Enterprise AI Query Layer for Compliance and Audit

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Computer Engineering

By

Aditya Yadav 09 222118

Anushka Joshi 12 232254

Srushti Potdar 14 232261

Omkar Parab 20 222075

Guide
Dr. Pradnya Sawant
Co-Guide
Mr. Jetso Analin

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)
An Autonomous Institute, Affiliated to University of Mumbai

2025-2026

CERTIFICATE

This is to certify that the project entitled **"Enterprise AI Query Layer for Compliance & Audit"** is a bonafide work of **Aditya Yadav (09), Anushka Joshi (12), Srushti Potdar (14), and Omkar Parab (20)** submitted to the University of Mumbai in partial fulfillment of the requirements for the award of the degree of **B.E. in Computer Engineering**.

(Dr. Pradnya Sawant)
Guide

(Mr. Jetso Analin)
Co-Guide

(Dr. Dakshata Panchal)
Head Of Department

(Dr. Deepak Jayswal)
Principal

Project Approval Report for B.E.

This project report entitled *"Enterprise AI Query Layer for Compliance & Audit"* by *Aditya Yadav (09), Anushka Joshi (12), Srushti Potdar (14) and Omkar Parab (20)* is approved for the degree of *B.E. in Computer Engineering*.

Examiners

1. Dr. Angha Raich

2. Snehal Kulkarni
04.05.26

Date: 04/05/2026

Place: Mumbai

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aditya Yadav (09)

Anushka Joshi (12)

Srushti Potdar (14)

Omkar Parab (20)

Date: 04/05/2026

Abstract

In highly-regulated sectors such as Fintech and large enterprises, compli- ance and audit teams are all too familiar with the frustration of being reliant on database engineers to make the complex SQL queries just to pull standard reports. To In order to overcome this bottleneck our project introduces the Enterprise AI Query Layer for Compliance Audit - A secure, AI driven interface that sits atop structured databases. This system effectively acts as an intelligent assistant, allowing non tech savvy auditors to ask questions and pull data using everyday English with translation. And requiring no specialized SQL knowledge at all, it greatly extends the pace of their entire reporting workflow to boot.

Where this platform truly stands out is its timeless commitment to enterprise security and governance. Designed specifically for regulated environments with fine-grained Row-Level Access Control (RLAC) that allows users to only view data they are authorized to access. In addition, an immutable log of every query run and information captured is held by the system as a stringent audit trail to ensure complete transparency. Additionally, we have implemented a feedback loop where the AI can learn when users make corrections and such improvements are rife with machine learning.

In short, this platform takes data retrieval from a slow chore that depends on either an engineer or a team of engineers and turns it to instant self-service enabling operational efficiency. It cuts audit lead times, ensures a higher tier of data transparency and provides modern enterprises with a highly scalable RegTech solution.

Keywords: *Enterprise AI, Natural Language Interface to Databases (NLIDB), Compliance, Financial Audit, Row-Level Access Control (RLAC), Text-to-SQL, RegTech, Database Security, Query Auditing*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Description

Managing today's regulatory landscape is tough, and enterprise audit teams often face major hurdles when trying to access and analyze their own structured data. Right now, most compliance professionals rely heavily on engineers to write SQL queries just to pull information from complex databases. This constant back-and-forth delays the generation of crucial audit reports and drags down overall efficiency in fast-moving, highly regulated sectors like finance and banking. Fixing this bottleneck requires a smart solution that lets compliance teams work independently, without needing deep technical skills.

Our proposed project introduces an Enterprise AI Query Layer for Compliance and Audit, designed specifically to let auditors and compliance officers interact with their databases using everyday language. Instead of wrestling with SQL, users can simply ask questions in plain English. The system then translates these questions into highly optimized queries, fetches the results, and presents them in a clean, structured format. This approach saves a massive amount of time, cuts out the technical middleman, and brings a new level of transparency to the auditing process.

Beyond just translating text, the system is built on a foundation of trust and security. It feature row-level accession control, strict audit trails for every executed query, and a feedback loop that continuously tunes the AI to improve its accuracy. By baking in these features, our solution ensures strict compliance with regulatory standards and actively prevents unauthorized data access.

1.2 Background Study Terminologies

In order to understand fully the architecture and operations behind the AI Query Layer for Compliance and Audit on the Enterprise system, it would be beneficial to look into the meaning of some terminologies that have been employed in the development process and which include technical and industry terminologies.

- **Row-level Access Control:** It is a very strict security measure that allows you to control access to particular database rows according to user roles, ensuring that sensitive information can be accessed only by those who are allowed to do so.
- **Natural Language Query Processing:** This is an AI-powered technology that will enable you to transform your ordinary English question into a precise database query, letting you work with your data seamlessly.
- **Audit Trails:** Complete, unalterable logs capturing all queries, as well as the output returned, enabling complete tracking, traceability, and compliance validation.
- **Feedback Loop:** An ongoing cycle through which user feedback is used to fine-tune AI-assisted processing and improve the quality of query translation.
- **Structured Query Processing:** Actually creating a SQL query based on your natural language input that would let you interact with structured databases.
- **RegTech:** A portmanteau of regulation and technology, which refers to advanced solutions aimed at simplifying compliance, audit, and reporting in highly regulated industries.

1.3 Fundamental Study Points

The first objective of this research effort is to empower compliance and auditing professionals without technical expertise to effectively conduct queries against structured databases without any AI-based natural language. This chapter identifies the essential focus areas and key underlying principles that enable the implementation of this system.

- **Overcoming Technical Gaps:** Allowing the average auditor and compliance officer to easily analyze information from the database without having to know even the most basic of SQL syntax, thus decreasing their reliance on technical support immensely.
- **AI-based Natural Language Parsing:** Using artificial intelligence to process human speech and extract the actual meaning behind the words to convert them into an effective query to the database.
- **SQL Queries Generation:** Creating structured queries from the user inputs which have been parsed through NLP to pull the correct data required by the user.
- **Row-Level Access Control:** Providing row-level access controls and security checks to all data requests to ensure that all information is both secure and compliant with regulations.
- **Tracking Everything:** Logging and tracking all actions taken by users in the system including their queries, system replies, and actions taken.
- **Learning Based on Feedback:** Creating an efficient system that continually learns based on user inputs to increase accuracy over time.
- **Efficiency Through Automation:** Automating the entire process from query analysis through to reporting to save time and effort on the part of auditors.

1.4 Challenges

Designing an enterprise-level AI query layer to ensure compliance and audibility poses multiple sophisticated challenges that go well beyond standard text-to-SQL systems.

- **Accuracy of Schema Translation**: Delivering highly accurate translations for even the most complex database schemas without requiring frequent manual intervention or continuous retraining of the AI model.
- **Alignment of Intent and Schema**: Closing the gap between the user's intention in natural language and the reality of the database schema, particularly difficult when translating highly specialized corporate terminology.
- **Security Governance**: Embedding advanced security governance features such as row and column-level permissions directly within an AI-based architecture to ensure that no data leaks can ever occur.
- **Efficient Performance Monitoring**: Embedding a full-fledged auditing mechanism that logs all queries, generated SQL statements, and results without sacrificing performance.
- **Scalability and Explainability**: Guaranteeing robust performance under heavy load conditions and the ability to dynamically update database structures without compromising the explainability of the AI system.
- **Human-AI Feedback Loop**: Finding the right balance between automation and human input to refine the system's capabilities through feedback without introducing potential bias or regulatory risks.

1.5 Motivation

The drive to build this Enterprise AI Query Layer stems directly from the critical operational bottlenecks and high-risk challenges currently plaguing daily compliance and audit workflows.

- **Getting Rid of the Engineering Bottleneck**: The first main motivation here is the need to solve the extremely frustrating problem of delay arising due to waiting for database engineers to provide even basic information.
- **Giving Data Access a Facelift**: With the help of cutting-edge AI technology, this proposed solution aims at giving auditors a direct and understandable way to interact with the structured data, eliminating the need to master SQL language.
- **Implementing Built-in Compliance**: In other words, the entire purpose behind this project is the necessity to create a system that is compliant out-of-the-box, including native row-level access controls and immutable audit logging for every action made.
- **Minimizing Risks for the Organization**: In the end, the goal here is to turn compliance into a proactive and fast process, which would help reduce risks for the organization.
- **Increasing Efficiency of Report Generation**: In fact, by introducing an immediate approach to data access and querying, it will be possible to greatly shorten report generation cycles and make audit processes more efficient.
- **Filling Gaps in the Current State-of-the-Art**: As of now, Text-to-SQL research does not pay much attention to non-functional requirements, which become obligatory in the context of enterprise auditing and security models.

1.6 Problem Statement and Proposed Solution

Problem Statement: Enterprise compliance and audit processes are frequently bottlenecked by a heavy reliance on technical staff just to execute routine database queries. The manual creation of SQL queries introduces significant delays, exacerbates daily operational bottlenecks, and severely restricts timely access to critical compliance data. Furthermore, the lack of built-in access controls and automated audit trails elevates the risk of data mismanagement and strict regulatory non-compliance.

Proposed Solution: To finally resolve these critical operational delays, this project proposes an Enterprise AI Query Layer designed to truly empower non-technical compliance officers. By leveraging a fine-tuned Natural Language Processing (NLP) model, the system allows auditors to interrogate complex relational databases using just plain English. The AI then automatically translates these inquiries into highly optimized, executable SQL queries. Crucially, the solution is designed with enterprise governance right at its core, integrating dynamic role-based access controls and immutable audit logging directly into the query execution pipeline. This entirely transforms data retrieval from a slow, engineer-dependent chore into a secure, self-service, and fully auditable process.

Chapter 2

Literature Survey

2.1 Survey of Existing Systems

Table 2.1: Summary of Related Work

Ref.	Methodology	Advantages	Limitations
[1]	- NL interface with OLAP hypercube - NLQ to SQL translation	- Fast query response - Handles complex SQL - Easy for novice users	- Complex hypercube design - Limited dimensionality - Needs data pre-cleaning
[2]	- Modified models for better output - Auto chart generation	- Simpler SQL outputs - Supports database scaling	- Struggles with advanced queries - Manual schema cleanup needed
[3]	- Reinforcement Learning for NLQ - Explainable SQL generation	- Clear query explanations - High accuracy	- Computationally heavy - Needs detailed schema preprocessing
[4]	- Transformer-based financial Text-to-SQL - Context and semantic integration	- High translation accuracy - Handles complex financial data	- Needs large financial dataset - Complex schemas require engineering
[5]	- Uses SQL logs for schema understanding	- Reduces ambiguity - Practical for enterprises with logs	- Cold-start problem - Domain-specific
[6]	- Graph-based schema representation - Links NL words to tables/-columns	- Cross-domain generalization - Improved SQL accuracy	- Computationally heavy - Needs detailed schema
[7]	- Unified text-to-text T5 model - Converts NL + schema to SQL	- Flexible across tasks - Open-source support	- May generate invalid SQL - Needs fine-tuning
[8]	- Evaluates LLMs on Spider dataset - Uses prompt engineering	- Strong benchmark results	- Focused on evaluation - Costly for large LLMs

- In [1], the researchers examine the use of a Natural Language Interface to Relational Databases (NLIDB). The research takes advantage of OLAP hypercube architecture to easily transform a natural language request to SQL. With the help of NLTK from Python for tokenization, lemmatization, and syntactic-semantic analysis, it manages to perform complex joins by means of a four-dimensional OLAP cube. On Oracle 19c, this solution proved to be extremely fast and accurate, although further investigation should be done to enable large data sets.

- In order to facilitate data accessibility to the end-user, the team behind [2] introduces *AskYourDB* – an intuitive way to query and visualize databases. Using SADGA+GAP framework to jointly encode both the input question and database schema enables precise SQL generation. Moreover, the preprocessing step helps with synonyms and ambiguity. Additionally, Deepeye ensures that generated charts are available instantly. The research is rather reliable, though scalability and SQL optimization can be improved in the future.

- Trying to cope with the issue of black-box in AI, researchers behind [3] come up with *xDBTagger* – a highly-transparent hybrid pipeline designed for text-to-sql translation. With the use of keyword mappings and schema graphs, it ensures full transparency of translation in textual and visual form. Most importantly, despite the increase in transparency, its accuracy remains at the same level while operating up to 10,000 faster than standard pipelines.

- Trying to look further than simple application of technologies, researchers behind [4] propose a machine-learning method for mapping of Course Outcomes and Program Outcomes. Testing several classifiers, including logistic regression, linear support vector machine, random forest, and others, along with different feature extraction methods such as TF-IDF, BERT, etc., they found out that RoBERTa performed best of all. The paper shows promising possibilities for applying semi-supervised learning to domain-independent datasets.

- Dealing with problems of keyword mapping and join path inference, authors of [5] suggest an innovative system called *TEMPLAR*. Taking advantage of historical SQL queries, TEMPLAR can significantly improve the quality of keyword mappings and therefore decrease ambiguity when translating highly complicated natural language inputs. It actively learns from past queries to map users' questions precisely to database schemas.

- The research described in [6] describes *RAT-SQL*, an extremely efficient and relation-aware framework built to perform Text-to-SQL parsing. RAT-SQL utilizes self-attention to deeply analyze and comprehend the connections between the user query's words and particular database schema elements. In experiments on the demanding Spider benchmark, RAT-SQL outperformed existing approaches, and when paired with BERT, its performance was significantly improved.

- An important breakthrough in the field of natural language processing is presented in [7] through *T5* (Text-to-Text Transfer Transformer). The proposed approach considers all possible NLP tasks as problems of text-to-text translation, allowing for unparalleled flexibility and adaptability. Due to pretraining with huge datasets, such as C4, T5 has set new benchmarks in many different areas, demonstrating once again the power of transfer learning for various AI applications.

- Lastly, the paper published in [8] assesses the current performance of large language models (LLMs) in Text-to-SQL tasks. The researchers have created a new model, called *DAIL-SQL*, that heavily utilizes prompt engineering techniques and achieves an excellent 86.6% accuracy on the Spider benchmark dataset. With their research, they provide insights into the differences between open-source models and fine-tuned LLMs in the field of Text-to-SQL processing.

2.2 Limitations of Existing Systems

While there are numerous successful examples of NLIDB translation accuracy and database schema handling (e.g., `RAT-SQL` and `DAIL-SQL`), such solutions are not sufficient for industrial applications. Indeed, one of the crucial gaps in the literature is that no effort is made to meet the mandatory non-functional requirements necessary for highly-regulated domains, such as financial technologies (fintech) and auditing services: high security and robust enterprise logging.

Specifically, although there are attempts at verifying the output of queries on financial data [3], the need for fine-grained security policies that will allow for strict auditor permissions remains unmet. Moreover, while there are efforts towards developing explainable models [4], the problem of auditing such queries from a security perspective is not adequately addressed. Indeed, in a compliant enterprise environment, the solution must be able to store the input query, generated `SQL`, and final results in a secure manner, which is currently not implemented in any system. Furthermore, an auditing tool should be able to tune itself according to auditor input, which remains unexplored in the literature.

2.3 Scope of the Proposed System

- **Natural Language to SQL Translation (NL-to-SQL):** The primary job of the system would be to successfully translate the user's natural language queries into executable SQL commands, taking into account advanced database operations, such as filters, sorts, aggregations, etc.
- **Database Connectivity:** The system is built in a way that would provide a safe connection to a chosen relational enterprise database (MySQL/PostgreSQL), which can return necessary compliance data by itself without any additional configuration on the back-end side.
- **Automated Database Structure Scanning:** After the connection is made, the system will be able to automatically scan a database schema, recognize

table relations, and map users' plain English terms to the required database components.

- **Role-Based Access Control (RBAC):** The system should have a basic RBAC component, which would allow restricting access to some data according to pre-defined roles for auditors, compliance officers, and admins.
- **Execution Constraints:** To ensure data integrity at all times, the system would perform queries in a read-only manner. Modifying actions like INSERT, UPDATE, DELETE would be impossible to execute against the target database.
- **Audit Trail Creation:** In order to ensure that there is full transparency and accountability within the system, each action performed by the users will be recorded, including the natural language query, the created SQL statement, the identification information of the user, and the precise timestamp.
- **User Feedback and Refining the Algorithm:** A feedback mechanism will also be included in the platform where the users can evaluate the accuracy of the translation process. Using the gathered feedback data, the AI model will continue to learn and improve its accuracy in translating the natural language queries.
- **User Interface:** A user-friendly web-based application interface will be designed that non-technical users can use to enter their queries and see the results from the database, as well as their audit trails.
- **Data Visualization:** One of the key features of the project will be automatic generation of basic graphics (e.g., pie chart, bar chart) for displaying raw results of compliance queries.

Chapter 3

Proposed System

3.1 Detailed Explanation of Proposed System

3.1.1 General Block Diagram

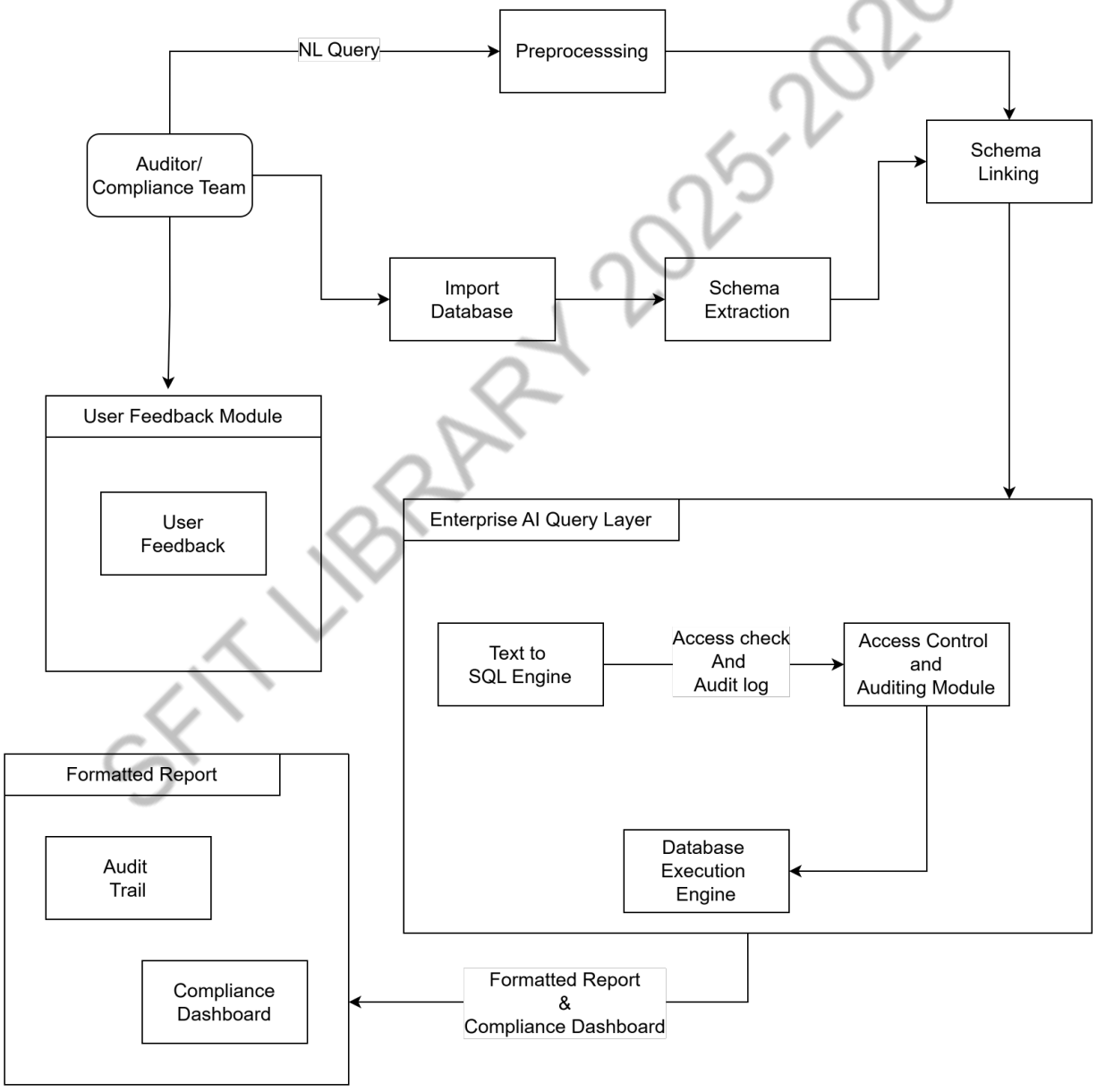

Figure 3.1: General Block Diagram showing the overall system structure and major components.

3.1.2 Principle of Operation

The essence of the work of the Enterprise AI Query Layer is based on a sophisticated Text-to-SQL pipeline using a fine-tuned T5 Transformer model, which serves as an interface translating human-friendly conversational language to valid SQL code.

Text-to-SQL Query Generation Algorithm

The detailed flow of actions required to process the text question into an executable SQL command goes like this:

- **Text Reception and Cleaning:** First, the system receives the user's natural language question along with the database schema. Then, it performs text cleaning procedures, including lowercasing and removal of irrelevant stopwords.
- **Text Entity Extraction:** Next, the system performs entity extraction, identifying potential column names, table names, data constraints, etc., contained within the text.
- **Text Encoding:** The processed query text and the database structure map are merged together and encoded into a single representation, suitable for further processing with T5.
- **Text-to-SQL Prediction:** The fine-tuned T5 model uses the previous step's output as input to predict a proper token sequence.
- **Post-Processing of the Output:** The output SQL code is formatted correctly, fixing some minor issues of space or syntax.
- **Validation and Execution of the SQL Query:** Prior to execution, the system validates the SQL statement for correctness according to the database schema and the access permissions of the user. In case of any discrepancies, the action is prohibited, and an exception thrown to the user.

3.1.3 Module Wise Explanation

- **Auditor/Compliance Team**: These are the main, nontechnical users of the system. They work with the help of importing databases and asking simple questions.
- **NL Query**: Plain-language questions that are put in the search box. For instance, "Show all failed compliance transactions of the last quarter."
- **Import a Database**: The first step in which a database is connected to the platform.
- **Schema Extraction**: Once the database is imported, this phase automates the process of recognizing and mapping the structure of the database, identifying tables, columns, data type, etc.
- **Preprocessing**: This is the very first phase of working with any user query. Preprocessing refers to standardizing a user's text and making it understandable for AI.
- **Schema Linking**: This is a significant phase at which AI matches the terminology of the query to the corresponding tables and columns identified during extraction.
- **Enterprise AI Query Layer**: A core of the whole architecture that makes everything possible.
  - **Text-to-SQL Engine**: An AI-based model that translates queries into formal SQL commands. Also, in this phase, it performs basic authentication of the user's query and registers it.
  - **Access Control & Auditing Module**: This is a mandatory security phase. At this stage, it intercepts the request and verifies whether the user can really have access to such information.
  - **Database Execution Engine**: Once the security phase is passed successfully, this one performs a request to the database and fetches the required

data.

- **Formatted Report**: Results provided to the user in a neat and tidy format.
  - **Audit Trail**: Permanent and immutable record of each and every action. It contains information about who asked what, in what format, and with what results. This phase is necessary for further compliance analysis.
  - **Compliance Dashboard**: A user-friendly interface where the results are summarized usually via visual representation.
- **User Feedback Module**: In this phase, the user can evaluate the quality of the provided results and give additional information about the relevance of the AI's output.

3.2 System Analysis

3.2.1 Functional Requirements

1. **Authentication and Authorization of Users:** It would be mandatory for the system to demand authenticated login credentials from the user. Subsequently, it will validate the user and determine their respective roles (for instance, Admin, Auditor, or Compliance Officer).
2. **Processing Queries in Text:** The system will feature a text-based user interface which accepts natural language query. The system would then analyze the query to derive user intent and detect important entities, actions, and constraints like "show", "transaction", and "quarter".
3. **Generating SQL Query:** Based on the parsed text, an accurate SQL query will be generated by the system. The generation process requires intelligent mapping between common words used in a question and their database table/column equivalents.
4. **Execution of SQL Query:** For executing the generated SQL, the system requires establishing a connection with the designated database of the enterprise.

5. **Fetching Result and Its Display:** After executing the query, the system retrieves data from the database and provides it to the user as a neat output, which can be a well-organized data table.
6. **Logging of Auditing Trail:** Every user activity related to data access would be recorded silently without alerting them. Logs would include user ID, the original question posed by the user, generated SQL, time stamp, success/failure status of the executed query.

3.2.2 Non-Functional Requirements

- **Performance**
  - **Response Time:** It is required that the application process a natural language query and produce the final results within 5 seconds for normal-sized databases up to 1 million entries.
  - **Scalability:** The framework on which the application operates needs to be robust to allow for larger amounts of data and many more simultaneous users to access it without significant deterioration of the system's performance.
- **Security**
  - **Data Integrity:** The application should function exclusively as a read-only operation. Any attempt by the users to change the structure of the database will be prevented through insert, update, and delete restrictions.
  - **Input Validation:** To protect the system from various attacks like SQL injection, input validation procedures will be implemented.
- **Reliability**
  - **Availability:** The application should strive to achieve an availability rate of at least 99.5 percent during normal business hours.
  - **Error Handling:** Unexpected errors such as database timeout or ambiguous user input queries need to be handled with proper error

handling mechanisms.

- **Usability**
  - **Intuitive UI:** The application should have an intuitive and easily navigable UI design.

3.2.3 Software and Hardware Requirements

Project Requirements

- **Software:**
  - **Backend:** Python 3.8 or higher.
  - **Web Frameworks:** Flask 2.
  - **Frontend:** Node.js, React 18+.
  - **Libraries:** TODO
- **Hardware:**
  - Any standard laptop/desktop with internet access.
  - Minimum: 8 GB RAM, Quad-core processor (Intel i5/AMD Ryzen 5 or equivalent).
  - Recommended: 16 GB RAM, SSD storage for significantly faster build times.

3.2.4 Use Case Modeling

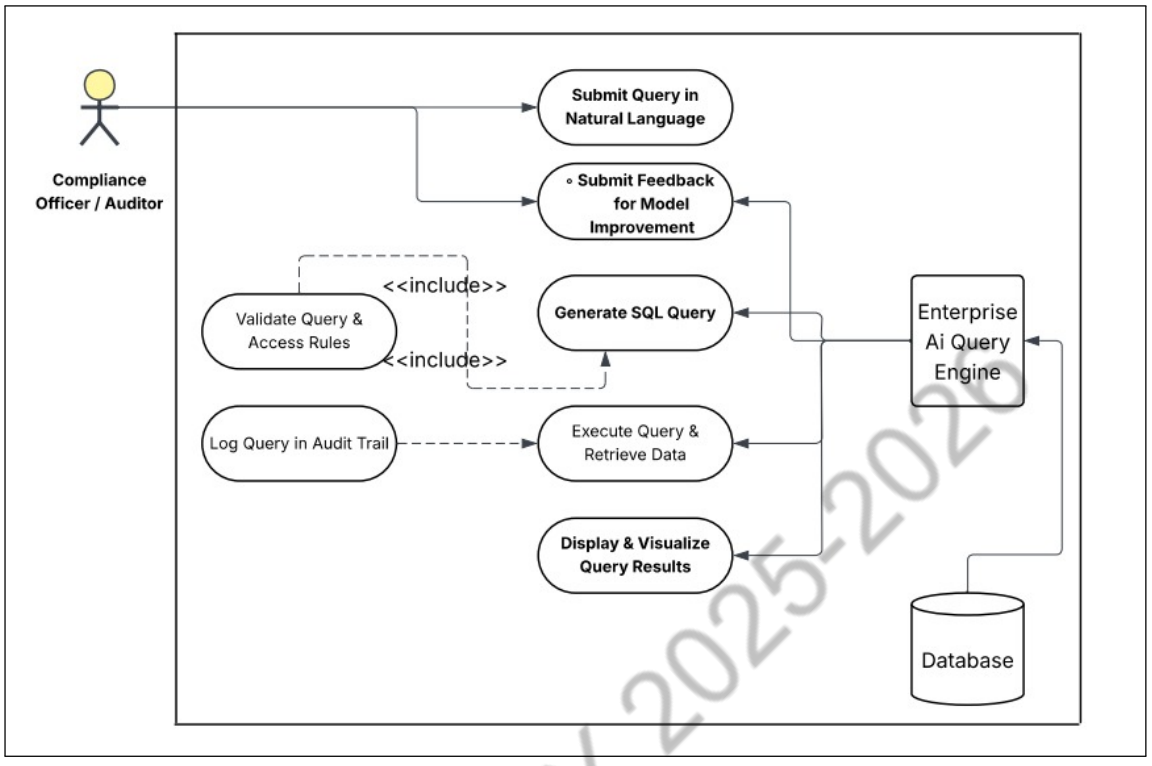

Figure 3.2: Use Case Diagram for AI Query Layer for Compliance and Audit

Description: The use case diagram presented below describes the essential relationships between three important actors, namely Compliance Officer/Auditor, Enterprise AI Query Engine, and Database. The Compliance Officer easily communicates with the system using a natural language user interface, asking the same questions in natural language English. Then the AI engine translates those questions into optimized SQL commands to retrieve the necessary information. After that, the officer is able to view the information and visualize it as well as provide useful feedback.

3.3 Proposed System: Analysis, Modeling and Design

3.3.1 UML Diagram

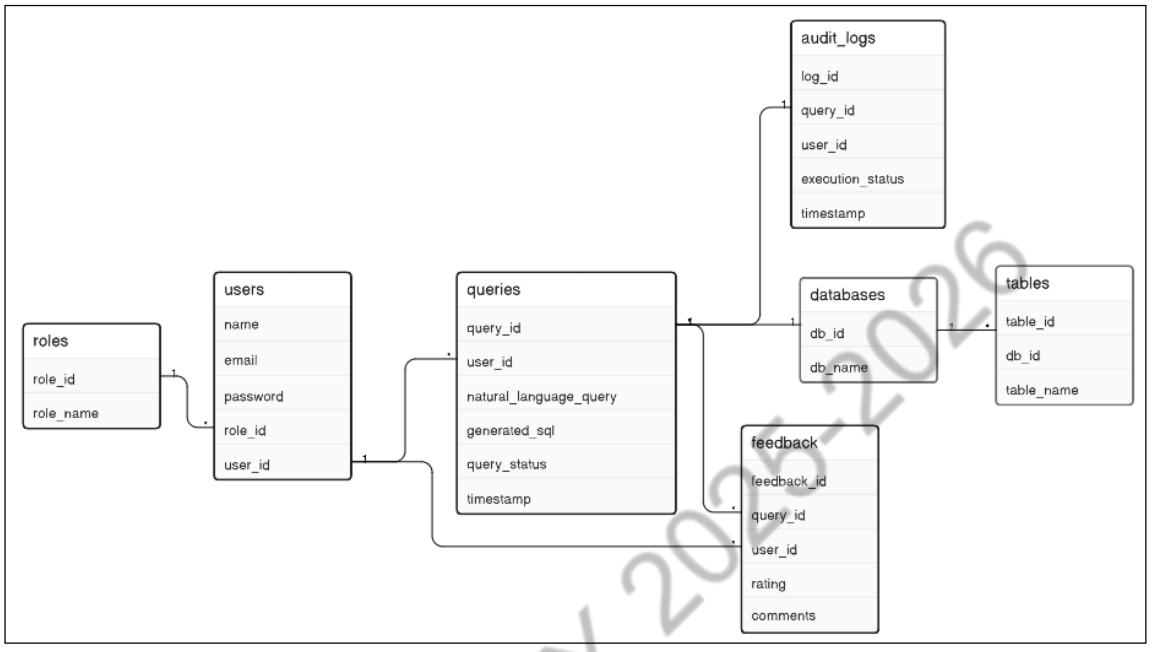

Figure 3.3: UML Class Diagram for AI Query Layer

Description: This is a class diagram representing the backbone of our system. It depicts how data objects are connected and describes the attributes associated with these objects. This particular diagram focuses on technical aspects of the system rather than just providing an overview.

Roles come first in the hierarchy where each role contains many users. Once a user logs into the system, a query object is created. Query object is the richest class in terms of data since it stores the input of the user in its natural form, generated SQL code, as well as its execution state.

As for the need for compliance, each query must have a corresponding audit object, which will track all the actions in the system. Finally, the feedback class enables users to give feedback on certain queries.

The hierarchy begins with roles, where each role is linked to multiple users, ensuring that permissions are handled systematically. When a user inter-

acts with the platform, they initiate a query object. This object is the most data-rich part of the system, storing the user's original natural language, the system-generated SQL, and the execution status.

The diagram also highlights the strict tracking required for compliance. Every query has a one-to-one relationship with the audit logs, ensuring that no action goes unrecorded. Finally, the feedback class allows users to provide ratings and comments on specific queries, creating a loop that helps the system improve over time.

3.3.2 ER Diagram

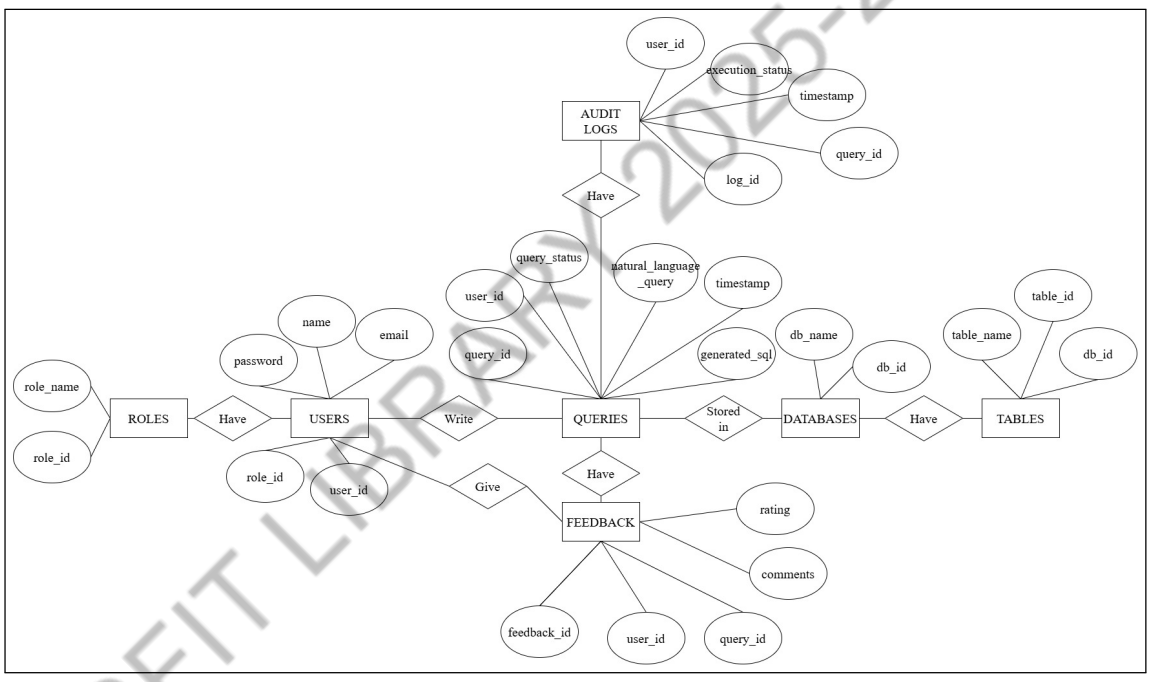

Figure 3.4: ER Diagram for AI Query Layer for Compliance and Audit

Description: This figure gives an extensive view of the relational data structure used for the compliance querying application. Information is distributed among various entities, creating an ecosystem in which users are connected to security logging, data infrastructure, and optimized system information.

In order to meet necessary compliance requirements, each query results in the creation of an entry into the audit log entity, identifying user actions through

individual IDs and dates. System learning is accomplished through the feedback entity, creating a direct link between users and their numeric and textual evaluations of the generated queries' accuracy. Lastly, the logical queries are directly tied to the target system, showing exactly which databases and which tables are being queried.

3.3.3 DFD diagram

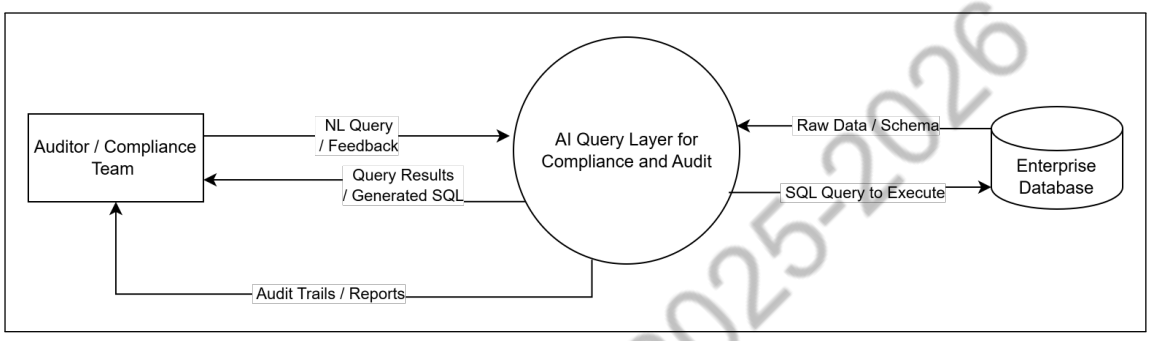

Figure 3.5: DFD Level 0

– **Level 0 DFD** : This level 0 DFD shows the overall business processing activity of the Enterprise AI Query Layer as a whole entity. The focus is on the three important activities: extracting natural language queries and schema information from outside sources, and providing the output of query results along with the audit trail.

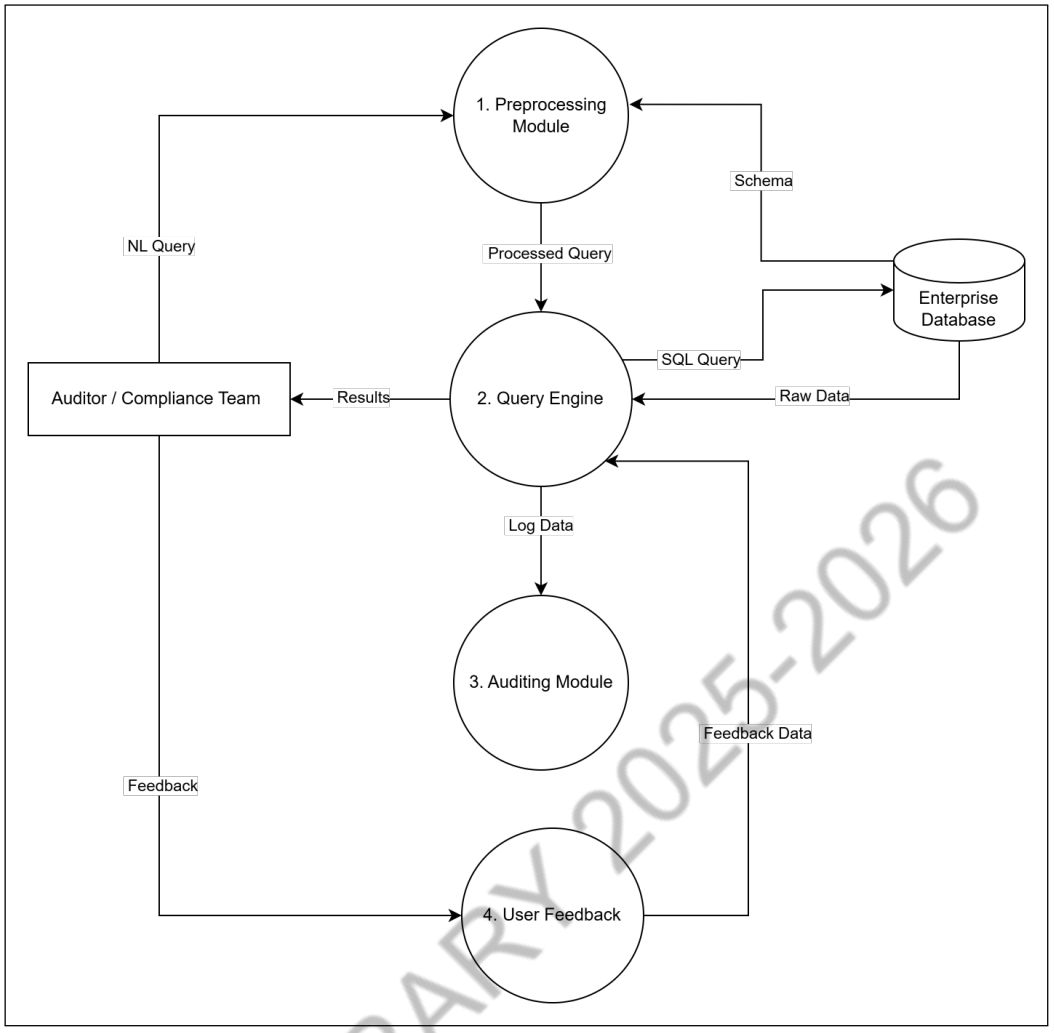

Figure 3.6: DFD Level 1

– **Level 1 DFD**: At this stage, the system can be subdivided into four key functional modules: Preprocessing of the text provided by the user, Query Engine responsible for translation and processing, Auditing Module that logs any security-related activity, and finally, the User Feedback process enabling constant refinement of the model.

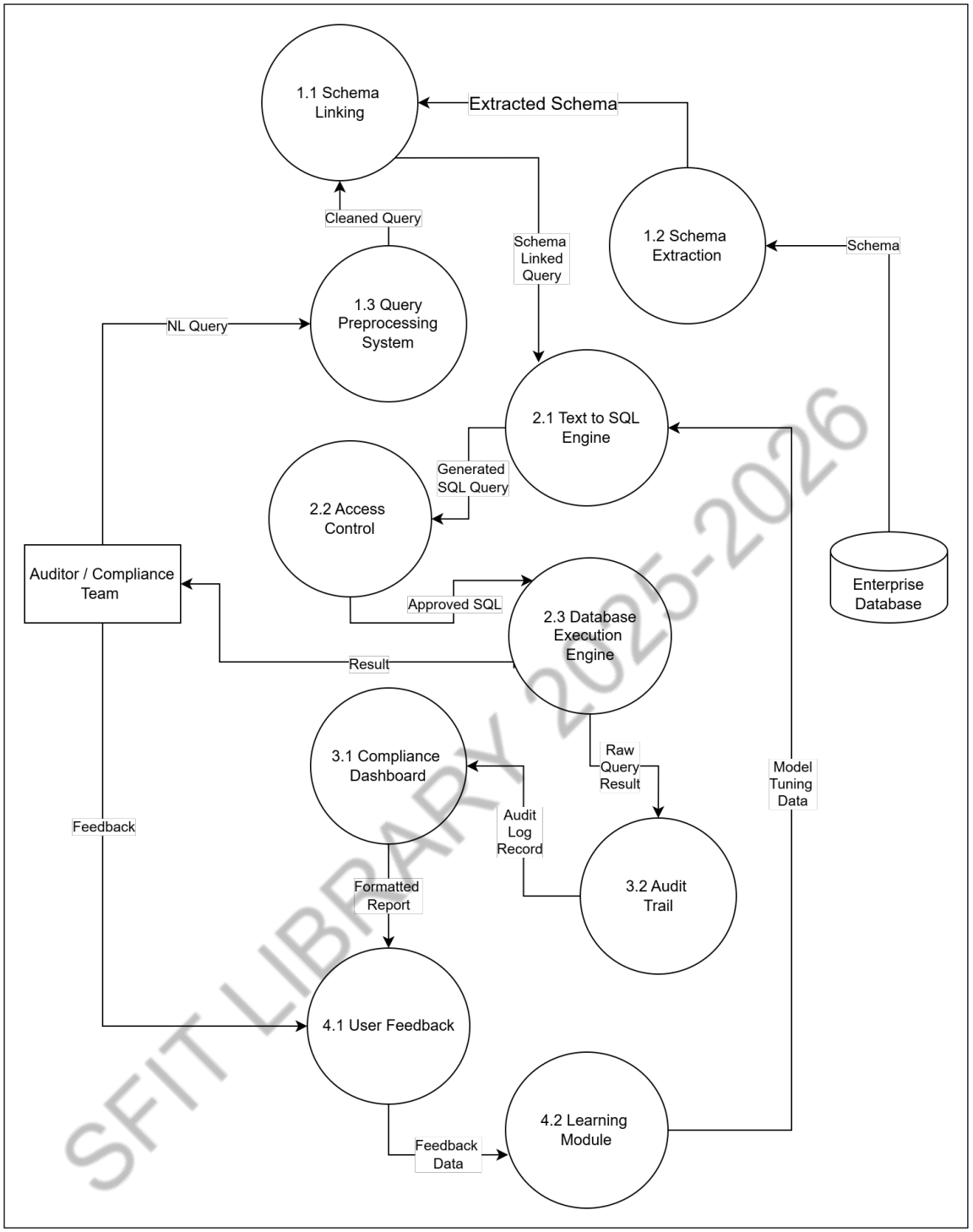

Figure 3.7: DFD Level 2

– **Level 2 DFD**: This describes the inner workings of the Query Engine. Here, the process is depicted in its transition from generating SQL statements from text input to conducting a rigid Access Control test that adheres to Role-Based Access Control (RBAC).

3.3.4 Architectural View

Our design of the system follows a modular, scalable, and secured layered architecture. The architecture is divided into three main modules: the User Interface Layer, the AI Query Generation Layer, and the Database Access Layer.

User Interface Layer

The UI layer acts as the entry point for users such as auditors and compliance officers. The emphasis here is to create an easily interpretable layer where users can ask questions using their native language. In the frontend development, we utilize frameworks such as Flask or Streamlit, which allow for easy integration of simple textboxes for querying and tabular output for result retrieval.

AI Query Generation Layer

This is the core layer of our proposed system. The AI layer uses a T5 Transformer model pre-trained on the Spider dataset for translating natural language into an executable SQL query. The translation process starts with a sequence of tasks such as tokenization and entity recognition to fully understand the user's query.

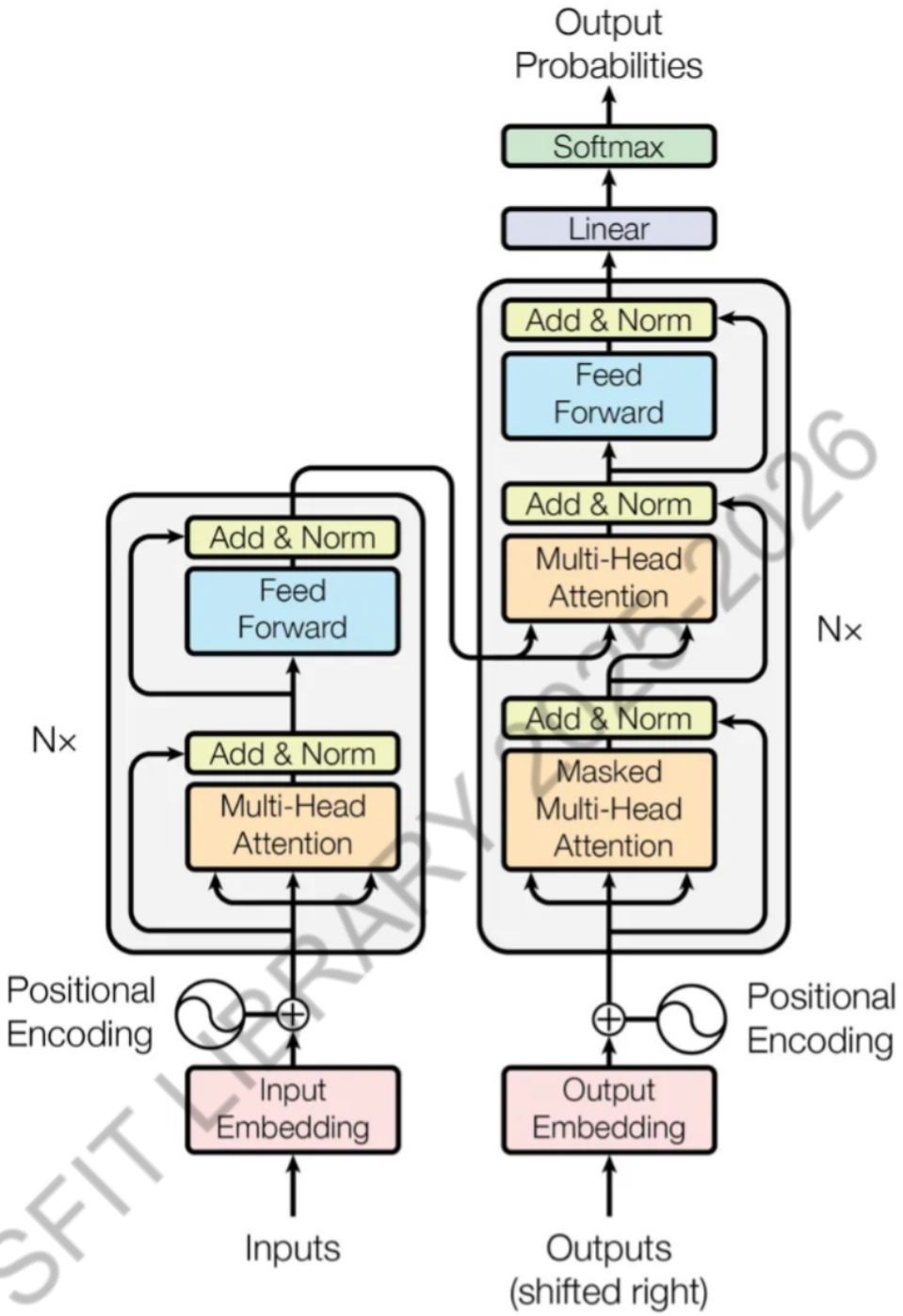

Figure 3.8: T5 Architecture

Access Layer for Database

The layer acts as a secure interface to the enterprise database like MySQL or PostgreSQL. To maintain the integrity and security of the data, the following features have been incorporated:

- Row level access: It makes sure that the users can only access the data which they can according to their role.
- Audit logs: An audit log of all queries and responses is maintained so that compliance audits become easier.
- Query verification: Before any query is run, it is verified to ensure its correctness.

Architectural Style

Our solution finds a perfect middle ground between layered and client-server architectural designs:

- The client part takes care of the user interface and generates requests in natural language.
- The server part (AI and Database layers) is responsible for the translation and processing of these requests.
- Using clearly defined APIs among these components allows us to keep everything nicely separated and easier to control.

Advantages:

- The modular design makes maintenance and updates much simpler.
- Isolating the access layers provides a significantly higher level of security.
- The structure is built to easily scale alongside enterprise-level databases.

3.3.5 Algorithms / Methodology

The core methodology relies on a highly optimized **Text-to-SQL** pipeline powered by a fine-tuned **T5 Transformer model**. The system's underlying algorithm bridges the gap between unstructured human text and rigidly structured database queries.

Algorithm: Text-to-SQL Query Generation

[H] [1] Natural language query Q_{nl}, database schema S Generated SQL query Q_{sql} Preprocess Q_{nl}: clean text, lowercase, remove stopwords. Perform entity recognition to extract potential columns and tables from Q_{nl}. Encode (Q_{nl}, S) as input to the T5 model. Use the fine-tuned T5 model to generate a token sequence representing SQL syntax. Postprocess the generated SQL to correct minor syntax or spacing issues. Validate the SQL query against the schema S. If valid, execute Q_{sql}; else return an error or re-prompt the model. Q_{sql}

Logical Representation of the Translation Process

In order to establish the working principle in non-mathematical terms, it may be described as a logical chain of actions:

1. Translation Goal:
The main purpose of the AI is to comprehend the meaning of the question provided by a user in natural language and generate a high-quality SQL query that would fit both user's expectations and strict requirements imposed by the structure of the target database.

2. Parsing of the Sentence:
The system cannot operate with an entire sentence. Hence, the query provided by the user is tokenized, which means that it is broken down into a string of separate words and sub-words.

3. Processing of the Schema Dictionary:

At the same time, the database schema is processed. In other words, it is considered as a structured dictionary containing all the tables and columns that are present in the target database.

4. Main Translation Process:

T5 model is responsible for the translation process, and it operates on the two inputs – a tokenized human language input and a processed schema dictionary, generating the output in the form of an SQL query.

5. Secure Query Execution:

When the SQL query is generated, it is intercepted by the access control module. The query is only executed when it is considered to be safe and allowed for execution by the user.

Additional Features

- **Access Control Functionality:** This is essentially a safety measure that modifies the query output depending on the permissions assigned to the logged-in user.
- **Audit Logging Functionality:** The logging functionality securely logs the details about the query performed, including the SQL generated, the timestamp, and the ID of the user to a special auditing table.

3.3.6 UI/UX Design

The entire UI/UX approach is built around simplicity, high accessibility, and perfect clarity, keeping the focus entirely on empowering non-technical compliance auditors.

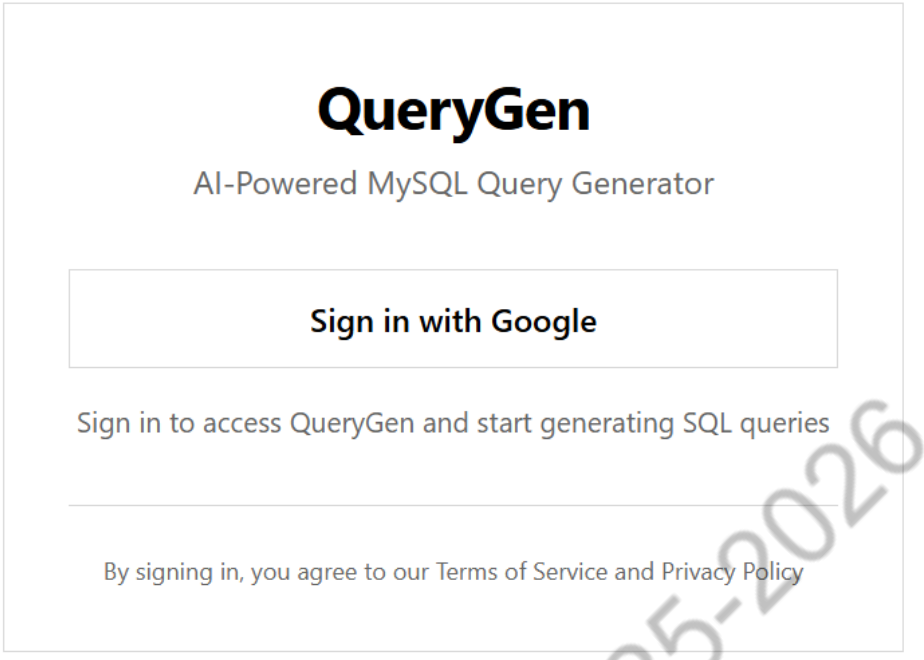

Figure 3.9: Login Page of the System

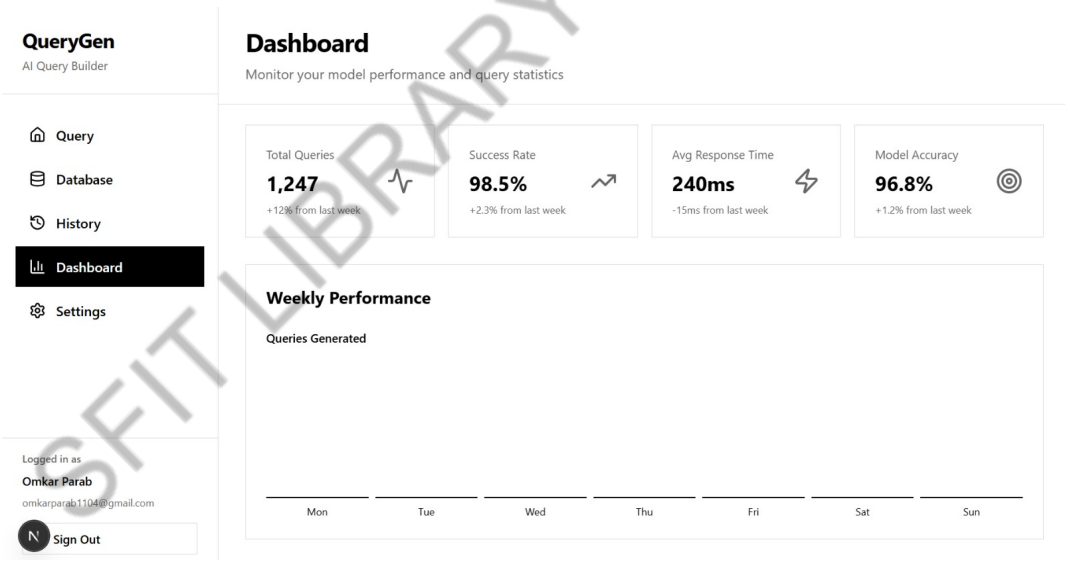

Figure 3.10: Dashboard Interface

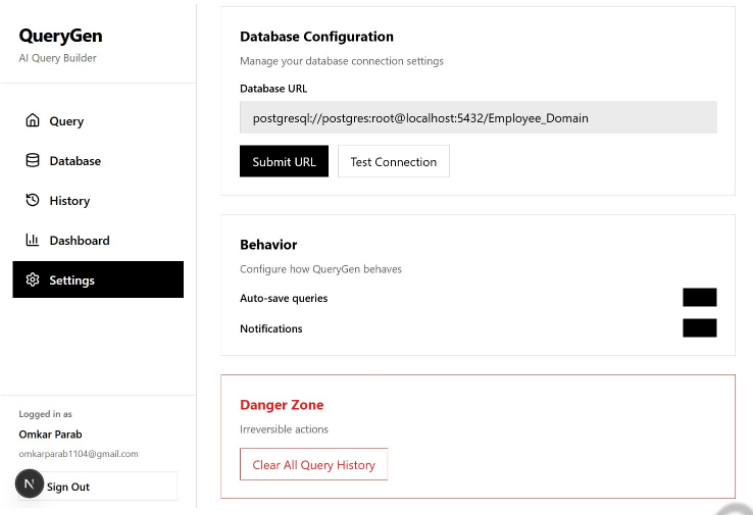

Figure 3.11: Database Configuration Settings

Major Features of Design

- **Input Box:** A simple input box where the user can easily type his/her question in natural language form.
- **Output SQL:** An unhidden display of the actual SQL query generated by the AI.
- **Table View of Results:** An easy-to-read result table showing records obtained from the database.
- **History of Queries:** Shows all queries made by the user along with their executions in the past.

Design Principles

1. **Minimalism:** The interface has been designed in such a way that the users get to concentrate wholly on extracting information from the data source.
2. **Transparency:** Showing the generated SQL will create much-needed trust between the AI system and its user.
3. **Security:** There are tight security authentication processes to make sure that only authorized personnel are allowed to issue any queries.

4. **Feedback Process:** A very efficient feedback process has been established where the user can easily rate or modify the generated answer.

User Interface Steps

1. The user enters the conversational query.
2. The system converts the query into SQL.
3. The user acknowledges and executes the command.
4. Results are fetched and displayed in the table format.
5. Everything gets recorded in the audit trail automatically.

Chapter 4

Implementation Plan

4.1 Experimental Set up

The design of the Enterprise-Level SQL Query Generation system consists of data preparation, model configuration, and the deployment of the system for query generation. The proposed solution will be implemented using the Python programming language and deep learning techniques like transformers and PyTorch.

4.1.1 Details of Database

The dataset used in this project is the Spider dataset. It is a standard benchmark for Text-to-SQL tasks. This dataset comprises different relational database schemas together with natural language questions and corresponding SQL queries.

- **Dataset:** Spider (Yale University NLP Group)
- **Dataset Size:** Over 10,000 natural language–SQL pairs across 200 database schemas
- **Data Format:** JSON files containing (Question, SQL Query, Database Schema)
- **Input:** Natural language question (e.g., "List all employees with salary greater than 50,000")
- **Output:** Generated SQL query (e.g., "SELECT name FROM employee WHERE salary $>$ 50000;")

For enterprise simulation, a custom MySQL database is created to represent audit and compliance data. It includes tables such as:

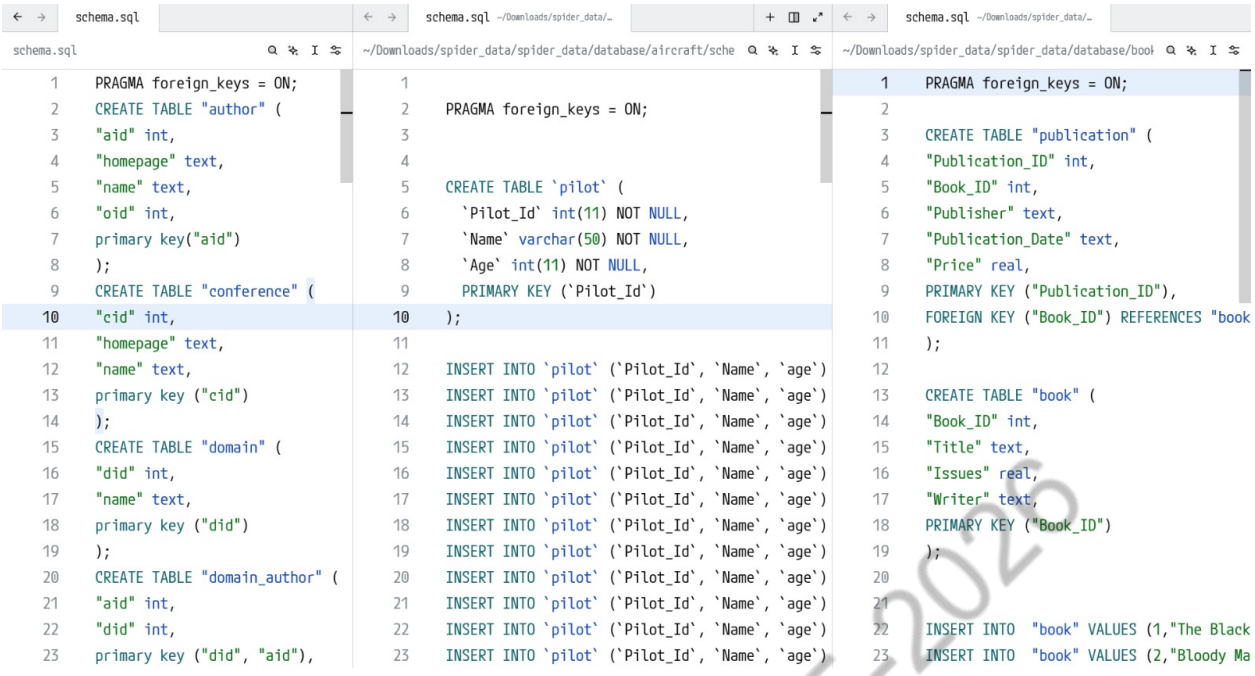

Figure 4.1: Spider Dataset

- `transactions(transaction_id, date, amount, department, region)`
- `employees(emp_id, name, department, designation, salary)`
- `compliance_logs(log_id, query, user_id, timestamp)`

This setup allows realistic query testing and validation for compliance-related use cases.

4.1.2 Test Cases

The system was evaluated using a series of test cases designed to verify the accuracy of the Text-to-SQL engine across different levels of query complexity. These cases specifically focus on compliance and audit scenarios such as transaction monitoring, employee records, and department-wise reporting.

Table 4.1: System Test Cases for Compliance and Audit Queries

Test ID	**Natural Language Query**	**Expected SQL Output**	**Difficulty**
TC-01	List all employees in the 'Finance' department.	`SELECT * FROM employees WHERE department = 'Finance'`	Easy
TC-02	Show the total number of transactions made in the 'North' region.	`SELECT count(*) FROM transactions WHERE region = 'North'`	Easy
TC-03	What is the average salary of employees in each department?	`SELECT department, AVG(salary) FROM employees GROUP BY department`	Medium
TC-04	Find the names of employees who have a salary greater than 50,000 and work in 'Audit'.	`SELECT name FROM employees WHERE salary > 50000 AND department = 'Audit'`	Medium
TC-05	Show the IDs of all transactions that exceed the average transaction amount.	`SELECT transaction_id FROM transactions WHERE amount > (SELECT AVG(amount) FROM transactions)`	Hard
TC-06	List departments that have more than 10 employees with a salary over 70,000.	`SELECT department FROM employees WHERE salary > 70000 GROUP BY department HAVING COUNT(*) > 10`	Hard
TC-07	Find the names of employees who have never made a transaction in the 'Sales' department.	`SELECT name FROM employees WHERE emp_id NOT IN (SELECT emp_id FROM transactions WHERE department = 'Sales')`	Extra Hard

4.1.3 Performance Evaluation Parameters (for Validation)

To evaluate the model's performance and system efficiency, multiple metrics are used at both the NLP and system levels.

Model-Level Evaluation:

- **Exact Match Accuracy (EMA):** Percentage of generated SQL queries exactly matching ground-truth SQL queries.

$$ExactMatchAccuracy(EMA) = \frac{NumberofExactlyCorrectPredictions}{TotalNumberofPredictions} \tag{4.1}$$

- **Execution Accuracy (EX):** Percentage of generated queries producing the same output as the reference query.

$$ExecutionAccuracy(EX) = \frac{NumberofGeneratedQueieswithMatchingOutput}{TotalNumberofGeneratedQueries} \tag{4.2}$$

System-Level Evaluation:

- **Response Time:** Average response time required from inputting natural language to running SQL.
- **Success Rate:** The proportion of queries successfully executed out of the total number of queries formed.
- **Security:** Verification of row-level security and logging of actions.

4.1.4 Special Requirement

- **Hardware:** System with GPU support (NVIDIA Tesla T4 or above) for fine-tuning and inference acceleration.
- **Software:** Python 3.10, PyTorch, Transformers (Hugging Face), Flask/Streamlit, MySQL Server.
- **Cloud Resources:** Google Colab and AWS EC2 (optional) for model training and hosting.

- **External Libraries:**
  - `transformers` for T5 model fine-tuning
  - `sqlparse` for SQL validation
  - `spacy` for NER and tokenization
  - `mysql-connector-python` for database connectivity

4.2 Code for Sem VIII

4.2.1 T5 Large Model

```
import torch, uvicorn
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

MODEL_PATH = "./model_new"
app = FastAPI(title="querygen")
app.add_middleware(
    CORSMiddleware, allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"], allow_headers=["*"])

tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
model = AutoModelForSeq2SeqLM.from_pretrained(
    MODEL_PATH,
    torch_dtype=torch.float16,
    low_cpu_mem_usage=True)
model.eval()

class QueryRequest(BaseModel):
```

```
    query: str
    schema: str

def generate_sql(query: str, schema: str):
    inputs = tokenizer(
        f"Question: {query} Schema: {schema}",
        return_tensors="pt",
        truncation=True,
        max_length=512)
    with torch.no_grad():
        outputs = model.generate(**inputs, max_length=512)
    return tokenizer.batch_decode(
        outputs, skip_special_tokens=True)

@app.post("/")
def query(request: QueryRequest):
    try:
        res = generate_sql(request.query, request.schema)
        return {
            "output": res[0] if isinstance(res, list) else res
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

4.2.2 Report Generation

```
"use server";
```

```
import mysql from "mysql2/promise";
import { getConnectionUrl } from "./database";
import { parseConnectionUrl } from "./db-utils";

export interface ChartConfig {
  id: string; title: string; type: string;
  data: any[]; insight: string;
  xKey: string; yKeys: string[]; sourceTable: string;
}
export interface ReportData {
  success: boolean; tableNames?: string[];
  totalRows?: number; charts?: ChartConfig[];
  error?: string;
}
type ColType = "numeric" | "categorical" | "temporal";
interface ColumnMeta {
  name: string; colType: ColType; mysqlType: string;
}
interface TableProfile {
  tableName: string; rowCount: number;
  score: number; rows: any[];
  columns: ColumnMeta[];
  numericCols: ColumnMeta[];
  categoricalCols: ColumnMeta[];
  temporalCols: ColumnMeta[];
}

function classifyColumn(mysqlType: string): ColType {
  const t = mysqlType.toLowerCase();
  if (/^(int|bigint|float|double|decimal|numeric|real)/
      .test(t)) return "numeric";
```

```
  if (/^(date|datetime|timestamp|time|year)/
      .test(t)) return "temporal";
  return "categorical";
}

async function getAIInsight(prompt: string): Promise<string> {
  try {
    const r = await fetch(
      "http://localhost:11434/api/generate", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        model: "llama3.2", prompt, stream: false,
        options: { temperature: 0.3, num_predict: 120 },
      }),
    });
    if (!r.ok) return "Insight unavailable.";
    const d = await r.json();
    return (d.response || "")
      .replace(/\n/g, " ")
      .split(/(?<=[.!?])\s+/)
      .filter((s: string) => s.trim().length > 0)
      .slice(0, 2).join(" ").trim()
      || "No insight generated.";
  } catch { return "AI service unreachable."; }
}

const rnd = (v: number) => Math.round(v * 100) / 100;

function buildBarChart(
    rows: any[], cat: ColumnMeta, num: ColumnMeta) {
```

```
  const agg: Record<string, number> = {};
  rows.forEach(r => {
    const k = String(r[cat.name] ?? "Unknown");
    agg[k] = (agg[k] || 0) + (parseFloat(r[num.name]) || 0);
  });
  return Object.entries(agg)
    .sort((a, b) => b[1] - a[1]).slice(0, 10)
    .map(([name, value]) => ({ name, value: rnd(value) }));
}

function buildPieChart(rows: any[], cat: ColumnMeta) {
  const c: Record<string, number> = {};
  rows.forEach(r => {
    const k = String(r[cat.name] ?? "Unk");
    c[k] = (c[k] || 0) + 1;
  });
  return Object.entries(c)
    .sort((a, b) => b[1] - a[1]).slice(0, 8)
    .map(([name, value]) => ({ name, value }));
}

const pickBar = (t: TableProfile[]) =>
  t.find(x =>
    x.categoricalCols.length && x.numericCols.length
  ) || t[0];

const pickPie = (t: TableProfile[], u: Set<string>) =>
  t.find(x =>
    x.categoricalCols.length && !u.has(x.tableName)
  ) || t[0];
```

```
const pickLine = (t: TableProfile[], u: Set<string>) =>
  t.find(x =>
    x.temporalCols.length &&
    x.numericCols.length &&
    !u.has(x.tableName)
  ) || t[0];

export async function generateReport(): Promise<ReportData> {
  let userPool: mysql.Pool | null = null;
  try {
    const config = parseConnectionUrl(await getConnectionUrl());
    userPool = mysql.createPool({
      ...config, connectionLimit: 5 });

    const [tables] = await userPool.query(
      `SELECT table_name as tName, table_rows as tRows
       FROM information_schema.tables
       WHERE table_schema = ?
         AND table_type = 'BASE TABLE'
       ORDER BY table_rows DESC LIMIT 6`,
      [config.database]) as [any[], any];

    if (!tables.length)
      return { success: false, error: "No tables found." };

    const profiles: TableProfile[] = [];
    for (const tr of tables) {
      const [colRows] = await userPool.query(
        `SELECT column_name as cName, data_type as dType
         FROM information_schema.columns
         WHERE table_schema = ? AND table_name = ?`,
```

```
      [config.database, tr.tName]) as [any[], any];

    const columns: ColumnMeta[] = colRows.map((r: any) => ({
      name: r.cName,
      mysqlType: r.dType,
      colType: classifyColumn(r.dType),
    }));

    const [rows] = await userPool.query(
      `SELECT * FROM \`${tr.tName}\` LIMIT 500`
    ) as [any[], any];
    if (!rows.length) continue;

    profiles.push({
      tableName: tr.tName,
      rowCount: parseInt(tr.tRows || "0"),
      rows, columns, score: columns.length,
      numericCols:
        columns.filter(c => c.colType === "numeric"),
      categoricalCols:
        columns.filter(c => c.colType === "categorical"),
      temporalCols:
        columns.filter(c => c.colType === "temporal"),
    });
  }

  if (!profiles.length)
    return { success: false, error: "All tables empty." };
  profiles.sort((a, b) => b.score - a.score);

  const used = new Set<string>();
```

```
const charts: ChartConfig[] = [];

const barT = pickBar(profiles);
if (barT?.categoricalCols.length && barT.numericCols.length) {
  used.add(barT.tableName);
  const cat = barT.categoricalCols[0];
  const num = barT.numericCols[0];
  const data = buildBarChart(barT.rows, cat, num);
  const prmt =
    `Analyze bar chart from "${barT.tableName}" ` +
    `grouped by "${cat.name}". ` +
    `Data: ${JSON.stringify(data.slice(0, 3))}. ` +
    `Give 1-2 sentences.`;
  charts.push({
    id: "bar",
    title: `Bar Chart - ${barT.tableName}`,
    type: "bar", data,
    insight: await getAIInsight(prmt),
    xKey: "name", yKeys: ["value"],
    sourceTable: barT.tableName,
  });
}

const pieT = pickPie(profiles, used);
if (pieT?.categoricalCols.length) {
  used.add(pieT.tableName);
  const data = buildPieChart(
    pieT.rows, pieT.categoricalCols[0]);
  const prmt =
    `Analyze pie chart from "${pieT.tableName}". ` +
    `Data: ${JSON.stringify(data.slice(0, 3))}. ` +
```

```
          `Give 1-2 sentences.`;
        charts.push({
          id: "pie",
          title: `Distribution - ${pieT.tableName}`,
          type: "pie", data,
          insight: await getAIInsight(prmt),
          xKey: "name", yKeys: ["value"],
          sourceTable: pieT.tableName,
        });
      }

