UNIVERSIDAD
NACIONAL
DE COLOMBIA

Universidad Nacional de Colombia

Lenguajes de Programación (2025966) 2024 2S

Analizador de un Lenguaje de Consulta Personalizado

Anderson Stick Barrera Tovar
Santiago García Rodriguez
Carlos Daniel García Chaparro
Manuel Nicolás Castiblanco Avendaño
Maria Paula Carvajal Martínez

Marzo 2025

Profesor: Jorge Alfonso Meléndez Acuña

1. Resumen

En este trabajo se presenta el diseño e implementación de un analizador de lenguaje de consulta SQL-like utilizando Haskell y la biblioteca Parsec. El objetivo principal es estructurar consultas SQL, identificando sus componentes clave y representándolas mediante un Árbol de Sintaxis Abstracta (AST).

El analizador desarrollado permite reconocer consultas que incluyen cláusulas SELECT, FROM y WHERE, soportando operadores de comparación y expresiones lógicas como AND y OR. Además, se ha incorporado una funcionalidad para exportar la estructura de las consultas en formato DOT, permitiendo su visualización mediante herramientas como Graphviz.

Se ha llevado a cabo un proceso de pruebas exhaustivas para validar la precisión del parser, asegurando su correcto funcionamiento en distintos escenarios. Asimismo, se han identificado y solucionado errores comunes relacionados con el manejo de espacios en blanco, la precedencia de operadores y la gestión de errores sintácticos.

A pesar de sus limitaciones, como la falta de soporte para valores de texto entre comillas y operadores avanzados como LIKE o IN, este proyecto sienta las bases para futuras mejoras. Entre los posibles trabajos futuros se plantea la extensión del parser para incluir funciones agregadas, una interfaz gráfica y una API REST para su integración en sistemas más complejos.

2. Introducción

En el campo de la computación, los lenguajes de consulta juegan un papel fundamental en la manipulación y recuperación de datos. SQL (Structured Query Language) es uno de los lenguajes más utilizados para este propósito, permitiendo a los usuarios realizar consultas complejas sobre bases de datos estructuradas. Sin embargo, la interpretación y análisis de estas consultas requieren la implementación de herramientas especializadas, como analizadores sintácticos.

En este proyecto, se desarrolla un analizador de un lenguaje de consulta personalizado utilizando **Haskell** y la biblioteca **Parsec**. Parsec es una biblioteca de parsing monádico que facilita la construcción de analizadores sintácticos eficientes y modulares. El objetivo principal es procesar y estructurar consultas SQL-like, identificando sus componentes clave y generando una representación estructurada en forma de **Árbol de Sintaxis Abstracta (AST)**.

Este documento detalla el diseño e implementación del analizador, proporcionando una base teórica sobre compiladores y parsing, así como una descripción del desarrollo del proyecto.

3. Fundamentos Teóricos

3.1. Compiladores y Parsing

Un **compilador** es un programa que traduce código fuente de un lenguaje de alto nivel a un formato ejecutable o intermedio (Aho, Lam, Sethi, & Ullman, 2006). Dentro de la estructura de un compilador, el **análisis sintáctico** es una fase fundamental que transforma una secuencia de tokens en una estructura jerárquica conocida como **Árbol de Sintaxis Abstracta (AST)** (Appel, 1998). Este árbol representa la estructura gramatical del código de manera simplificada y permite la evaluación y optimización del mismo.

3.2. Lenguajes de Programación Funcional y Haskell

Haskell es un lenguaje de programación funcional pura que enfatiza la inmutabilidad y el uso de funciones como elementos de primera clase (Bird, 2014). Su modelo basado en evaluación perezosa y su potente sistema de tipos lo convierten en una herramienta ideal para el desarrollo de analizadores sintácticos.

3.3. Parsec: Biblioteca de Parsing en Haskell

Parsec es una biblioteca de parsing basada en combinadores, lo que permite definir analizadores sintácticos de manera modular y declarativa (Leijen & Meijer, 2001). A diferencia de los analizadores tradicionales

generados mediante herramientas como *YACC* o *ANTLR*, Parsec permite construir parsers directamente en Haskell mediante combinaciones de funciones.

Algunas de las ventajas de Parsec incluyen:

- Capacidad de manejar gramáticas ambiguas y recursivas.
- Soporte para mensajes de error detallados y precisos.
- Integración con el sistema de tipos de Haskell para garantizar robustez y seguridad.

Este proyecto hace uso de Parsec para definir y estructurar consultas SQL-like, facilitando el reconocimiento de palabras clave (SELECT, FROM, WHERE), la extracción de listas de columnas y el análisis de condiciones lógicas dentro de las consultas.

En las siguientes secciones, se detallará el diseño del AST y la implementación del analizador en Haskell.

4. Diseño del Analizador

Este diseño se basa en el código desarrollado, considerando la representación estructurada de consultas SQL-like mediante Haskell y Parsec. Se ha implementado un **Árbol de Sintaxis Abstracta (AST)** para representar las consultas y un conjunto de parsers para identificar los distintos componentes de una instrucción SQL.

El analizador desarrollado consta de los siguientes módulos principales:

- **Definición del AST**: Se define la estructura de datos en Haskell para representar consultas.
- **Parser de operadores**: Se manejan operadores como =, ¿, ¡, ¿=, ¡=, y !=.
- **Parser de condiciones**: Se analiza la condición en la cláusula WHERE.
- **Parser de expresiones**: Se soportan operadores lógicos AND y OR.
- **Parser de consultas**: Se analiza la estructura general de una consulta SELECT ... FROM ... WHERE ....

Cada componente del analizador se desarrolla utilizando combinadores de Parsec, lo que permite modularidad y reutilización del código.

5. Implementación del Parser

La implementación del parser se basa en el uso de combinadores de parsers proporcionados por la biblioteca Parsec en Haskell. Se han desarrollado diferentes funciones para reconocer y procesar los elementos de una consulta SQL-like.

Los principales parsers implementados incluyen:

- **Parser de palabras clave**: Detecta y maneja palabras clave como SELECT, FROM y WHERE.
- **Parser de columnas**: Identifica y extrae los nombres de las columnas en una instrucción SELECT.
- **Parser de tablas**: Analiza el nombre de la tabla especificada en la cláusula FROM.
- **Parser de condiciones**: Procesa las condiciones presentes en WHERE, incluyendo operadores de comparación y expresiones lógicas.

Cada uno de estos parsers se integra en un parser principal que se encarga de analizar una consulta completa y generar su representación en el AST definido previamente.

6. Pruebas y Evaluación

Para garantizar la funcionalidad y precisión del analizador, se han llevado a cabo diversas pruebas unitarias y de integración. Estas pruebas validan que el parser reconoce correctamente las consultas SQL-like y maneja adecuadamente diferentes casos.

6.1. Casos de Prueba

Los casos de prueba implementados incluyen:

- Consultas simples sin WHERE.
- Consultas con una sola condición en WHERE.
- Consultas con múltiples condiciones usando AND y OR.
- Uso de operadores de comparación como =, ¿, ¡, ¿=, ¡=, y !=.
- Consultas con nombres de tablas y columnas que contienen guiones bajos y números.

6.2. Resultados Obtenidos

Los resultados muestran que el analizador es capaz de interpretar correctamente las consultas SQL-like y generar el AST esperado. Además, se han realizado pruebas con entradas malformadas para evaluar la robustez del manejo de errores.

6.3. Optimización y Mejoras

A partir de los resultados de las pruebas, se identificaron algunos errores recurrentes que fueron corregidos para mejorar la precisión y eficiencia del parser.

6.3.1. Errores Identificados y Soluciones

1. Manejo de espacios en blanco:

- **Error:** Algunas consultas fallaban debido a espacios adicionales o faltantes entre los tokens clave.
- **Causa:** El parser no contemplaba correctamente la eliminación de espacios en ciertas reglas gramaticales.
- **Solución:** Se agregaron combinadores spaces de Parsec en los lugares adecuados para garantizar un reconocimiento flexible de espacios en blanco.

2. Orden de precedencia en AND y OR:

- **Error:** El parser evaluaba AND y OR en el mismo nivel de precedencia, causando errores en consultas con lógica combinada.
- **Causa:** chainl1 procesaba AND y OR con la misma prioridad en la evaluación de la condición.
- **Solución:** Se implementó un nuevo parser que diferencia la precedencia de AND y OR, asegurando que AND se evalúe primero.

3. Manejo de errores sintácticos:

- **Error:** Los mensajes de error eran poco informativos al fallar la validación de una consulta.
- **Causa:** Parsec generaba errores genéricos sin indicar la parte exacta de la consulta donde ocurría el fallo.

- **Solución:** Se implementó try en los parsers más susceptibles de fallar y se mejoró el manejo de errores con mensajes detallados usando ¡?¿en Parsec.

4. Identificadores con caracteres especiales:

- **Error:** No se reconocían correctamente identificadores que contenían guiones bajos o números.
- **Causa:** El parser de identificadores solo aceptaba letras sin incluir otros caracteres válidos.
- **Solución:** Se modificó el parser de identificadores para aceptar `letter <|> digit <|> char '_'`, permitiendo nombres de tablas y columnas más flexibles.

Con estas mejoras, el analizador ahora tiene una mayor robustez, permitiendo procesar consultas más diversas y generando mensajes de error más claros para facilitar la depuración.

6.4. Limitaciones del Programa

El presente trabajo desarrolla un parser en Haskell utilizando *Parsec* para interpretar y transformar consultas SQL-like en estructuras de datos formales. Sin embargo, el modelo actual presenta diversas limitaciones que restringen su aplicabilidad en escenarios de análisis más complejos. A continuación, se detallan las principales deficiencias identificadas en la implementación actual:

- **Restricción en los nombres de columnas y tablas:** El parser solo permite nombres de columnas y tablas conformados por caracteres alfabéticos, lo que impide el uso de identificadores que contengan caracteres especiales como guiones, guiones bajos, números al inicio o espacios. Esta restricción limita la compatibilidad con esquemas de bases de datos que utilizan convenciones de nomenclatura más flexibles.
- **Ausencia de soporte para valores de texto entre comillas:** En su estado actual, el parser no admite valores entre comillas en la cláusula `WHERE`, lo que impide la interpretación de comparaciones que involucren datos de tipo cadena de texto. Esto representa una limitación significativa, ya que muchas consultas SQL requieren la evaluación de atributos textuales.
- **Falta de operadores avanzados:** La implementación solo reconoce operadores de comparación básicos, como igualdad e inecuaciones, excluyendo operadores más sofisticados como `LIKE`, `IN` y `BETWEEN`. Esta deficiencia restringe la capacidad del parser para analizar consultas que dependen de coincidencias parciales, listas de valores y rangos específicos.
- **Evaluación incorrecta de la precedencia entre AND y OR:** El parser no respeta la precedencia de operadores lógicos en la cláusula `WHERE`, lo que puede llevar a interpretaciones erróneas de consultas que combinan `AND` y `OR`. En SQL, el operador `AND` tiene mayor prioridad que `OR`, pero la implementación actual del modelo no refleja esta jerarquía, afectando la precisión de los resultados obtenidos.
- **Falta de reconocimiento de alias en SELECT:** El uso de alias mediante la cláusula `AS` no es soportado por el parser, lo que impide la interpretación de consultas que redefinen los nombres de las columnas en los resultados. Esta limitación afecta la compatibilidad con prácticas comunes en la manipulación y presentación de datos.
- **Ausencia de soporte para ORDER BY, GROUP BY y LIMIT:** El parser no incluye reglas para procesar cláusulas esenciales en SQL, como `ORDER BY`, `GROUP BY` y `LIMIT`. La ausencia de estas funcionalidades reduce la utilidad del modelo en entornos donde se requiere ordenar, agrupar o restringir la cantidad de datos devueltos por una consulta.

Si bien la implementación desarrollada proporciona una base funcional para el análisis estructural de consultas SQL-like, las limitaciones identificadas restringen su aplicabilidad en escenarios más avanzados. La extensión del modelo para soportar identificadores más flexibles, valores textuales, operadores avanzados y estructuras de consulta adicionales mejoraría su precisión y alineación con el estándar SQL. La corrección de la precedencia de operadores lógicos y el reconocimiento de alias en `SELECT` representan mejoras clave para garantizar una interpretación más fiel de las consultas analizadas.

7. Conclusiones y Trabajos Futuros

Este proyecto ha permitido el desarrollo de un analizador para un lenguaje de consulta SQL-like utilizando Haskell y la biblioteca Parsec. A lo largo del proceso, se abordaron diversos desafíos relacionados con el diseño del AST, la implementación de parsers eficientes y el manejo de errores sintácticos.

7.1. Conclusiones

Los principales logros y aprendizajes obtenidos en este proyecto incluyen:

- Se logró implementar un parser modular y flexible basado en combinadores de Parsec, lo que facilita su mantenimiento y ampliación.
- Se definió un **Árbol de Sintaxis Abstracta (AST)** estructurado que permite representar de manera clara y organizada las consultas SQL-like.
- Se optimizó el manejo de espacios en blanco, garantizando que el parser pueda interpretar correctamente consultas con diferentes formatos.
- Se mejoró la gestión de errores mediante mensajes descriptivos que facilitan la depuración y corrección de consultas mal formadas.
- Se realizaron pruebas exhaustivas que validaron el correcto funcionamiento del parser en diferentes escenarios, incluyendo consultas simples y complejas con operadores lógicos.

7.2. Trabajos Futuros

Si bien el analizador desarrollado es funcional y cumple con los objetivos iniciales, existen diversas oportunidades de mejora y expansión:

- **Extensión de operadores:** Incorporar soporte para operadores adicionales como ‘LIKE‘, ‘IN‘ y ‘BETWEEN‘, ampliando la expresividad del lenguaje de consultas.
- **Soporte para funciones agregadas:** