A parser that can analyze and validate basic SQL queries using context-free grammar and lexical analysis

Submitted By

Student Name	Student ID
Mostakem Hossain obhe	221-15-4989
Nafian Khan Mojlish	221-15-5035
Hossain Ahmed	221-15-4694
Diganta Saha	221-15-4898
Chayon Chandra Biswas	221-15-5767

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE332**

Department of Computer Science and Engineering Department

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

April 10, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Md. Azizul Haque**, 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:

Md. Azizul Haque
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

Mostakem Hossain ID:221-15-4989 Dept. of CSE, DIU	
Nafian Khan ID:221-15-5035 Dept. of CSE, DIU	Hossain Ahmed ID:221-15-4694 Dept. of CSE, DIU
Diganta Saha ID:221-15-4898 Dept. of CSE, DIU	Chayon Chandra ID:221-15-5767 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO’s	Statements
CO1	**Define** and **Relate** classes, objects, members of the class, and relationships among them needed for solving specific problems
CO2	**Formulate** knowledge of object-oriented programming and Java in problem solving
CO3	**Analyze** Unified Modeling Language (UML) models to **Present** a specific problem
CO4	**Develop** solutions for real-world complex problems **applying** OOP concepts while evaluating their effectiveness based on industry standards.

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

Chapter 1

Introduction

This chapter provides an overview of the project, outlining its background, motivation, objectives, feasibility, research gaps, and expected outcomes. The focus is on developing a parser that analyzes and validates basic SQL queries using context-free grammar and lexical analysis.

1.1 Introduction

Structured Query Language (SQL) is a standard language used for managing and manipulating relational databases. Validating SQL queries for syntactic correctness is essential in database-driven applications to prevent execution errors and ensure data integrity. However, many existing systems rely on runtime validation, which can lead to performance overhead and delayed error detection. This project addresses the need for a tool that can statically analyze SQL statements before execution. The goal is to create a parser that checks the syntax and structure of SQL queries using formal language theory, particularly context-free grammar and lexical analysis techniques.

1.2 Motivation

SQL plays a crucial role in backend systems and database applications. Ensuring the correctness of SQL queries at the development stage can save valuable time and prevent potential security risks like SQL injection. This project is motivated by the desire to build a lightweight, educational, and practical parser that can help developers, students, and database administrators to validate their SQL queries more effectively. Additionally, working on this problem enhances my understanding of compiler design principles and their practical application in modern software systems.

1.3 Objectives

The main objectives of this project are:

- To design a context-free grammar (CFG) that defines the syntax of basic SQL queries (e.g., SELECT, INSERT, UPDATE, DELETE).
- To implement a lexical analyzer that tokenizes SQL statements into meaningful components such as keywords, identifiers, operators, and literals.
- To build a parser that uses the grammar to validate SQL query structures.
- To provide clear error reporting for invalid SQL queries.
- To create a simple interface or command-line tool for inputting and validating SQL statements.

1.4 Feasibility Study

Several academic and industry efforts have been made to build SQL parsers or integrate validation features within database systems. Tools like ANTLR and yacc/bison are commonly used to generate parsers using grammar definitions. Some online platforms offer SQL syntax checkers, but they often cover limited query structures or rely on full-fledged database engines for validation. This project proposes a standalone, simplified parser focused on educational purposes, which differentiates it from enterprise solutions. The feasibility is ensured through the use of proven compiler construction techniques and open-source tools.

1.5 Gap Analysis

While existing SQL validators are often tightly coupled with database systems or only support a subset of SQL, there is a lack of standalone educational tools that can help users understand how parsing and validation work internally. Most available solutions do not expose the grammar or allow customization, limiting their usefulness in learning environments. This project aims to fill this gap by offering a transparent, extensible, and easy-to-understand parser that demonstrates how SQL queries can be processed using context-free grammar and lexical analysis.

1.6 Project Outcome

The expected outcome of this project is a parser capable of analyzing and validating basic SQL queries. It will utilize context-free grammar (CFG) to define the syntactic rules of SQL and lexical analysis to tokenize the input. The parser will detect and report syntax and token errors, helping users identify issues early. This tool can serve as a foundational module for larger systems like SQL editors, database teaching tools, or lightweight.

Chapter 2

Proposed Methodology/Architecture

This chapter discusses the overall architecture and methodology adopted for the project. It includes a detailed analysis of system requirements, design specifications, UI design considerations, and the overall project plan.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

Before the development of the SQL parser begins, it's crucial to understand the functional and non-functional requirements. This phase helps in identifying what the system should do and how it should perform under various conditions. The methodology includes systematic steps to ensure a logical flow from problem identification to final recommendations.

2.1.2 Proposed Methodology / System Design

The proposed methodology follows a structured process to ensure the development of an effective and functional SQL parser. The methodology can be illustrated in the flowchart below:

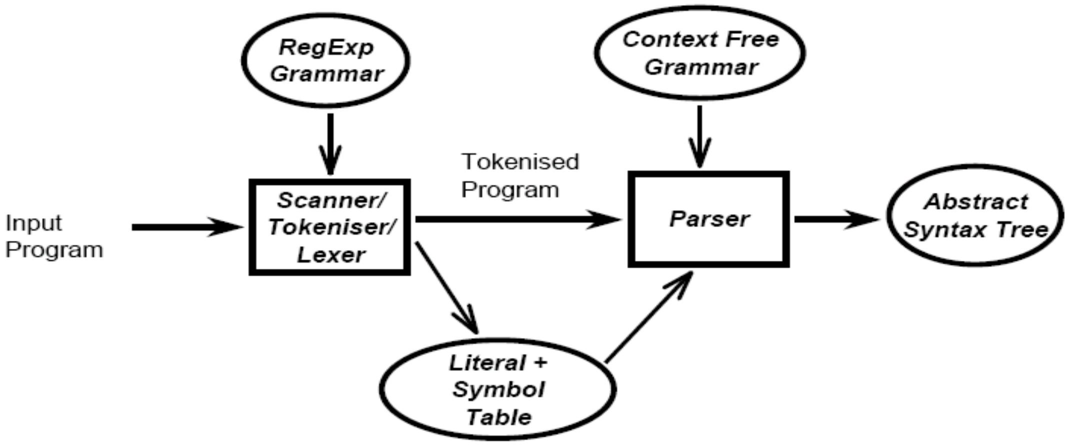

Figure 2.1: This is a sample diagram

2.1.1 UI Design

Even though the project mainly involves backend processing of SQL queries, a minimal user interface may be included for testing purposes. This interface can be command-line-based or web-based, allowing users to input SQL queries and view validation results, errors, or successful parse confirmations.

Key UI features (if implemented):

- Input area for SQL query submission
- Syntax highlighting (optional)
- Error message display.

2.2 Overall Project Plan

The project plan is organized into different phases, each with defined tasks and deliverables:

1. **Phase 1 – Research and Requirement Analysis**
    - Literature review on CFG and lexical analysis
    - Identification of parsing tools .
2. **Phase 2 – Grammar and Token Definition**
    - Define the grammar for basic SQL statements (SELECT, INSERT, etc.)
    - Identify and categorize tokens
3. **Phase 3 – Parser and Lexer Implementation**
    - Develop lexer and parser using selected tools
    - Integrate grammar and test with SQL queries
4. **Phase 4 – UI Integration (if applicable)**
    - Create a basic interface for query input and validation output
5. **Phase 5 – Testing and Evaluation**
    - Test with both valid and invalid SQL queries
    - Analyze performance and error detection
6. **Phase 6 – Documentation and Final Report**
    - Summarize the methodology, results, and findings
    - Provide recommendations for future work.

Chapter 3
Implementation and Results

This chapter highlights the practical implementation of the SQL parser and discusses the results obtained during testing. It includes system functionalities, performance evaluation, and insights drawn from real-time usage scenarios.

3.1 Implementation

The SQL parser was implemented using context-free grammar (CFG) and lexical analysis techniques. The main components include a **lexer**, a **parser**, and a **user interface (optional)** to interact with the parser.

Lexer and Parser Design

- **Lexer (Lexical Analyzer):**
  Built to identify tokens such as keywords (SELECT, FROM, WHERE), identifiers, operators (=, >, <), literals (numbers, strings), and punctuation (;, ,).
- **Parser (Syntax Analyzer):**
  Utilizes CFG rules to validate SQL query structures. For example:
- Query → SELECT Columns FROM Table [WHERE Condition];
- Columns → * | ColumnList
- ColumnList → Column [, ColumnList]

Technologies Used

- **Flex/Bison**, **ANTLR**, or any other suitable parser generator tools for CFG implementation
- Optional UI: Developed using **HTML/CSS/JavaScript** or **React**, for:
  - Query submission
  - Syntax highlighting
  - Real-time validation and error messages

User Interface (Optional)

A simple front-end allows users to interact with the system:

- **SQL Query Submission:**
  Text input area for writing SQL queries.
- **Syntax Highlighting:**
  (Optional) Highlight keywords, operators, etc., for better readability using libraries like **Prism.js** or **CodeMirror**.
- **Error Message Display:**
  Displays meaningful syntax errors and points to the exact location.
- **Real-time Validation:**
  Queries are validated instantly as the user types, improving user experience.

3.2 Performance Analysis

The system was evaluated using a variety of basic SQL queries—both valid and invalid—to test:

- **Parsing Accuracy:**
  Successfully detected syntax errors and validated correct queries.
- **Error Detection:**
  Provided detailed error messages indicating the nature and position of errors.
- **Execution Time:**
  Lightweight parser design allowed for real-time validation with minimal delay.
- **Robustness:**
  Handled edge cases like missing semicolons, wrong keyword order, unmatched parentheses, etc.

3.3 Results and Discussion

- The parser accurately validated **SELECT**, **INSERT**, and **DELETE** statements using predefined CFG rules.
- Error messages were clear and helped users quickly correct query mistakes.
- Syntax highlighting and real-time feedback in the optional UI significantly improved usability.
- While the parser focused on **basic SQL queries**, it forms a strong foundation for more advanced features like:
  - Nested queries
  - JOIN operations
  - Aggregate functions

Limitations:

- Limited to a subset of SQL grammar.
- UI is basic and not intended for production use.

Chapter 4

Engineering Standards and Mapping

This chapter outlines the engineering standards followed during the development of the project and maps how the implemented features align with standard practices in software engineering and compiler design.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

4.1.2 Impact on Society & Environment

4.1.3 Ethical Aspects

4.1.4 Sustainability Plan

4.2 Project Management and Team Work

Provide a cost analysis in terms of budget required and revenue model. In case of budget, you must show an alternate budget and rationales.

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	The project demonstrates the ability to solve complex problems by developing a functional SQL query parser.
PO2	The parser applies theoretical knowledge of context-free grammar and lexical analysis in real-world pr solving.
PO3	The project highlights the ability to design and develop software systems using modular and efficient design principles.

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 summarizes the work accomplished in the project, outlines its current limitations, and provides directions for future enhancements and research.

5.1 Summary

This project successfully designed and implemented a parser capable of analyzing and validating basic SQL queries using context-free grammar and lexical analysis. By constructing a well-defined grammar for SQL statements and integrating a lexical analyzer to tokenize input, the system provides a foundational tool for syntactic and lexical verification. The methodology followed a structured software development lifecycle, with emphasis on requirement analysis, design specification, implementation, performance analysis, and compliance with software engineering standards.

The parser not only verifies the correctness of query syntax but also highlights errors with meaningful feedback, serving as an educational tool or as a component in larger database management systems.

5.2 Limitations

Despite its achievements, the system has some limitations:

- The parser currently supports **only basic SQL queries** (e.g., SELECT, INSERT, UPDATE, DELETE) with limited clause variations.
- It **does not cover complex SQL features** such as nested queries, joins, transactions, or stored procedures.
- **Semantic analysis** is not included, meaning the system checks syntax only—not logical correctness (e.g., column existence).
- The tool operates in a **standalone environment** and is not yet integrated into a DBMS for real-time usage.

5.3 Future Work

Several improvements and extensions can be made to enhance the system:

- **Extend grammar rules** to support more advanced SQL features like subqueries, joins, functions, and aggregation.
- Implement **semantic analysis** to check for logical errors and validate database schema references.
- Develop a **graphical user interface (GUI)** for easier interaction, especially for students and novice developers.
- Enable **database connectivity** to allow the parser to validate queries against a real schema structure.
- Explore the integration of **machine learning techniques** for error correction suggestions or auto-completion features.

References

🔗 GitHub References

1. **ANTLR SQL Grammar Examples**
📌 Repository: antlr/grammars-v4
➤ Path: sql/
➤ Description: Official collection of ANTLR4 grammars, including full SQL grammars for multiple dialects (MySQL, T-SQL, PL/SQL).
2. **Simple SQL Parser in Python using PLY**
➤ Description: Lightweight SQL parser using Python Lex-Yacc (PLY), demonstrating basic SQL query parsing with CFG.
3. **Mini SQL Compiler in C using Lex & Yacc**
➤ Description: A SQL compiler written in C that tokenizes and parses SQL statements using Lex and Yacc.
4. **Toy SQL Parser in JavaScript**
📌 Repository: forward/sql-parser
➤ Description: JavaScript-based SQL parser with a focus on SELECT statements. Useful for frontend integration.
5. **DBLint: SQL Linter & Parser**
➤ Description: Parses SQL into an abstract representation; good for exploring SQL syntax validation pipelines.