SQL to NoSQL Translator Mini Lab Project

Submitted By

Student Name	Student ID
Md.Sakibol Hasan Sohan	232-15-821
Dipankar Saha	232-15-862
Jihad Hossain Khan	232-15-664
Mir Md. Ziad	232-15-935
Rafa Tasnia	232-15-118

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE314: Compiler Design Lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

December 9, 2024

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Tamanna Sultana**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

____
Tamanna Sultana
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

Md.Sakibol Hasan Sohan ID:232-15-821 Dept. of CSE, DIU	
Dipankar Saha ID:232-15-862 Dept. of CSE, DIU	Jihad Hossain Khan ID:232-15-664 Dept. of CSE, DIU
Mir Md. Ziad ID:232-15-935 Dept. of CSE, DIU	Rafa Tasnia ID:232-15-118 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO's	Statements
CO1	Understand the working procedure of a compiler for debugging programs.
CO2	Analyze the role of syntax and semantics of Programming languages in compiler construction.
CO3	Apply the techniques, algorithms, and different tools used in Compiler Construction in the design and construction of the phases of a compiler's components.
CO4	Create a project by explaining complex computer engineering activities with the computer engineering community by performing effective communication through demonstration and presentation.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Introduction **1**
1.1 Introduction 1
1.2 Motivation 1
1.3 Objectives 2
1.4 Feasibility Study 2
1.5 Gap Analysis 3
1.6 Project Outcome 3

Proposed Methodology/Architecture **4**
2.1 Requirement Analysis & Design Specification 4
2.2 Overall Project Plan 7

Implementation and Results **9**
3.1 Implementation 9
3.2 Performance Analysis 10
3.3 Results and Discussion 11

Engineering Standards and Mapping **13**
4.1 Impact on Society, Environment and Sustainability 13
4.2 Project Management and Team Work 15
4.3 Complex Engineering Problem 15

Conclusion **17**
5.1 Summary 17
5.2 Limitation 17
1. Limited SQL grammar support 18
2. Basic handling of conditions 18
3. No semantic validation 18
4. User Interface is minimal 18
5. No multi-database support 18
6. No reverse translation 19
5.3 Future Work 19

References **20**

Chapter 1

Introduction

This chapter provides a comprehensive overview of the background, motivation, problem statement, and scope of the project. It also outlines the feasibility, existing gaps, and expected outcomes associated with the SQL to NoSQL Translator. The goal is to establish the context and necessity of this project in both academic and real-world environments, especially in the era of modern, data-driven applications.

1.1 Introduction

Data storage and retrieval have evolved significantly over the past decade. Traditional relational database systems such as MySQL, Oracle, and PostgreSQL rely heavily on SQL (Structured Query Language) to perform data operations. In contrast, the increasing demand for flexible, scalable, and schema-less data storage has accelerated the adoption of NoSQL databases such as MongoDB, Cassandra, and Firebase. Each follows a different query mechanism, often requiring developers to adapt to unfamiliar syntactic structures.

As the industry transitions into distributed, flexible systems, many developers — especially beginners — face challenges converting SQL queries to their NoSQL equivalents. This project addresses this challenge through the development of **a** SQL to NoSQL Translator, a compiler-like translation tool that takes an SQL SELECT statement and converts it into a MongoDB query format. Using Flex (Lex) for lexical analysis and Bison (Yacc) for parsing, the project demonstrates core compiler phases while producing a practical tool that bridges a meaningful skill gap in database systems.

This project is developed as part of the Compiler Design Lab (CSE314)**,** allowing students to apply theoretical concepts in a real-world system while exploring how language parsing, grammar rules, and token generation interact in complex software engineering tasks.

1.2 Motivation

The motivation for undertaking this project arises from several academic and industrial factors. First, SQL remains the foundational query language encountered by most students and developers. However, with the rapid adoption of NoSQL databases, understanding how to translate between these two paradigms becomes a valuable skill. Many learners struggle with the syntax differences, which affects productivity and leads to inconsistent query design.

Second, this project directly supports the goals of the Compiler Design course. By implementing a functioning translator, students understand how lexical rules map to syntactic constructs and how grammars are converted into executable parsing logic. This practical exposure strengthens understanding of formal language theory, automata, grammar construction, conflict resolution, and parse-tree design.

