CS 754: Programming Language April 16, 2024

MID PROJECT EVALUATION 1

Instructor: Prof. SUJIT KUMAR CHAKRABARTI Mayank 086, Aasmaan 006, Nimish 077

1 Problem Description: Core Learning Objectives and Outcomes

This project aims to develop an in-memory Relational Database Management System (RDBMS) using a functional programming approach with OCaml. The goal is to implement the foundational features of traditional RDBMS but with a unique twist by leveraging OCaml's rich type system and functional programming paradigms. This approach provides a distinct learning platform for understanding both database management systems and advanced functional programming concepts. Here are the core learning objectives and outcomes envisaged from this project:

1.1 Objectives

The project is designed around several key learning objectives:

1. **Understanding RDBMS Architecture:** We will explore how relational databases utilize tables, rows, and columns to manage data efficiently and how SQL operations such as CREATE, READ (SELECT), UPDATE, and DELETE functions within an RDBMS.
2. **OCaml Programming Proficiency:** The project aims to enhance skills in OCaml, especially for systems programming and data management, including mastering data types, modules, and pattern matching.
3. **SQL Parsing and Execution:** Perform lexical analysis and syntax parsing to interpret SQL queries using OCaml tools like 'ocamllex' and 'ocamlyacc'.
4. **Implementing Functional Programming Principles:** The project will apply functional programming principles to typical database operations, utilizing high-order functions for data manipulation tasks.

1.2 Outcomes

Upon completion of the project, the following outcomes are expected:

1. **Functional RDBMS Prototype:** A fully functional RDBMS prototype in OCaml capable of handling basic SQL queries and demonstrating effective in-memory data management.
2. **Comprehensive Documentation:** Detailed documentation that covers system design, implementation details, and usage guidelines, serving as an instructional material for future reference.

2 Solution Outline

This section describes the anticipated deliverables of the project, which encompass a wide array of outputs, including study reports, codebase, algorithmic solutions, and conceptual frameworks.

2.1 Functional RDBMS Software

- **Description:** Development of a fully functional RDBMS written in OCaml capable of handling basic SQL queries.
- **Features:**

- In-Memory Data Management for efficient data access and manipulation.
- SQL Command Interpreter utilizing OCaml's lexing and parsing tools.
- User Interface for interaction, available via command-line or basic GUI.

2.2 Study Report

- **Description:** A comprehensive report documenting the design, implementation, and evaluation of the RDBMS.
- **Contents:**
  - System Architecture and design rationale.
  - Algorithmic Solutions for SQL parsing and execution.
  - Performance Analysis including operation time complexity and memory usage.

2.3 Algorithmic Solutions

- **Description:** Detailed documentation of the algorithms implemented for various database operations.
- **Details:**
  - SQL Parsing Algorithms for efficient query processing.
  - Data Indexing Methods to improve query response times.
  - Query Optimization Techniques for enhancing performance.

3 Main References

3.1 Developing SQL Interpreter Using Antlr for .NET — Build Your Own Lexer and Parser

Citation: Kakkar, Nitin. "Developing SQL Interpreter Using Antlr for .NET — Build Your Own Lexer and Parser." Medium, [date], URL.

Relevance and Contribution: This article provides an in-depth exploration of the foundational concepts necessary for building an SQL interpreter, particularly focusing on the lexing and parsing stages. While our project utilizes OCaml rather than .NET and ANTLR, the principles outlined remain highly pertinent. The article elucidates the following aspects:

- **Lexical Analysis:** It explains how source code is transformed into tokens, emphasizing the importance of this process as the first step in interpreting SQL commands. The detailed discussion on tokenization reinforces our strategy for developing a robust lexer in OCaml that efficiently segments SQL queries into meaningful syntactic units.
- **Parsing and AST Generation:** The article delves into how tokens are analyzed to understand the syntactic structure of SQL commands, culminating in the construction of an Abstract Syntax Tree (AST). Understanding this process is crucial for our project as it forms the basis for our own parser implementation, which aims to convert parsed tokens into a structured format that our OCaml interpreter can process.
- **Tree Data Structures:** The emphasis on tree data structures as central to the compilation process provides valuable insights into managing hierarchical relationships within our interpreter, enhancing our understanding of how to efficiently handle nested SQL queries and complex data relationships in our RDBMS.

Contribution to Project: While the specific tools and programming languages differ, the theoretical groundwork laid by this article aids in solidifying our understanding of the key components of SQL interpretation. The explanations of lexing, parsing, and AST construction directly inform the development of corresponding functionalities in our OCaml-based RDBMS, ensuring that our implementation is both efficient and scalable.

4 Team's Progress

4.1 Finalization of System Workflow

Our team has successfully outlined a detailed workflow for processing SQL commands through a functional programming approach, laying a conceptual foundation for the RDBMS functionalities. Below is a detailed overview of the conceptual workflow devised for our RDBMS, from processing input to executing database operations:

4.1.1 Input Processing and Lexical Analysis

- **Input SQL Command:** Initially, a user provides an SQL command. Example: `SELECT * FROM Students WHERE Age > 20;`.
- **Lexer Implementation:** We will implement a lexer to tokenize the input string into syntactic elements like keywords and identifiers, facilitating the parsing phase.

4.1.2 Syntax Parsing and AST Creation

- **Parser Implementation:** We plan to implement a parser that will convert the token stream provided by the lexer into an Abstract Syntax Tree (AST). This AST will structurally represent the SQL command.

4.1.3 AST Interpretation and Execution

- **Interpreter Design:** An interpreter will be designed to process the AST. It will determine the necessary actions for database operations and map these to our database manipulation functions.
- **Database Function Execution:** Based on instructions from the interpreter, specific functions will be executed to manage database operations such as fetching, creating, or updating data.

4.1.4 Data Storage and Retrieval

- **Data Representation Strategy:** Our data will be stored in-memory using sophisticated data structures that emulate tables, rows, and columns.
- **Data Manipulation and Access Strategy:** Functions will be developed to manipulate these data structures directly to implement commands like `SELECT`, `INSERT`, and `UPDATE`, ensuring real-time data manipulation and retrieval.

4.2 Implementation Overview

Our team has successfully implemented a series of foundational functionalities in our in-memory RDBMS project, focusing on establishing a robust framework for database and table management. The core of our development has included the definition of necessary data structures and the creation of several critical operational functions.

4.2.1 Data Type Definitions

We have defined essential data structures to accurately represent various components of a relational database:

- Custom types for column data representation, supporting both integer and character data.
- Structured records for columns, rows, and tables that include all necessary details such as names, types, and the actual data.
- A comprehensive type to encapsulate the entire database with its constituent tables.

4.2.2 Helper Functions

A series of helper functions have been developed to facilitate the manipulation and querying of the database:

- Functions to locate and retrieve tables by name, ensuring that operations such as additions and deletions are applied to the correct entities.
- Recursive functions that allow for the dynamic modification of the database's table list without altering its inherent structure.

4.2.3 Database and Table Creation

We have implemented mechanisms to:

- Initialize a new database instance with an empty set of tables.
- Add new tables to the database, equipped with the capability to start with an empty schema and no existing data.

4.2.4 Table Modification

Our implementation supports modifying table structures dynamically:

- Functions to add new columns to existing tables, accommodating evolving data requirements.
- Capabilities to remove tables from the database, allowing for flexible schema management as the business requirements change.

4.2.5 Filtering and Data Retrieval

As our RDBMS evolves, it has become crucial to implement sophisticated filtering capabilities to efficiently query and manage the data stored in our system. The following functionalities have been developed to support dynamic data retrieval based on specific conditions:

- **Data Comparison:** We have introduced mechanisms to compare data entries against specified conditions. This functionality is essential for performing searches and queries within the database, allowing us to determine whether data entries meet certain criteria such as equality or inequality.
- **Predicate Evaluation:** Our system can evaluate complex predicates on rows of data. This is crucial for filtering operations where only rows that meet certain conditions are retrieved. It involves checking each piece of row data against the given predicate, enhancing the system's ability to handle conditional logic in queries.
- **Row Filtering:** We have implemented functions that iterate over lists of row data, applying predicate evaluations to determine which rows should be included in the query results. This functionality supports the dynamic filtering of data based on user-defined criteria, making our database more flexible and responsive to user queries.
- **Table-Specific Filtering:** To streamline operations and improve performance, filtering capabilities are extended to entire tables. This approach allows entire datasets within a table to be quickly filtered according to predefined predicates, thereby optimizing data retrieval and manipulation based on business logic.

4.2.6 Next Steps

As our project progresses, we aim to complete the development of critical components that form the backbone of our in-memory RDBMS. The upcoming phases will include the implementation of a lexer and parser to handle SQL command interpretation. These components are essential for converting user-inputted SQL commands

into a format that our system can understand and process efficiently. Following this, an interpreter will be developed to execute the parsed commands, enabling the dynamic manipulation of database content based on user queries.

Additionally, we will enhance our system's capabilities by integrating advanced SQL features such as JOIN operations, subqueries, and transaction management, which are crucial for supporting complex database operations. We will also focus on optimizing query execution to improve performance and ensure scalability. Further developments will include expanding our data aggregation functions to support a broader range of statistical computations and introducing more sophisticated relational operations to meet the diverse needs of our users. These enhancements will solidify the functionality of our RDBMS and extend its applicability to more complex real-world scenarios.

4.3 Challenges Encountered

- **Integration of Components:** One of the initial challenges includes ensuring that the lexer, parser, and interpreter work seamlessly together, facilitating smooth data flow and function execution.
- **Performance Optimization:** Another major challenge is optimizing the system for performance, balancing the computational overhead of real-time interpretation against the need for rapid data access and manipulation.