicagocrimedb

Query the Database

Show all instances of theft.

Database Schema:

Label: Crime
Properties: date_of_occurrence, primary_description, secondary_description, arrest, domestic, beat, ward, x_coordinate, y_coordinate, latitude, longitude

Sample Data:

```
{
  "arrest": "No",
  "beat": 331,
  "date_of_occurrence": "2024-05-14",
  "domestic": "No",
  "latitude": 41.773085006,
```

Learning Outcomes

- Gained experience in integrating Neo4j with Flask for backend development.
- Learned to implement natural language query parsing using Python and NLTK.
- Enhanced understanding of database schemas, relationships, and query patterns.

Challenges Faced

- Neo4j Connection: Overcame errors during the initial database connection and data import.
- Query Pattern Matching: Designed and implemented dynamic templates for generating sample queries.
- Natural Language Parsing: Ensured accurate keyword extraction and mapping to Cypher constructs.

Individual Contribution

- Frontend: Designed and implemented the UI for schema exploration and query execution.

- Backend: Developed APIs for query execution, pattern generation, and schema exploration.
- Database: Prepared the dataset and established relationships in Neo4j.

Conclusion

ChatDB successfully bridges the gap between natural language querying and database interactions, providing an intuitive learning platform. Users can explore schemas, generate queries, and execute them seamlessly in Neo4j.

Future Scope

- Add support for other database systems like MySQL or MongoDB.
- Enable multi-database querying to compare data across systems.
- Integrate more advanced query patterns and analytics.
- Add role-based access control for collaborative learning.

GitHub code link: https://github.com/malllikag/DSCI551_Final/tree/main