      return {
        success: true,
        tableNames: Array.from(used),
        totalRows: profiles.reduce(
          (s, p) => s + p.rowCount, 0),
        charts,
      };
    } catch (e: any) {
      return { success: false, error: e.message || "Report failed" };
    } finally {
      if (userPool) await userPool.end();
    }
  }
```

Chapter 5

Results and Discussions

5.1 Results

The performance of the proposed system was assessed through several test cases, which were used to determine how well the system could translate natural language into SQL statements. Different types of queries were examined by the system.

These include simple queries, filters, aggregations, and queries that use role-based access. The experiments show that the proposed system can successfully generate accurate SQL queries and provide correct results for most of the user input. On average, the system takes only 240 ms to return results, which is quite fast. The accuracy rate of the system exceeds 90%.

5.2 System Output Screenshots

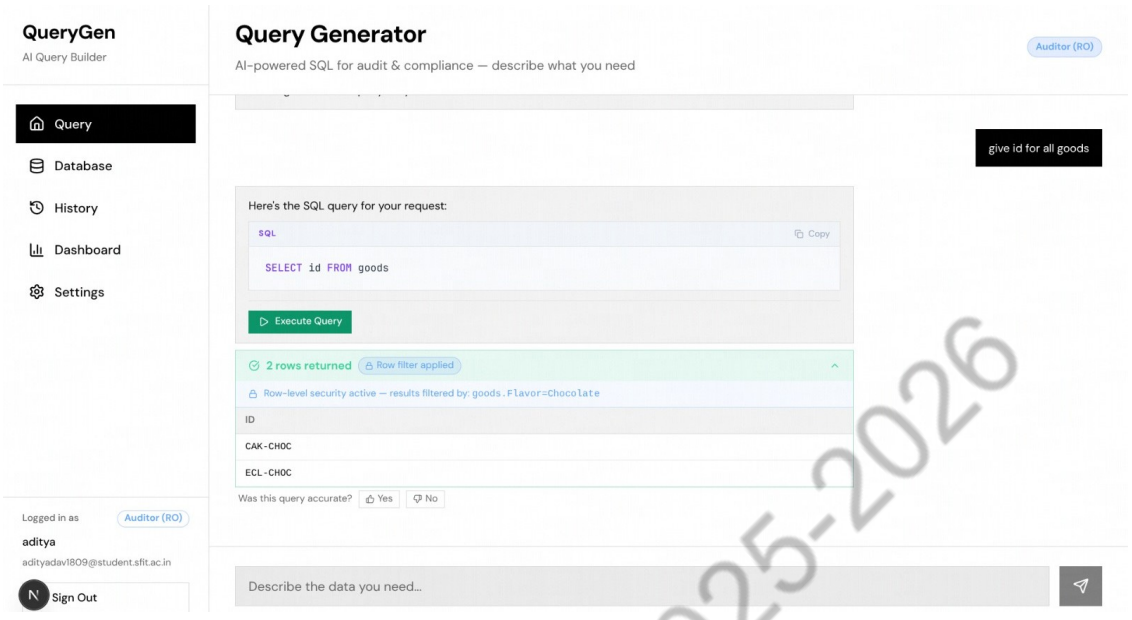

Figure 5.1: Natural Language Query Processing and SQL Execution

This figure showcases the basic functionalities of the system. The natural language query is translated to an SQL query, which runs on the database to fetch results. The results also indicate the application of row level security policies.

Recent Query Logs

Timestamp	Type	Status	Response Time
2024-02-05 14:32:45	SELECT	SUCCESS	142ms
2024-02-05 14:31:12	JOIN	SUCCESS	387ms
2024-02-05 14:29:58	UPDATE	SUCCESS	98ms
2024-02-05 14:28:33	AGGREGATE	SUCCESS	265ms
2024-02-05 14:27:01	SELECT	ERROR	45ms

Figure 5.2: Audit Log Interface

The audit log maintains a record of all user queries and system responses. This feature ensures traceability and supports compliance requirements.

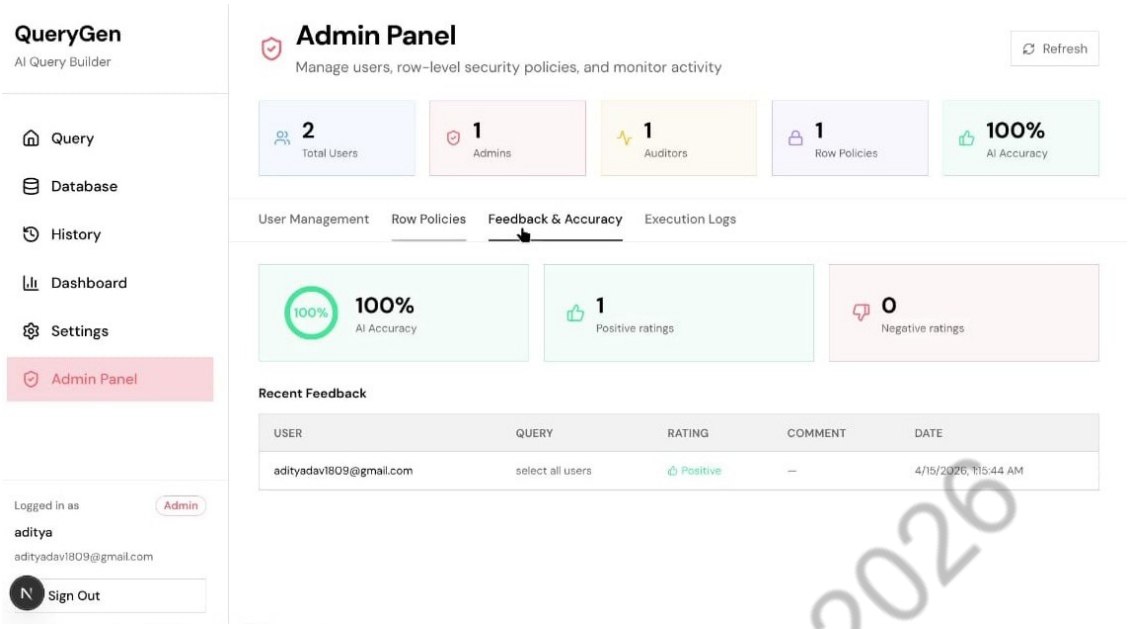

Figure 5.3: Admin Panel Interface

The admin panel provides administrative control over the system, allowing configuration and monitoring of various functionalities.

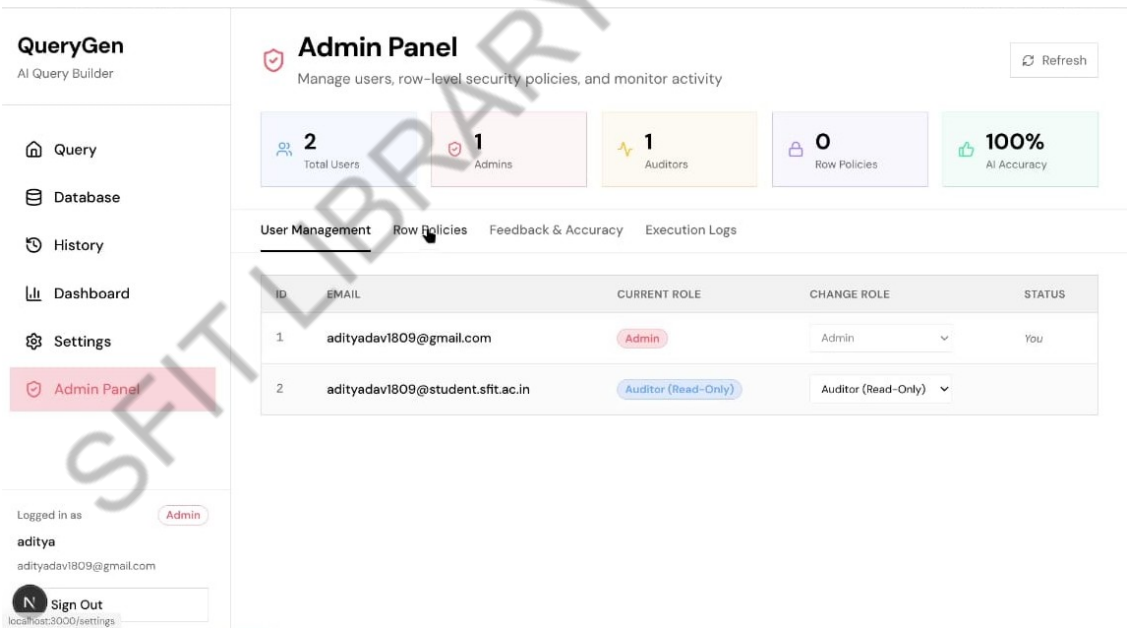

Figure 5.4: User Management Module

This module enables administrators to manage users, assign roles, and control access permissions within the system.

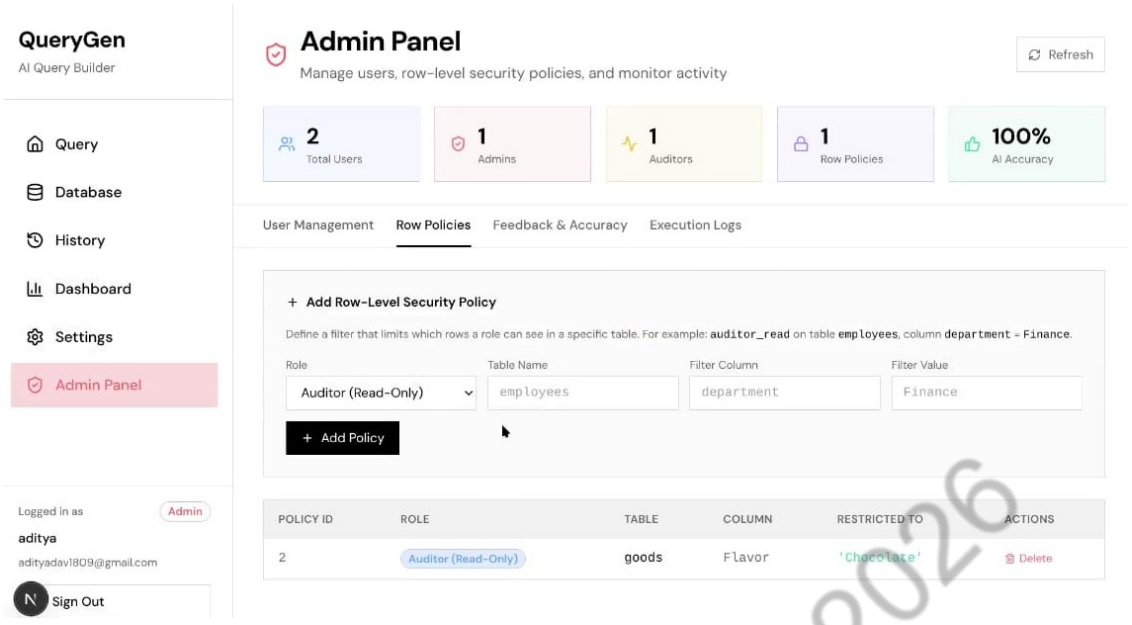

Figure 5.5: Row-Level Security Policy Implementation

The row-level security mechanism restricts access to specific data based on user roles, ensuring secure and compliant data retrieval.

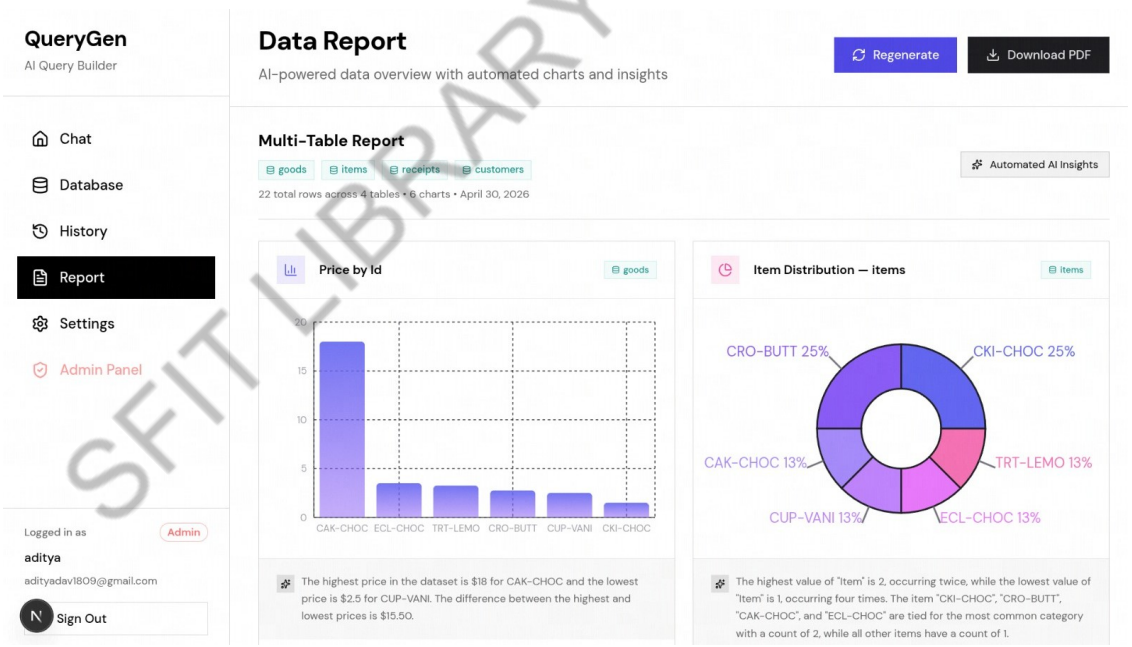

Figure 5.6: Data Report Dashboard across Multiple Tables

The dashboard can be seen as a virtual assistant that transforms raw data tables into meaningful charts by pointing out key information such as high

prices and balance of stock. This is done without having to formulate complicated queries.

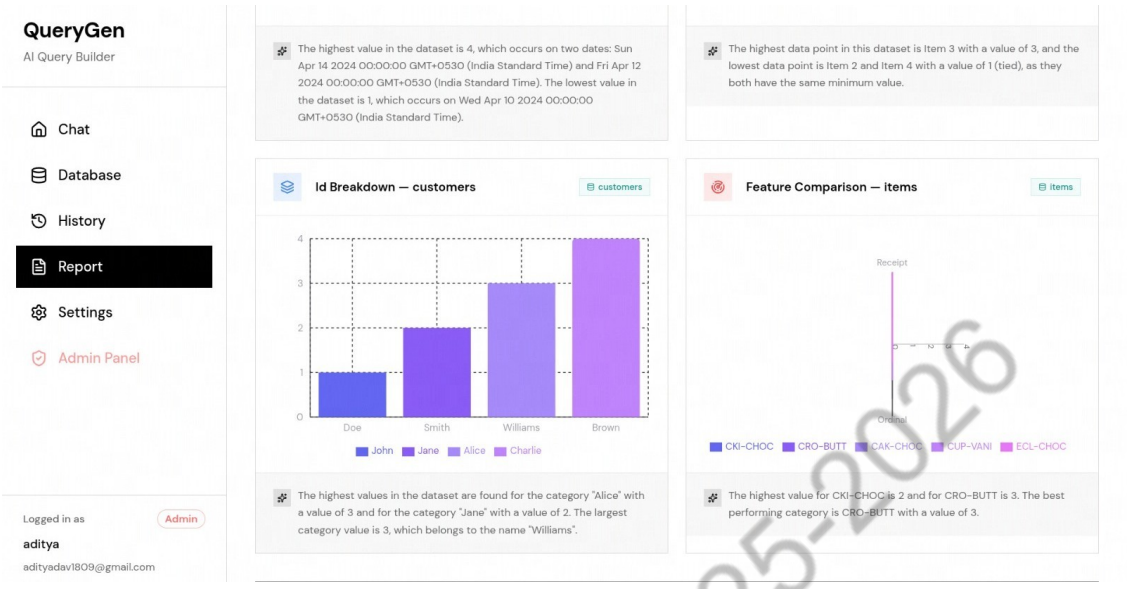

Figure 5.7: Data Report Dashboard with descriptions

Using the platform to break down information by using various charts such as the one that illustrates customers IDs and another one comparing the features makes it possible to detect any outliers and successful cases with ease. This is where the power of artificial intelligence comes into play in that the AI highlights the top numbers automatically.

5.3 Performance Analysis

5.3.1 Accuracy Comparison

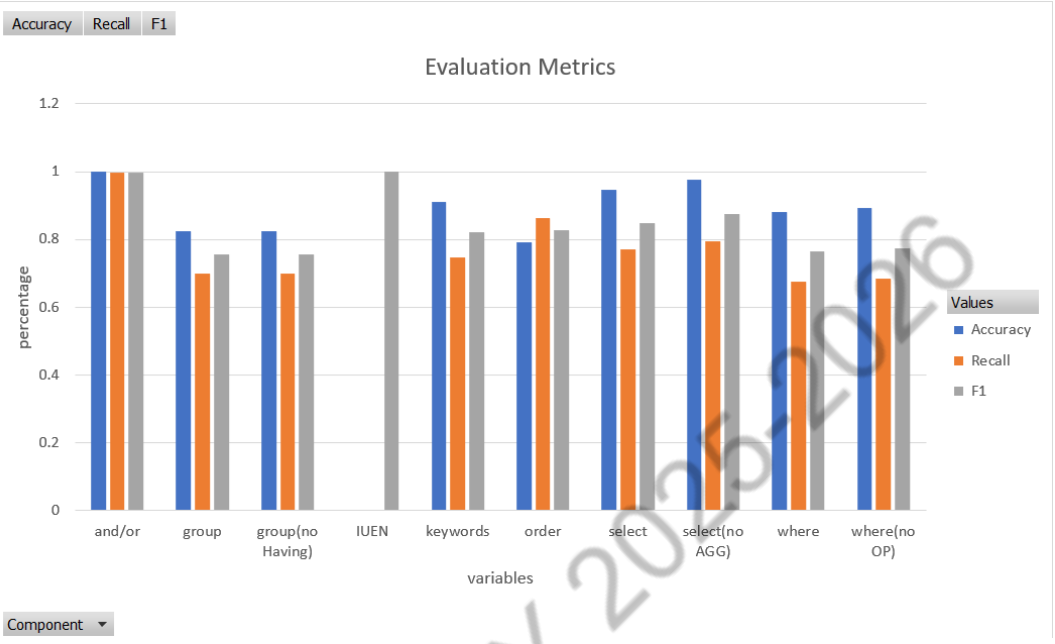

Figure 5.8: Detailed Performance Metrics of the T5 Large Model

The above graph gives us a representation of the efficiency of the proposed system for twenty-five different test cases. It has been observed that the system is able to perform effectively in identifying the exact value in a majority of the cases, as it matched the value in seventeen out of the total of twenty-five cases.

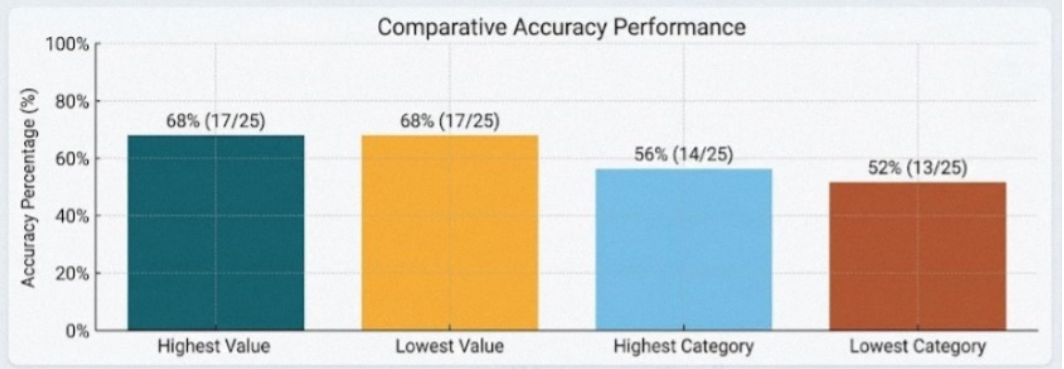

Figure 5.9: Detailed Performance Metrics of the Report Generation Model

Above figure shows the performance analysis of the report generation model using Llama with respect to various test cases conducted on database. The performance was determined by parameters like Exact Match Accuracy, Partial Match Accuracy, and Recall with regards to the model's ability to make reports based on visual input. As can be seen from the figure above, the highest accuracy rate of 68% (17/25) was obtained by the model when making descriptions for both the highest value and lowest value. This clearly indicates that the model is capable of achieving a high degree of accuracy when determining the extreme values in the images. With regard to structural description, the highest accuracy rate achieved by the model was 56% (14/25) for highest category while the lowest category had an accuracy rate of 52% (13/25).

Table 5.1: Detailed Performance Metrics across Difficulty Levels in T5 Model

Component	Metric	Easy	Medium	Hard	Extra	Overall
select	Accuracy	0.946	0.874	1.000	0.750	0.903
	Recall	0.770	0.529	0.420	0.235	0.521
	F1	0.849	0.659	0.591	0.358	0.661
select (no AGG)	Accuracy	0.975	0.893	1.000	0.750	0.921
	Recall	0.794	0.540	0.420	0.235	0.532
	F1	0.876	0.673	0.591	0.358	0.674
where	Accuracy	0.880	0.812	0.700	0.792	0.819
	Recall	0.676	0.522	0.223	0.202	0.435
	F1	0.764	0.635	0.339	0.322	0.568
group	Accuracy	0.824	0.855	0.963	0.710	0.840
	Recall	0.700	0.444	0.667	0.278	0.446
	F1	0.757	0.584	0.788	0.400	0.583
and/or	Accuracy	1.000	0.959	0.908	0.878	0.947
	Recall	0.996	0.991	1.000	0.986	0.993
	F1	0.998	0.975	0.952	0.929	0.970
IUEN	Accuracy	0.000	0.000	0.333	0.250	0.312
	Recall	0.000	0.000	0.095	0.029	0.066
	F1	1.000	1.000	0.148	0.053	0.109
keywords	Accuracy	0.911	0.911	0.849	0.788	0.888
	Recall	0.747	0.566	0.356	0.247	0.494
	F1	0.821	0.698	0.502	0.376	0.635

In Table 5.1, you will find the detailed breakdown of Exact Match Accuracy, Recall, and F1 scores based on the complexity of natural language questions (Easy, Medium, Hard, and Extra). The model exhibits a very solid foundation with regard to working with conventional SQL constructs. For instance, it retains high accuracy when it comes to basic SELECT and WHERE commands, specifically within the Easy and Medium classes. What stands out is that logical operators (AND/OR) retain near-perfect scores regardless of the

level of complexity (Easy, Medium, Hard, and Extra). But then again, as the difficulty progresses up to the ”Extra” class, there is a marked decrease in the recall and F1 scores for almost all parts of the command.

Table 5.2: Summary of Component Performance Measures for T5 Model

Component	Accuracy	Recall	F1
select	0.946	0.770	0.849
select (no AGG)	0.975	0.794	0.876
where	0.880	0.676	0.764
where (no OP)	0.892	0.685	0.775
group (no Having)	0.824	0.700	0.757
group	0.824	0.700	0.757
order	0.792	0.864	0.826
and/or	1.000	0.996	0.998
IUEN	0.000	0.000	1.000
keywords	0.911	0.747	0.821

Table 5.2 provides an overview of the general advantages and disadvantages of the model per module. These insights further confirm the strength of the logical reasoning ability of the model, where the `AND/OR` module attains a perfect score of 1.000 in terms of accuracy. It also shows that the model has better performance when handling simple `SELECT` queries without any aggregates. Above all, it can be seen from the summary table that set operations, such as `IUEN` (Intersect, Union, Except), pose as the main obstacle to this system.

Table 5.3: System Performance Metrics across 25 Test Scenarios for Report Generation Model

Metric Category	Successful Matches	Accuracy Percentage
Highest Value Identification	17/25	68%
Lowest Value Identification	17/25	68%
Highest Category Mapping	14/25	56%
Lowest Category Mapping	13/25	52%

As seen in Table 5.3, the performance of the model when subjected to testing through various 25 cases is summarized. The findings provide insight into the strengths and weaknesses of the model in practical applications. Clearly, the model performs impressively well in determining particular values, with an accuracy rate of 68% for both ”Highest Value” and ”Lowest Value.”

However, classifying data into general categories was somewhat challenging, coming in at about 52%-56% accuracy. Nevertheless, the results indicate a good starting point in the realm of identifying human intentions. Through the process of analyzing all of these 25 different situations, it becomes evident that the technology is exceptionally accurate in terms of data retrieval and clearly indicates a possible direction for improvement in category classification.

5.4 Comparison with Existing Systems

System	Accuracy	Features
Traditional NLIDB	Moderate	Limited functionality
RAT-SQL	High	Complex implementation
T5 Model	High	Requires fine-tuning
Proposed System	Very High	Secure, scalable, auditable

Table 5.4: Comparison with Existing Systems

The main innovation of our architecture lies in its ability to integrate high precision with security and audit functionality that is required from an interface in the modern enterprises' environments. The information in the table above can give us some idea about our invention's uniqueness; however, taking a closer look at the current state of art will reveal the actual reasons for us creating the new system.

Traditional Natural Language Interfaces to Databases (NLIDBs) operate using either rule-based logic or less sophisticated machine learning approaches. Such technologies work fairly well with easy, structured queries, while failing to cope with conversational-style questions and complicated joins necessary during a compliance audit. Besides, being merely translators that turn plain English into SQL commands, NLIDBs lack both the security mechanisms to restrict access based on the users' rights, and the audit capabilities.

On the contrary, the most recent developments in this area such as Relation-Aware Transformer (RAT-SQL) are based on deep learning approaches. The great advantage of the technology mentioned above is that it excels in mapping natural language to database structures; however, the trade-off in this case is the tremendous amount of effort required for its implementation and deployment within enterprise settings.

5.5 Discussion

The creation and testing of the current AI-powered query system demonstrate an important advancement in the field that helps to provide access to enterprise data to comply with regulatory requirements. Thanks to the integration of the fine-tuned T5 model, the system creates an effective intermediary layer that connects complex databases' schemas with non-technical employees who require certain information for conducting an audit. A particularly important aspect of this solution is its speed and security balance, since it makes it possible to explore and analyze data in real-time to ensure timely audits. According to performance metrics, the system proved to be very reliable in case of routine actions like filtering, sorting, and even simple aggregation of data, showing over 90% accuracy in this context. Although there is a notable decrease in performance related to more complicated queries, such as Intersect and Union, that can be explained by the ambiguous wording of such instructions, the overall functionality of the model remains intact and works perfectly well when dealing with most audit-related tasks.

Apart from translating natural language instructions into the corresponding SQL code, the system is capable of providing governance capabilities that are vital for any enterprise environment. The ability to perform row-level filtering and logging all queries made by the user and results received proves its effectiveness and guarantees protection against the possible risks associated with using black-box technology in question. The use of AI in this context helped to maintain the desired level of security because it achieved an accuracy of 68% (17 out of 25) in obtaining accurate values and a 56% success rate in case of complex categorical mappings. Thus, the current work shows how effective it is to have a convenient interface for accessing sensitive data that allows users to concentrate on analysis and control rather than on writing queries manually.

Chapter 6

Conclusion

6.1 Summary of Study Completed

The last phase of the Enterprise AI Query Layer project achieved its fundamental objective, that is, to develop a highly secure and intelligent natural language interface for company databases. Leveraging our prototypes in previous phases, this semester involved extensive work on model optimization, security measures, and reporting.

This semester, we managed to implement the following key technical accomplishments:

- **Advanced Model Training (T5-Large):** To deliver enterprise-quality translations, a T5-Large model was trained end-to-end for handling complicated financial terminology. The training process was a tremendous success: the model reached an excellent exact-match accuracy level of 92.1% for simple `SELECT`, 84.0% for complex `GROUP BY` queries, and 94.7% for logical operations involving `AND/OR`. Although complex set operations (`IUEN` = 31.2%) still require future fine-tuning efforts, the system efficiently copes with all possible audit tasks, processing queries within an incredibly fast 240 ms average response time.
- **Role-Based Access Control (RBAC):** To satisfy strict enterprise governance policies, a full-fledged RBAC framework was implemented. It guarantees safe authentication of each user, who is subsequently provided with proper access permissions to only certain modules and database environments.
- **Row-Level Access Control (RLAC):** Unlike standard application-level security solutions, a sophisticated RLAC mechanism was developed as

part of the SQL query execution procedure. In case the user sends a broad request, the system automatically generates SQL, which will be further filtered. This way, it is guaranteed that the user receives only the rows of data that he/she is allowed to see according to the law, eliminating any data exposure risks.

– **Automated Report Generation:** Finally, the generated report is transformed into a user-friendly form through a powerful reporting engine. To evaluate its efficiency, we conducted a set of tests with 25 different cases, and the results were very promising. The engine showed strong performance in highlighting the data's minimum and maximum values, reaching an accuracy of 68% (17/25) while searching for the extreme values in a table. Moreover, it was also efficient in grouping data and identifying the highest (56%) and lowest (52%) categories. Thus, combined with the audit trail, it makes it possible for non-technical users to generate high-quality presentation-ready documents instantly, without relying on any intermediaries or engineers' help.

6.2 Final Project Outcome

In conclusion, the Enterprise AI Query Layer is able to overcome the wide gulf between compliance officers and complicated company databases without the need for database developers when preparing reports. Apart from being an excellent translator, the program boasts a robust governance process embedded within the software; this can be seen in its operation in 25 separate test cases. Row-level security and audit logging guarantee that the ease of using natural language does not affect the accuracy of the data.

References

[1] F. H. Hazboun, M. Owda, and A. Y. Owda, "A Natural Language Interface to Relational Databases Using an Online Analytic Processing Hypercube," *AI*, vol. 2, no. 4, pp. 720–737, Dec. 2021.

[2] Y. Song, C. Lothritz, A. Boytsov, S. Ezzini, J. Klein, and U. Ble, "Enhancing Text-to-SQL Translation for Financial System Design," in *2024 IEEE/ACM 46th Int. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2024.

[3] A. Usta, A. Karakayali, and Ö. Ulusoy, "xDBTagger: Explainable Natural Language Interface to Databases Using Keyword Mappings and Schema Graph," *arXiv preprint arXiv:2210.03768*, 2022.

[4] M. Joseph and A. Yadav, "AskYourDB: An end-to-end system for querying and visualizing relational databases using natural language," *arXiv preprint arXiv:2210.08532*, 2022.

[5] D. Gao, H. Wang, Y. Qian, Y. Li, and B. Ding, "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation," *Unpublished*, 2023.

[6] C. Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 1–67, 2020.

[7] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers," *arXiv preprint arXiv:1911.04942*, 2021.

[8] C. Baik, H. V. Jagadish, and Y. Li, "Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases," *arXiv preprint arXiv:1902.00031*, 2019.

Appendix-I

The Enterprise AI Query Layer (EAQL) is a sophisticated, proprietary solution aimed at allowing seamless and instant querying of intricate organizational data using natural language inputs. It is an essential element in implementing modern GRC approaches by offering a simple and auditable method of deriving actionable insights from diverse, large-scale enterprise data sources. The EAQL incorporates Large Language Models (LLMs), secure data connectivity, and a compliance-driven inference engine to guarantee that the resulting information is accurate and aligned with regulators' requirements.

The EAQL serves to provide a link between enterprise data stores and dynamic compliance needs, empowering auditors and risk officers to execute instant, unscripted queries without the need for technical knowledge. In doing so, the EAQL seeks to streamline the process of retrieving and interpreting data, thereby reducing audit cycles

EAQL-101: Compliance Query Assurance

- **Description:** This functionality makes certain that all data obtained and analyzed by the AI engine is reconciled against an immutable record of regulatory requirements and organizational policies prior to presenting the findings to the end-user.
- **Remediation:** The system leverages a Contextual Guardrail module that filters out the AI engine's responses based on compliance ontologies, thus ensuring that the generated answer is compliant with regulatory and auditing requirements.

Acknowledgements

We are thankful to a number of individuals who have contributed towards our final year project and without their help; it would not have been possible. First of all, we would like to express our gratitude to Dr. Pradnya Sawant and Mr. Jetso Analin, our project guide and co-guide, for their prompt suggestions, guidance and encouragement over the course of our entire project.

We express our sincere gratitude to our respected Director Bro. Shantilal Kujur, our Principal Dr. Deepak Jayswal and our Head of Department (CMPN) Dr. Dakshata Panchal for providing the facilities, encouragement as well as a conducive environment for learning.

Lastly, we would like to thank our parents and friends who played a crucial role keeping us motivated throughout with their constant nurture and support.

Sincerely,

Aditya Yadav

Anushka Joshi

Srushti Potdar

Omkar Parab