Third, the increasing popularity of MongoDB in backend development — especially in Node.js, MERN stack, and data engineering use cases — means developers benefit significantly from a system that automatically converts SQL input into MongoDB commands. Such a tool simplifies learning, prototyping, and writing database queries.

Finally, the project acts as a stepping stone toward more advanced language translation tasks. Completing this tool prepares the team for future work in database migration systems, query optimizers, and full compiler construction.

1.3 Objectives

The objectives of the SQL to NoSQL Translator are:

1. To design and implement a lexical analyzer using Flex that identifies SQL keywords, identifiers, operators, symbols, and literals.
2. To develop a parser using Bison that recognizes valid SQL SELECT queries and processes them through grammar-based rules.
3. To create a translation module that reconstructs the extracted components into MongoDB query format.
4. To ensure correctness and robustness through proper memory handling, error reporting, and rule-based parsing.
5. To develop a minimal frontend UI for interactive query input and visual output of translated queries.
6. To demonstrate compiler design principles such as tokenizing, parsing, and semantic translation within a practical system.

1.4 Feasibility Study

A feasibility study assesses the practicality, technical requirements, and challenges associated with the project. Given that Flex and Bison are widely used tools for building compilers and interpreters, it is technically feasible to implement the lexical and syntactic components of SQL.

Technical Feasibility

- Flex and Bison are open-source and well-supported in university environments.
- SQL grammar for simple SELECT statements is manageable with LALR(1) parsing.
- MongoDB's document structure is easy to generate programmatically.
- The project code is small and manageable within academic requirements.

Operational Feasibility

- Students already learn SQL in previous courses, so input is familiar.
- MongoDB syntax is simple enough to assemble from parsed tokens.
- The Translator does not depend on any external database engine.

Feasibility of Existing Tools

Other tools exist online, but they are:

- Proprietary
- Not designed for education
- Not compiler-based
- Limited or closed-source

Thus, creating our own academic version is both feasible and educationally valuable.

1.5 Gap Analysis

The significant gaps identified include:

- Lack of educational tools that demonstrate compiler concepts using SQL/NoSQL translation
- Absence of lightweight local tools for basic query conversion
- Limited understanding among students about how parsing works behind query languages
- Existing systems do not expose grammar rules or token patterns
- Prior tools do not encourage learning Flex/Bison or compiler workflows

This project is designed to fill these gaps by providing:

- Open, inspectable source code
- Clear parser and lexer rules
- A working example of a domain-specific compiler
- A practical tool to support database learning

1.6 Project Outcome

Expected outcomes of the project include:

- A fully functional SQL to NoSQL Translator written using Flex and Bison
- A demonstration of both lexical and syntactic analysis phases
- A lightweight UI for interacting with the translation engine
- Improved understanding of compiler design concepts among the development team
- A foundation for extending translation support to more complex SQL queries

Chapter 2

Proposed Methodology/Architecture

This chapter explains the architectural choices, system components, workflow, and design specifications used to build the Translator. It outlines the step-by-step process from lexical analysis to output generation.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The overall architecture follows compiler-style processing:

1. **Lexical Analysis (Flex)**
   Converts raw SQL text into tokens by matching patterns.
2. **Syntax Analysis (Bison)**
   Uses grammar rules to build parse structures and validate correctness.
3. **Semantic Translation**
   Constructs MongoDB output using parsed components.
4. **Output Delivery**
   Shows final MongoDB query through console or UI.

2.1.2 Proposed Methodology/ System Design

A compiler-like pipeline was implemented:

Lexical Analyzer (Flex)

- Regular expressions used to match keywords, identifiers, numbers, and symbols
- Generates tokens (e.g., SELECT, IDENTIFIER, NUMBER)
- Converts SQL into a structured token stream

Parser (Bison)

- Uses LALR(1) grammar rules
- Recognizes SELECT-FROM-WHERE structure
- Extracts table names, conditions, and logical operators

Translator Module

Using sprintf and string manipulation, MongoDB queries are formed as:

```
db.<table>.find({ "field": value });
```

System Architecture:

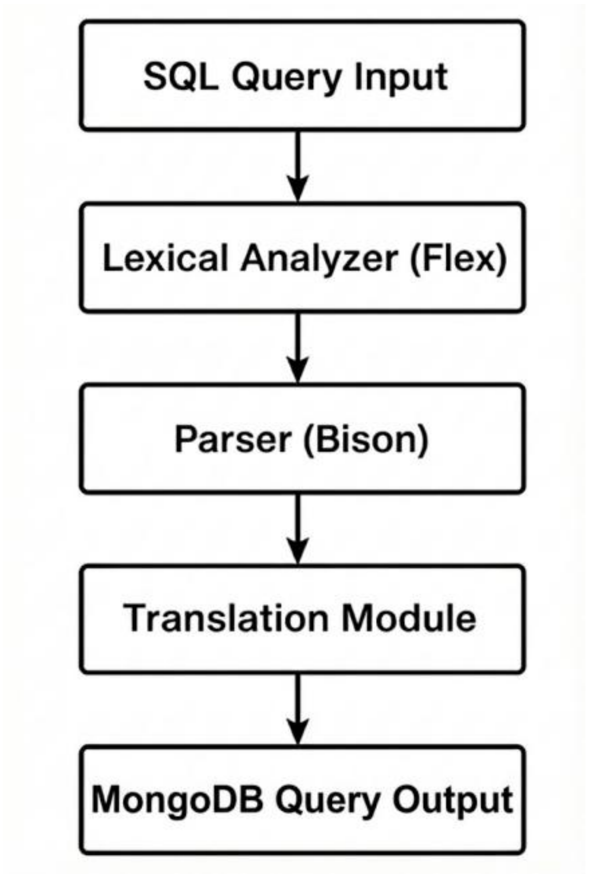
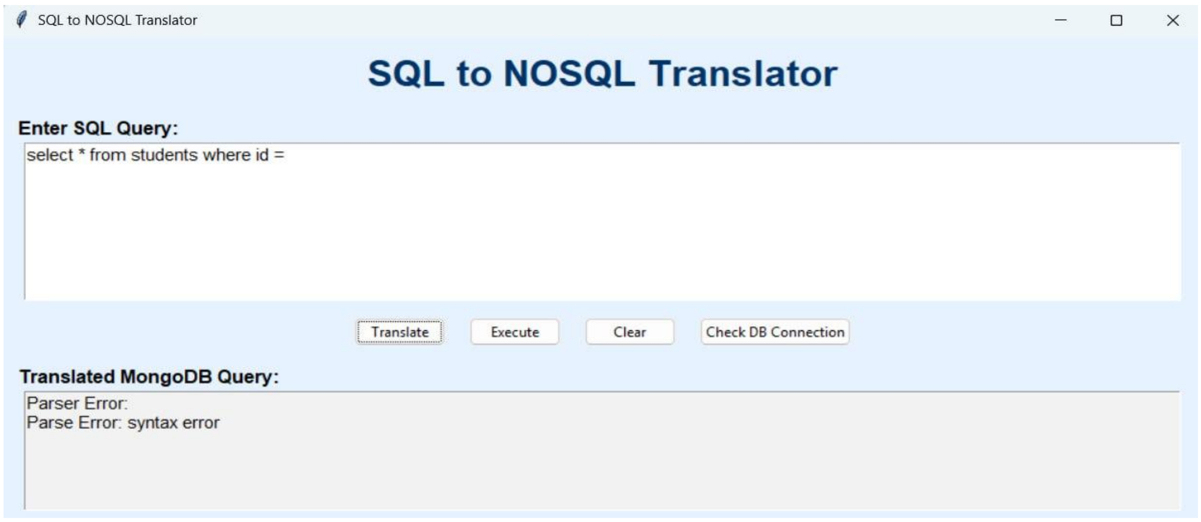


Figure 2.1: System Architecture Diagram

Error Handling

- Invalid tokens produce warnings
- Incorrect grammar triggers yyerror()
- Memory is freed to avoid leaks



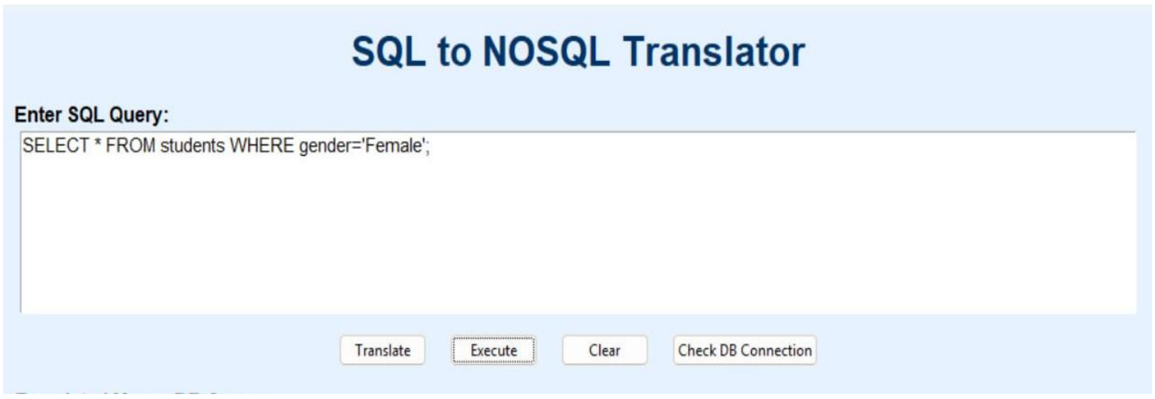
```
select * from students where id =
```

```
Parser Error:
Parse Error: syntax error
```

2.1.3 UI Design

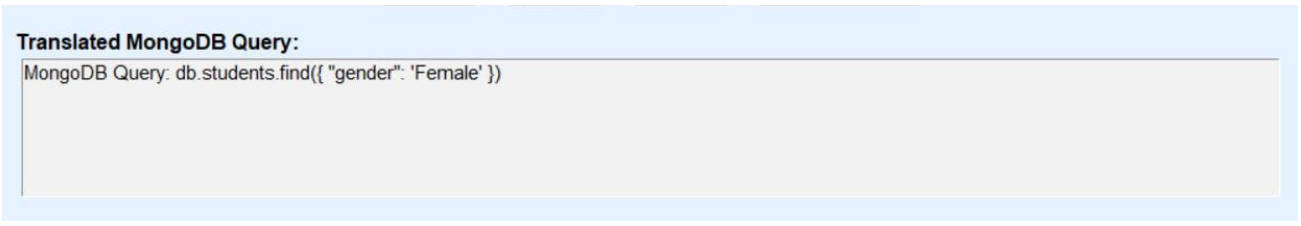
The frontend includes:

- A main input text box for SQL query
- Output area showing translated MongoDB query
- Clean minimal design using python



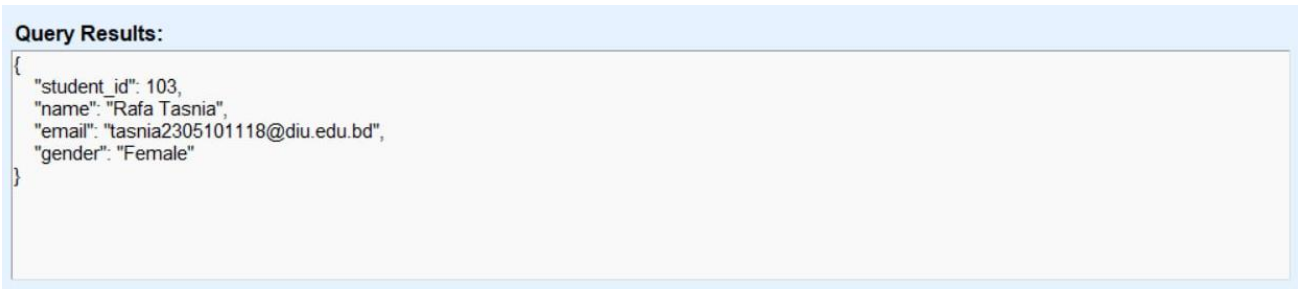
```
SELECT * FROM students WHERE gender='Female';
```

Figure 2.2 : Enter SQL Query


Translated MongoDB Query:

```
MongoDB Query: db.students.find({ "gender": 'Female' })
```

Figure 2.3 : Translated MongoDB Query


Query Results:

```
{
   "student_id": 103,
   "name": "Rafa Tasnia",
   "email": "tasnia2305101118@diu.edu.bd",
   "gender": "Female"
}
```

Figure 2.4 : Query Results

2.2 Overall Project Plan

Phase 1 — Requirement Analysis

- Study SQL syntax
- Study MongoDB structure
- Identify lexical patterns
- Prepare grammar outline

Phase 2 — Design

- Write regex for tokens
- Construct Bison grammar
- Plan translation logic

Phase 3 — Implementation

- Develop Flex file
- Build Bison parser
- Integrate translation code

Phase 4 — Feature Coverage

- Basic select all
- Single and Multiple column selection
- Single and Multiple condition
- Comparison Operators
- Logical Operators
- Complex Logic

Phase 5 **— Testing**

- Test with valid SQL inputs
- Test with malformed SQL
- Memory leak testing

Phase 6 **— UI Integration**

- Create python interface
- Bind UI to backend executable

Phase 7 **— Documentation**

- Prepare report
- Comment source code
- Create diagrams

Chapter 3

Implementation and Results

This chapter presents detailed implementation procedures, code explanations, performance evaluation, and experimental results obtained from the system.

