আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম
الجامعة الإسلامية العالمية شيتاغونغ
International Islamic University Chittagong

Department of Computer Science and Engineering

Compiler Project Report

Course Code: CSE-3528

Course Title: Compiler Lab

Project Title: Mini SQL Parser and Validator

Submitted To: Mr. Rayhanuzzaman
Lecturer, Department of CSE

Submitted By:

ID No.	Name
C231053	Mohammed Sami Hasan
C231068	Hasnain Kabir Nabil
C231070	Md. Nasim UL Haque

Semester: 5'th

Section: 5BM

Submission Date:

Remarks:

Project title: Mini SQL Parser and Validator

Introduction:

Structured Query Language (SQL) is the standard language used for managing and manipulating relational databases. Understanding how SQL statements are processed internally—through parsing, validation, and execution—is fundamental to both database design and compiler construction. This project, titled "Mini SQL Parser and Validator", aims to simulate the initial phases of SQL statement processing, offering insights into how database systems interpret and validate user queries.

Objective of the Project:

The objective of this project is to design and implement a simplified SQL compiler that can:

- Recognize and tokenize SQL input strings.
- Parse SQL statements and construct their Abstract Syntax Trees (AST).
- Maintain a symbol table for semantic understanding.
- Provide a detailed output of each stage for analysis and debugging.

Tools and Technologies Used:

- **Programming Language**: C++
- **IDE**: VS Code editor
- **Platform**: Windows
- **Optional Libraries**: sys (for input/output)

Project Features:

1 Tokenization

The tokenizer (Lexer) breaks down raw SQL input into structured tokens.

- Recognizes:
    - SQL **keywords**: SELECT, FROM, WHERE, INSERT, etc.
    - **Identifiers**: Table names, column names
    - **Literals**: Numbers, strings, Boolean values
    - **Operators**: =, <, >, <=, >=, !=
    - **Punctuation**: Comma, semicolon, parentheses
- Outputs a list of tokens with their types.

2 Parsing

The parser converts tokens into structured **Abstract Syntax Trees (AST)**.

- Handles statement types:
    - SELECT
    - INSERT INTO
    - UPDATE
    - DELETE
- Parses expressions in WHERE clauses using logical operators (AND, OR).
- Performs basic **syntax error handling** for invalid token sequences.

3 Symbol Table Management

Maintains metadata about tables and columns:

- Add new tables with associated column names

- Verify existence of tables and columns during parsing
- Supports retrieval and printing of table schemas

4 Interactive Input

- Reads multiple SQL statements from the terminal
- Execution continues until an empty line is entered
- Immediate feedback after each statement

5 Output

The compiler prints:

- **Token List**: With type annotations
- **AST**: With structured breakdown of statement components
- **Symbol Table**: In a readable table format

Working Mechanism:

The **Mini SQL Parser and Validator** project follows a structured pipeline that mimics the initial phases of a real SQL compiler. It processes SQL input through the following stages: tokenization, parsing, semantic validation, and output generation. Below is a detailed explanation of each stage:

1. User Input (Interactive Mode)

The system operates in an interactive mode where users can enter one SQL statement at a time. The process continues until an empty line is entered, allowing users to execute multiple statements in a single session.

2. Tokenization (Lexical Analysis)

The **tokenizer** takes the raw SQL statement and converts it into a list of meaningful tokens. It identifies:

- **Keywords**: SELECT, FROM, WHERE INSERT, UPDATE, etc.
- **Identifiers**: Table and column names.
- **Literals**: Numbers, strings, and Boolean values.
- **Operators**: =, <, >, <=, >=, !=
- **Punctuation**: Comma, semicolon, and parentheses ( )

This stage ensures that the SQL string is broken down into understandable parts for the parser.

3. Parsing (Syntax Analysis)

The **parser** analyzes the sequence of tokens and constructs an **Abstract Syntax Tree (AST)** representing the structure of the SQL statement. It determines:

- The type of SQL statement (SELECT, INSERT, UPDATE, DELETE)
- The components involved:
    - Column names
    - Table names
    - Condition expressions (e.g., WHERE clauses using AND/OR)
    - Values for insertion or update

If there is any incorrect syntax (e.g., missing keywords or misplaced tokens), the parser reports an error.

4. Semantic Validation (Symbol Table Check)

To ensure the SQL statement is not only syntactically but also **semantically correct**, the system uses a **symbol table**. This structure stores:

- Names of declared tables
- Corresponding column names

The parser validates:

- If the referenced table exists
- If the specified columns are part of the table

Errors such as "Table not found" or "Column does not exist" are reported at this stage.

5. Output Generation

For each processed SQL statement, the following outputs are generated:

- **Token List**: Displays the type and value of each token.
- **Abstract Syntax Tree (AST)**: Shows the internal structure of the statement.
- **Symbol Table**: Displays the current state of the symbol table with all known tables and their columns.
- **Error Messages**: In case of any syntax or validation failure, a descriptive error message is shown.

6. Loop Execution

After making a statement, the system waits for the next input. This loop continues until the user enters an empty line, allowing the project to handle multiple SQL commands in one session

Input:

```
==========================================
Please provide SQL Statement:
SELECT name, age FROM users WHERE age > 30 AND salary < 5000;
==========================================
```

Output:

```
Tokens:
+-----------------+---------------+
| Token           | Type          |
+-----------------+---------------+
| SELECT          | SELECT        |
| name            | IDENTIFIER    |
| ,               | COMMA         |
| age             | IDENTIFIER    |
| FROM            | FROM          |
| users           | IDENTIFIER    |
| WHERE           | WHERE         |
| age             | IDENTIFIER    |
| >               | GREATER       |
| 30              | NUMBER        |
| AND             | AND           |
| salary          | IDENTIFIER    |
| <               | LESS          |
| 5000            | NUMBER        |
| ;               | SEMICOLON     |
+-----------------+---------------+

Parsed Statement (AST):
------------------------------------------
Type: SELECT
Columns:
  - name
  - age
Table: users
WHERE: age > 30 AND salary < 5000
------------------------------------------

Symbol Table:
------------------------------------------
Table Name: users
Columns:
  - age
  - name
------------------------------------------
```

Error Handling:

- Syntax errors are captured during parsing
- Semantic errors such as "table not found" or "column not found" are flagged using the symbol table
- Each error displays a meaningful message

Limitations:

- Only supports a subset of SQL (SELECT, INSERT, UPDATE, DELETE)
- No query execution or actual database connectivity
- Basic error messages; no recovery
- Does not support JOINs, GROUP BY, or nested queries

Future Improvements:

- Support for more SQL statements and expressions (e.g., CREATE TABLE, DROP TABLE)
- Enhanced error messages with line/column numbers
- GUI interface for interactive input/output
- Query optimization phase (as part of future compiler enhancements)
- Integration with a database engine or in-memory DB

Conclusion:

In this project, a Mini SQL Parser and Validator was developed to analyze and validate basic SQL queries. The system successfully tokenizes SQL statements, checks their syntax structure, and identifies errors in a simplified yet effective manner. This project enhanced understanding of parsing techniques and compiler principles while demonstrating how theoretical concepts can be applied in practical software tools.

References

- Aho, Lam, Sethi, Ullman – *Compilers: Principles, Techniques, and Tools*
- SQL Documentation: https://www.w3schools.com/sql/
- Blackbox AI – Used for debugging and code suggestions
- ChatGPT by OpenAI – Used to understand parsing concepts and troubleshooting issues

Source Code: