Mini SQL Compiler – Full Technical Report

This report documents the complete implementation of the Mini SQL Compiler, covering the three phases: Lexical Analysis, Syntax Analysis, and Semantic Analysis, along with overall integration and data flow.

1. Introduction

The Mini SQL Compiler validates SQL queries through a standard compiler front-end pipeline. Phase 01 performs tokenization using DFA-based recognition. Phase 02 validates grammar and builds a parse tree using a recursive-descent parser. Phase 03 enforces semantic correctness using a symbol table and type checks. The supported SQL subset includes CREATE TABLE, INSERT, SELECT, UPDATE, and DELETE (with optional WHERE clauses).

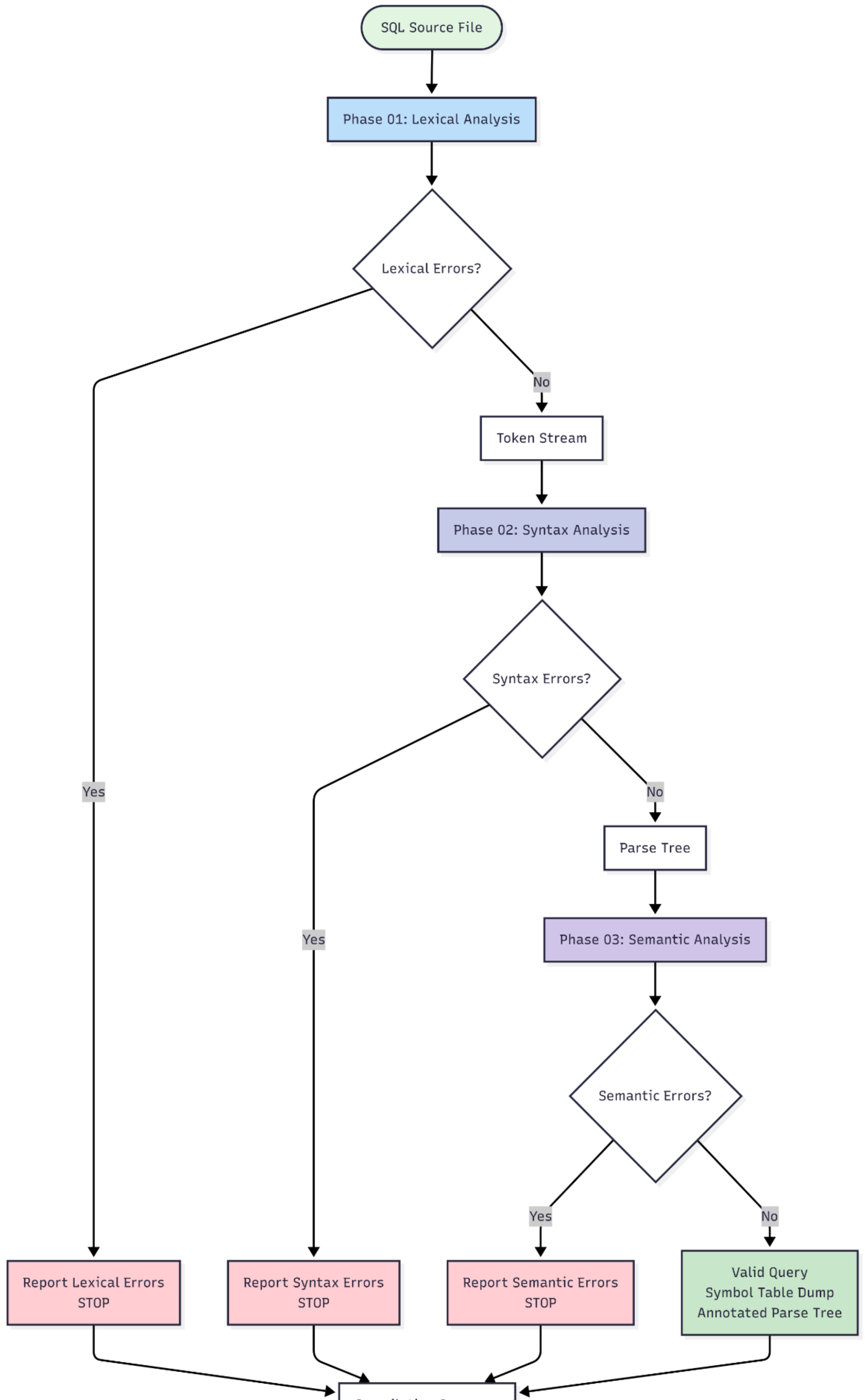

SQL Source File
Phase 01: Lexical Analysis
Lexical Errors?
Yes
No
Token Stream
Phase 02: Syntax Analysis
Syntax Errors?
Yes
No
Parse Tree
Phase 03: Semantic Analysis
Semantic Errors?
Yes
No
Report Lexical Errors STOP
Report Syntax Errors STOP
Report Semantic Errors STOP
Valid Query Symbol Table Dump Annotated Parse Tree

Figure 1: Overall compilation pipeline (phases and outputs).

2. Phase 01: Lexical Analysis

Lexical analysis scans the raw SQL source character-by-character and produces a stream of tokens. The lexer classifies characters (letter, digit, operator, delimiter, quote, whitespace/comment) and applies DFAs to recognize valid lexemes.

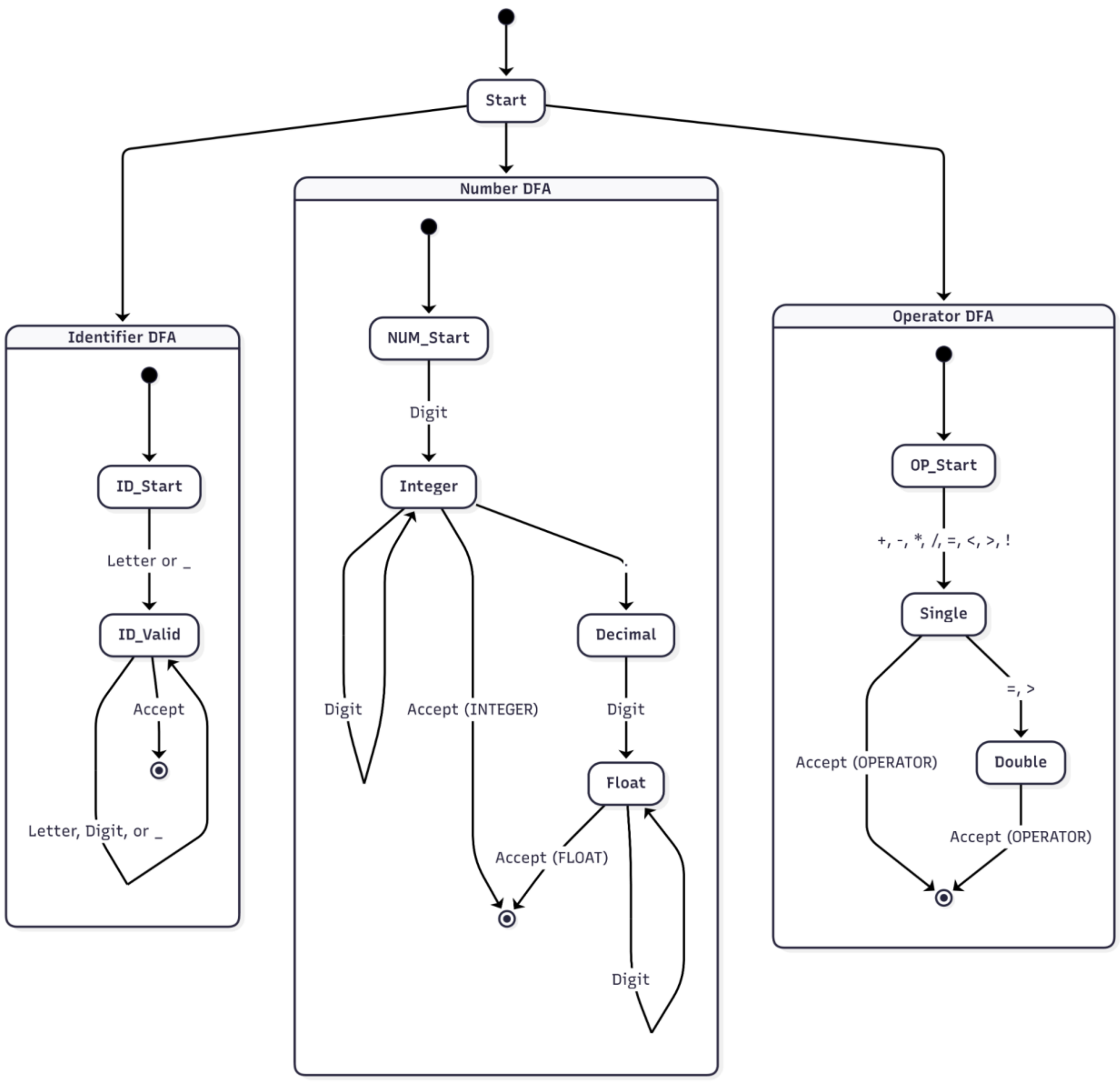


Figure 2: DFAs used for token recognition (Identifier, Number, Operator).

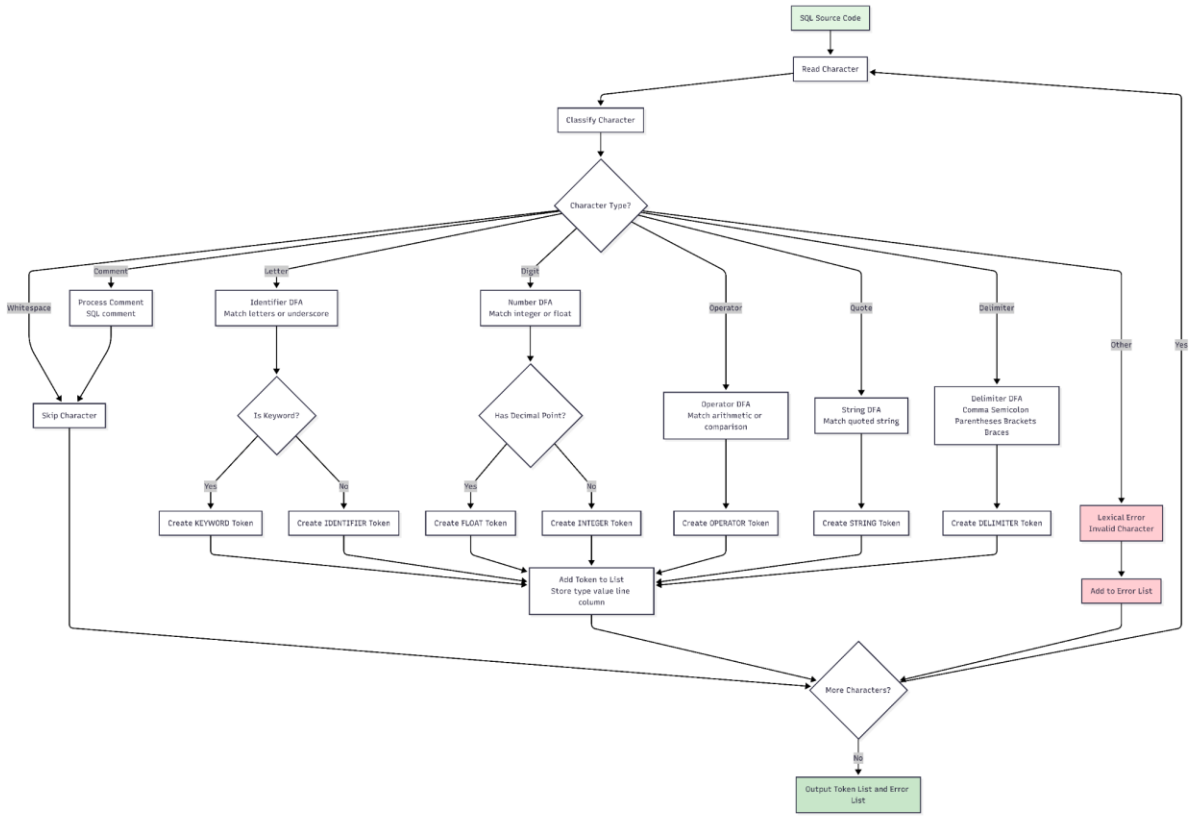


Figure 3: Lexical analysis workflow (character classification → DFA match → token emission).

For each token, the lexer stores: token type, lexeme/value, and the exact (line, column) position. This location tracking is used directly in syntax/semantic error messages.

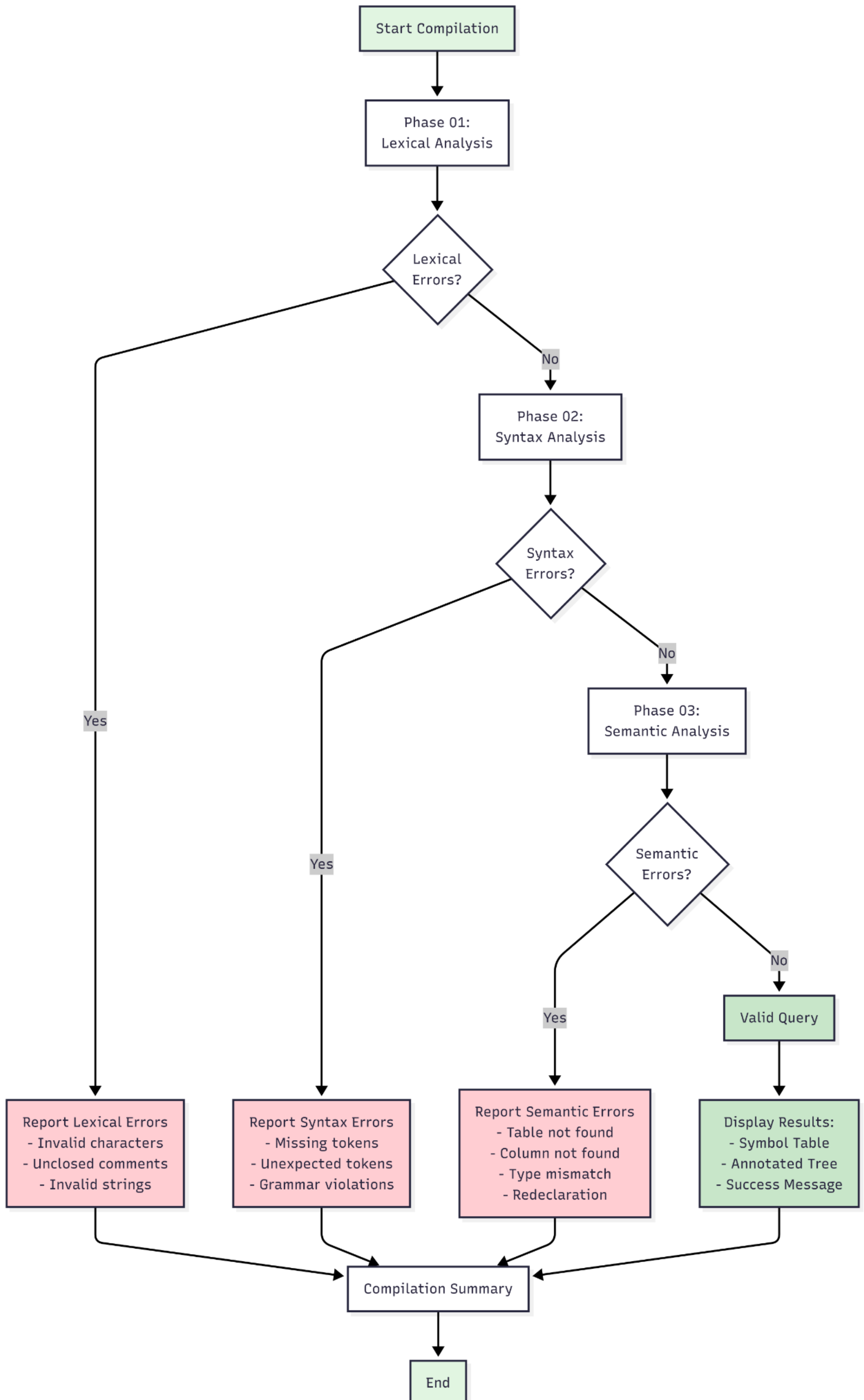
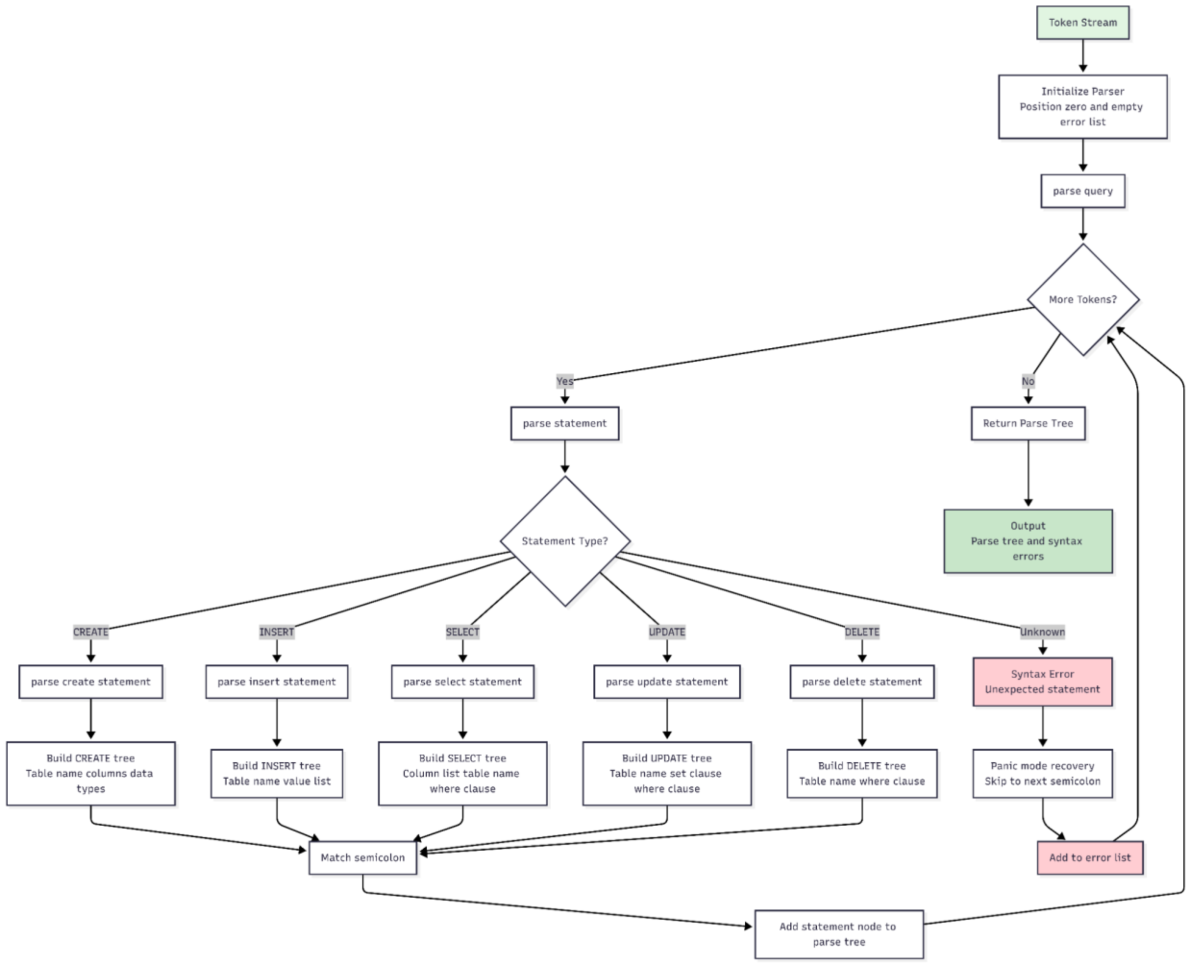

Start Compilation
Phase 01: Lexical Analysis
Lexical Errors?
Yes
No
Phase 02: Syntax Analysis
Syntax Errors?
Yes
No
Phase 03: Semantic Analysis
Semantic Errors?
Yes
No
Valid Query
Report Lexical Errors
- Invalid characters
- Unclosed comments
- Invalid strings
Report Syntax Errors
- Missing tokens
- Unexpected tokens
- Grammar violations
Report Semantic Errors
- Table not found
- Column not found
- Type mismatch
- Redeclaration
Display Results:
- Symbol Table
- Annotated Tree
- Success Message
Compilation Summary
End

Figure 4: Error handling strategy for lexical, syntax, and semantic phases.

3. Phase 02: Syntax Analysis

Syntax analysis consumes the token stream and checks it against the Mini SQL grammar (EBNF). The parser is implemented as a recursive-descent parser. It constructs a hierarchical parse tree where each node represents a grammatical construct (Query, Statement, CreateStmt, InsertStmt, SelectStmt, UpdateStmt, DeleteStmt, WhereClause, Condition, etc.).



Token Stream
Initialize Parser Position zero and empty error list
parse query
More Tokens?
Yes
No
parse statement
Return Parse Tree
Output Parse tree and syntax errors
Statement Type?
CREATE
INSERT
SELECT
UPDATE
DELETE
Unknown
parse create statement
parse insert statement
parse select statement
parse update statement
parse delete statement
Syntax Error Unexpected statement
Build CREATE tree Table name columns data types
Build INSERT tree Table name value list
Build SELECT tree Column list table name where clause
Build UPDATE tree Table name set clause where clause
Build DELETE tree Table name where clause
Panic mode recovery Skip to next semicolon
Match semicolon
Add to error list
Add statement node to parse tree

Figure 5: Parser control flow and statement dispatch (recursive descent + recovery).

When the parser encounters an unexpected token, it reports a syntax error and applies panic-mode recovery by skipping tokens until the next semicolon, allowing analysis to continue and collect multiple errors.

4. Phase 03: Semantic Analysis

Semantic analysis verifies the logical correctness of syntactically valid statements. It operates on the parse tree and uses a symbol table to validate names (tables/columns), scope, and types.

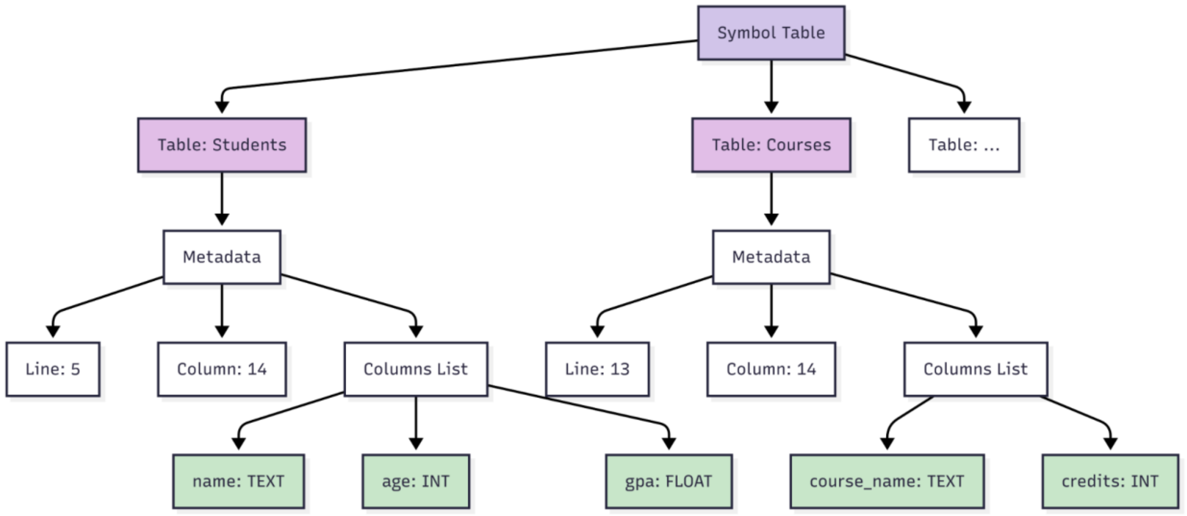
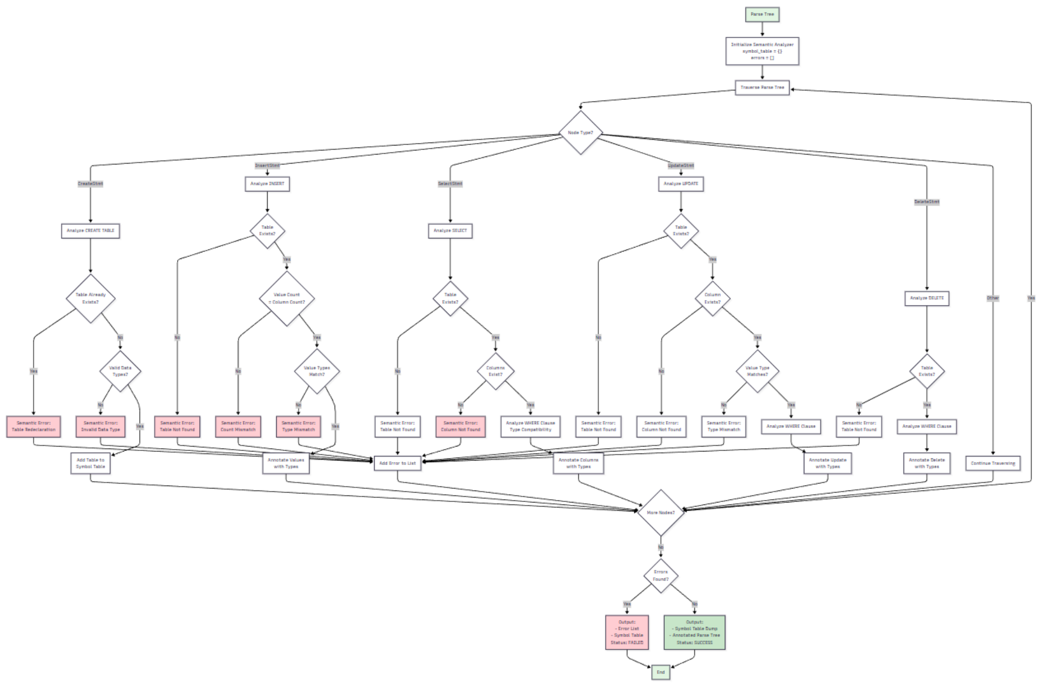


Figure 6: Symbol table structure storing tables and their column/type metadata.



Parse Tree
Initialize Semantic Analyzer
symbol_table = {}
errors = []
Traverse Parse Tree
Node Type?
CreateStmt
InsertStmt
SelectStmt
UpdateStmt
DeleteStmt
Other
Yes
Analyze CREATE TABLE
Analyze INSERT
Analyze SELECT
Analyze UPDATE
Analyze DELETE
Table Already Exists?
Valid Data Types?
Table Exists?
Value Count = Column Count?
Value Types Match?
Columns Exist?
Column Exists?
Value Type Matches?
Yes
No
Semantic Error: Table Redeclaration
Semantic Error: Invalid Data Type
Semantic Error: Table Not Found
Semantic Error: Count Mismatch
Semantic Error: Type Mismatch
Semantic Error: Column Not Found
Analyze WHERE Clause Type Compatibility
Analyze WHERE Clause
Add Table to Symbol Table
Annotate Values with Types
Add Error to List
Annotate Columns with Types
Annotate Update with Types
Annotate Delete with Types
Continue Traversing
More Nodes?
Errors Found?
Output: - Error List - Symbol Table Status: FAILED
Output: - Symbol Table Dump - Annotated Parse Tree Status: SUCCESS
End

Figure 7: Semantic analysis workflow (symbol table + type checker + outputs).

Semantic rules checked include:

• Name rules: referenced tables must exist; referenced columns must exist; redeclaration is rejected.

• Scope rules: a single-table context is maintained per statement, and WHERE identifiers resolve within it.
• Type rules: INSERT values must match declared column types; WHERE comparisons require compatible types (INT/FLOAT compatible; TEXT requires TEXT).

Errors are collected (collect-all strategy) and reported in the format:
Semantic Error: <description> (Line X, Column Y)

5. Integration and Execution Flow

The integration layer coordinates the three phases. The main entry point runs lexical analysis first; if successful, it proceeds to syntax analysis; if successful, it runs semantic analysis. Each phase emits its own error list and intermediate artifacts (tokens, parse tree, symbol table).

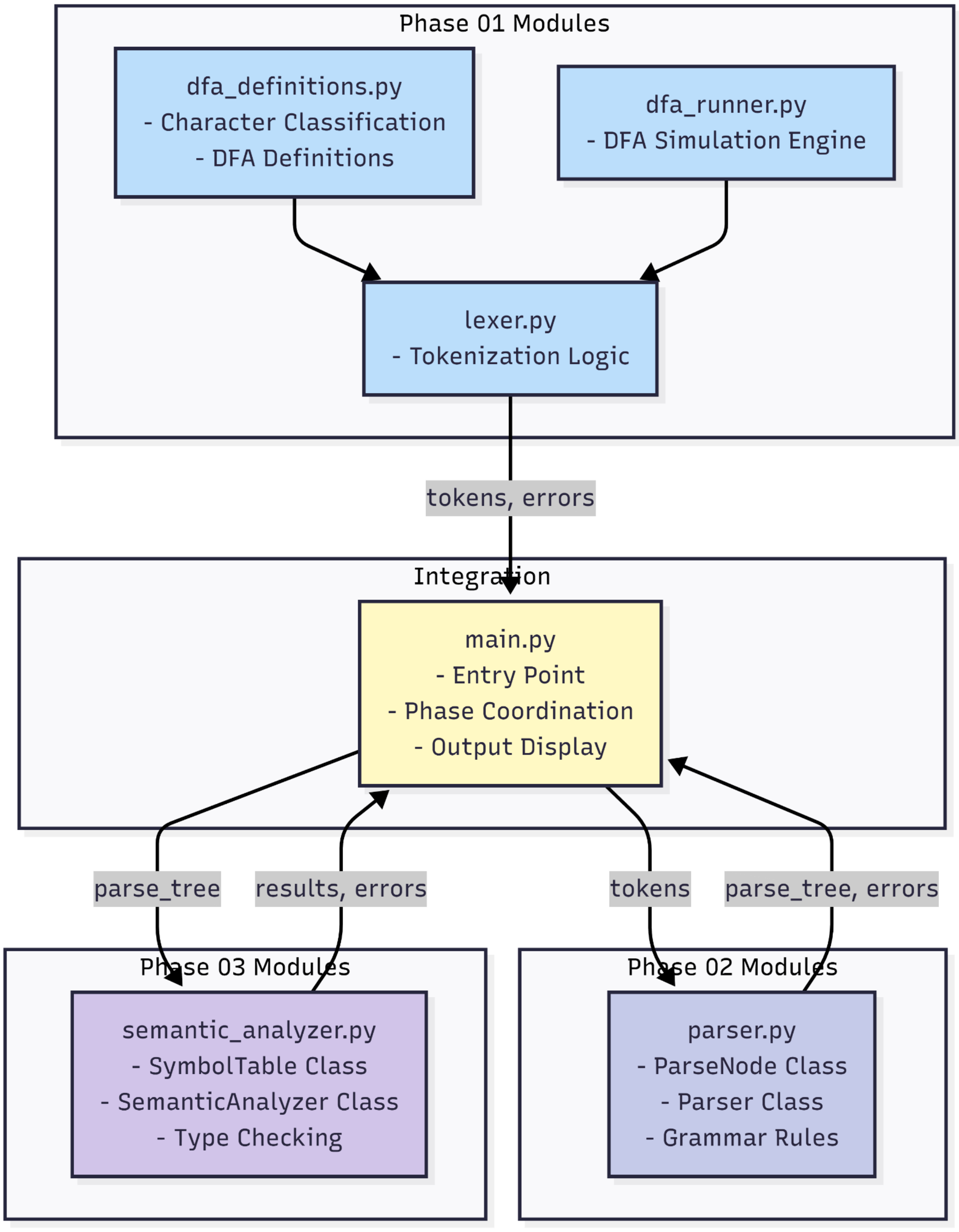


Figure 8: Main module orchestration of phases and outputs.

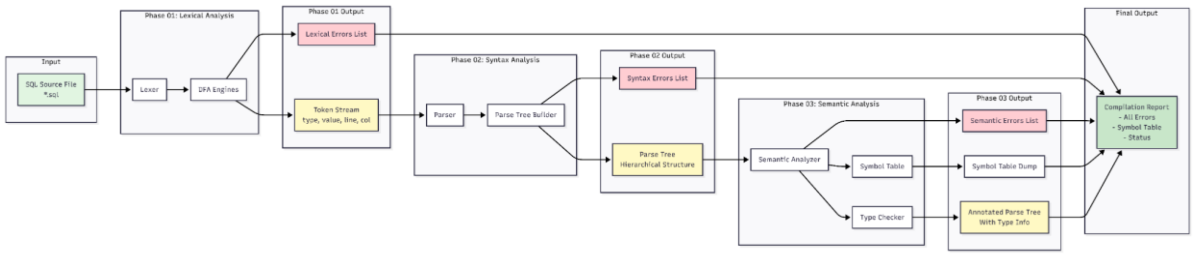


Figure 9: Data flow across phases (tokens → parse tree → semantic checks → final report).

6. Conclusion

The Mini SQL Compiler provides a complete front-end pipeline for a SQL subset. DFA-based tokenization ensures robust lexical validation, recursive-descent parsing produces structured parse trees, and the semantic analyzer guarantees logical correctness through symbol table management and type checking. The layered design supports clear error reporting and future extension of the grammar and rules.