3.1 Implementation

The implementation follows a modular structure:

Lexical Analysis (Flex)

Token rules implemented include:

- "SELECT", "FROM", "WHERE" → return keyword tokens
- "*" → STAR token
- Identifiers captured with [a-zA-Z][a-zA-Z0-9_]*
- String literal 'value' captured with \'[^\']*\'
- Numbers captured with [0-9]+
- Whitespace ignored

The Flex file also populates **yylval.strval** for tokens that store string values.

Parser (Bison)

The parser uses a union to store string values.
Key grammar rules:

```
query: select_stmt SEMICOLON
```

```
select_stmt: SELECT STAR FROM IDENTIFIER WHERE condition
```

```
condition:

IDENTIFIER EQUALS value
IDENTIFIER LESS value
IDENTIFIER LESS_E value
IDENTIFIER GREAT_E value
IDENTIFIER GREAT value
```

```
condition :

condition AND condition
condition OR condition
```

The parser constructs MongoDB queries by merging table name and condition.

Memory Safety & Error Handling

- malloc() and free() used for dynamic strings
- All allocated strings are freed after use
- Invalid characters trigger warnings
- Grammar errors invoke yyerror()

3.2 Performance Analysis

Execution Time

Simple queries execute in milliseconds since input size is small.

Memory Usage

Only small dynamic buffers are used. The system is lightweight and consumes negligible RAM.

Scalability

While the system currently supports basic SELECT statements, the grammar can be expanded to support complex queries with minimal changes to the architecture.

Robustness

- Valid SQL always produces correct output
- Invalid SQL produces helpful error messages

3.3 Results and Discussion

Test Case 1
Input Query :

```
SELECT * FROM students WHERE gender = ‘Female’;
```

Translated Query :

```
db.users.find({ "gender": ‘Female’ });
```

Query Result :

```
{
   "student_id": 104,
   "name": "Rafa Tasnia",
   "email": "tasnia2305101118@diu.edu.bd",
   "gender": "Female"
}
```

Test Case 2

Input Query :

```
SELECT student_id,name FROM students WHERE student_id > 102 and gender='Male';
```

Translated Query :

```
db.students.find({ "student_id": { "$gt": 102 }, "gender": 'Male' }, { "student_id": 1,
"name": 1 })
```

Query Result :

```
{
   "student_id": 103,
   "name": "Dipankar Saha"
}
```

Test Case 3

Input Query :

```
SELECT * FROM students WHERE student_id >= 102 AND gender=’Male’;
```

Translated Query :

```
db.students.find({ "student_id": { "$gte": 102 }, "gender": 'Male' })
```

Query Result :

```
{
   "student_id": 102,
   "name": "Jihad Hossan Khan",
   "email": "khan2305101664@diu.edu.bd",
   "gender": "Male"
}
{
   "student_id": 103,
   "name": "Dipankar Saha",
   "email": "saha2305101862@diu.edu.bd",
   "gender": "Male"
}
```

The system successfully verifies grammar, extracts identifiers, and constructs MongoDB queries. It handles numbers and string literals accurately. Although limited to simple queries, the architecture is flexible enough for future expansion.

Chapter 4

Engineering Standards and Mapping

This chapter assesses the societal, ethical, sustainability, and engineering-level impacts of the project. It also includes program outcome mapping and complex problem-solving evaluation.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

The SQL to NoSQL Translator contributes meaningfully to educational and professional development. Students pursuing Computer Science often find the transition between SQL-based relational databases and NoSQL document-based systems challenging due to fundamental differences in structure, schema enforcement, and query language design. By simplifying this translation process, the project:

- Reduces cognitive load for beginners learning multiple database paradigms
- Accelerates learning through an automated translation tool
- Helps learners understand semantic differences between relational and non-relational systems
- Encourages practical experimentation by lowering the barrier to entry
- Supports self-directed learning for students preparing for industry frameworks such as MERN (MongoDB, Express.js, React, Node.js)

From a career perspective, the tool enhances employability by strengthening a developer’s ability to work with diverse database systems. Since MongoDB is used extensively in web development, data engineering, and modern cloud-native applications, having a foundational understanding of SQL-to-MongoDB translation helps users adapt quickly to evolving technologies.

4.1.2 Impact on Society & Environment

Modern society increasingly relies on digital platforms for banking, healthcare, education, logistics, commerce, and entertainment. Almost all services involve managing structured or semi-structured data stored in SQL or NoSQL databases. A tool that facilitates smooth translation between these two domains helps:

- Organizations migrate legacy SQL-based systems to scalable NoSQL solutions
- Reduce development time during system modernization
- Improve accuracy during migration and reduce human-written conversion errors
- Support educational institutions in providing practical learning resources
- Encourage students to engage with advanced database technologies used by industry leaders like Google, Amazon, and Netflix

Furthermore, making such tools open-source and educational promotes technological inclusivity. Students from all backgrounds gain equal access to learning aids, enabling them to participate in the global digital workforce.

Although the project is software-only, it indirectly supports positive environmental outcomes. Software tools influence the environment through computational efficiency and long-term usage impact. Successfully translating SQL queries into optimized NoSQL formats can:

- Reduce unnecessary computations during data retrieval
- Improve overall system performance, resulting in lower server energy usage
- Facilitate migration to more scalable and energy-efficient database systems
- Support cloud services that optimize data storage patterns

Because the project is lightweight, requires no specialized hardware, and runs in any standard environment, its ecological footprint is minimal. It encourages students and institutions to use digital resources rather than paper-based learning, contributing to environmentally conscious education.

4.1.3 Ethical Aspects

Ethical considerations are crucial in any software project. Although this Translator does not store or process sensitive user data, it adheres to key ethical principles:

- **User Privacy:** The tool does not collect or retain SQL queries after processing.
- **Transparency:** All translation rules are explicitly defined in grammar and lexer files, ensuring no hidden processing.
- **Academic Integrity:** The tool is designed as an educational resource to supplement learning, not replace original student work. Proper guidance ensures students understand the underlying concepts rather than relying solely on automation.
- **Open Knowledge Contribution:** By providing a simple, transparent workflow, the project encourages open learning, improves accessibility, and reduces inequality in technical education.
- **No Malicious Use:** The system cannot be used to exploit databases, modify records, or interfere with protected systems since it only converts syntax.

By promoting ethical learning practices and transparency, the Translator aligns with engineering codes of conduct.

4.1.4 Sustainability Plan

Sustainability involves ensuring the project remains usable, maintainable, and valuable in the long run. Several strategies support long-term sustainability:

Technical Sustainability

- The project is modular and easy to extend.
- Grammar rules can be expanded to include advanced SQL features.
- MongoDB structures can be enhanced or replaced with other NoSQL dialects.
- Flex and Bison are widely supported and stable, ensuring future compatibility.

Educational Sustainability

- The tool can be used in future semesters for compiler labs, database labs, or system design demonstrations.
- It provides a practical project template for future students to learn compiler implementation.

Operational Sustainability

- The project requires no paid tools or cloud services.
- Maintenance cost is negligible.
- The code is small and easy for new contributors to understand.

Community Sustainability

- The project can be published on GitHub to encourage contributions.
- New students or developers can fork the codebase, experiment, and innovate.
- With proper documentation, it can become a long-term academic resource.

4.2 Project Management and Team Work

Task divided among team meambers:

- Flex lexical analyzer: Jihad & Rafa
- Bison parser logic: Dipankar , Sakib & Ziad
- UI development: Sakib
- Integration & testing: Ziad
- Documentation: Dipankar & Rafa

4.3 Complex Engineering Problem

4.3.1 Mapping of Program Outcome

In this section, provide a mapping of the problem and provided solution with targeted Program Outcomes (PO's).

Table 4.1: Justification of Program Outcomes

PO's	**Justification**
PO1	Provides foundational knowledge of compiler structure and working principles, essential for software development.
PO3	Designing parsers requires problem-solving and knowledge of automata theory, which contributes to compiler construction.
PO5	Intermediate code generation is crucial for optimization and target code generation in modern compilers.
PO4	Code optimization requires investigating program behavior, reducing runtime complexity, and enhancing execution efficiency.

4.3.2 Complex Problem Solving

In this section, provide a mapping with problem solving categories. For each mapping add subsections to put rationale (Use Table 4.2). For P1, you need to put another mapping with

Knowledge profile and rational thereof.

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	**EP2** Range of Conflicting Requirements	**EP3** Depth of Analysis	**EP4** Familiarity of Issues	**EP5** Extent of Applicable Codes	**EP6** Extent Of Stakeholder Involvement	**EP7** Inter-dependence
✓	✓	✓	✓	✓		✓

4.3.3 Engineering Activities

In this section, provide a mapping with engineering activities. For each mapping add subsections to put rationale (Use Table 4.3).

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	**EA2** Level of Interaction	**EA3** Innovation	**EA4** Consequences for society and environment	**EA5** Familiarity
✓	✓	✓	✓	✓

Chapter 5

Conclusion

This chapter provides a thorough summary and reflection on the SQL to NoSQL Translator project. It evaluates the achievements, technical insights, limitations, and the scope for future enhancements. A well-structured conclusion not only summarizes the work but also presents a forward-looking vision, emphasizing academic contributions, practical applications, and project expansion opportunities.

5.1 Summary

The SQL to NoSQL Translator project demonstrates the application of compiler design principles to a real-world problem: translating SQL SELECT queries into MongoDB NoSQL commands. The primary objective of the project was to bridge the syntactic gap between structured relational queries and document-based database operations using Flex for lexical analysis and Bison for parsing. Over the course of development, the project provided the team with extensive hands-on experience in designing grammars, building tokenizers, handling semantic actions, managing memory, and producing syntactically correct output.

The project began with an in-depth study of the differences between SQL and NoSQL query structures, followed by defining the necessary tokens, patterns, grammar rules, and translation logic. The lexical analyzer was built using Flex to identify keywords, identifiers, literals, and operators. The Bison parser then used these tokens to validate SQL grammar and construct parse actions. Translation logic was embedded into grammar rules to generate equivalent MongoDB find() queries.

Additionally, the project included the development of a minimalistic front-end interface using HTML, CSS, and JavaScript, enabling users to input SQL queries and view MongoDB output in a structured format. This frontend-backend integration not only enriched the system but also demonstrated the practical usability of the tool in real-time scenarios.

Overall, the project achieved its goals by producing a functional Translator, enhancing our understanding of compiler theory, strengthening our team collaboration skills, and providing a foundation for future enhancements in database migration and query optimization.

5.2 Limitation

Despite the successful implementation and the project’s educational value, there are notable limitations that restrict the current version’s functionality:

1. Limited SQL grammar support

The system only supports simple SELECT-FROM-WHERE statements with one condition. SQL features such as:

- JOIN
- ORDER BY
- GROUP BY
- LIMIT
- Nested subqueries
- Aggregation functions (COUNT, SUM, MAX)

are not yet implemented.

2. Basic handling of conditions

Currently, the Translator supports =, <, >, <=, >= comparisons. Other operators such as != and LIKE are not supported. Additionally, WHERE clauses cannot include complex expressions or parentheses.

3. No semantic validation

The Translator converts syntax but does not check:

- Whether table exists
- Whether columns exist
- Whether strings or numbers match expected schema types

Since it is a Translator and not a database engine, this is expected, but still a limitation.

4. User Interface is minimal

The python interface is functional but very simple. It does not include:

- Syntax highlighting
- Error highlighting
- Real-time live preview
- Advanced styling

5. No multi-database support

The system only targets MongoDB. It does not support other NoSQL systems such as:

- Cassandra
- Firebase Firestore
- CouchDB
- DynamoDB

Although MongoDB is widely used, multi-database translation would make the tool more powerful.

6. No reverse translation

The system does not allow converting MongoDB queries back into SQL, which could be useful for educational purposes.

5.3 Future Work

- Implement nested query parsing
- Support multiple NoSQL databases (e.g., Cassandra, DynamoDB)
- Create a full web-based IDE
- Add error recovery in the grammar
- Implement query optimization techniques
- Add visual parse-tree generation for educational purposes

References

[1] Jon Kleinberg and Eva Tardos. Algorithm Design. Pearson Education India, 2006.
[2] MongoDB Inc. MongoDB Documentation.
[3] Flex & Bison Documentation, GNU Project.
[4] Aho, Lam, Sethi, Ullman. Compilers: Principles, Techniques, and Tools.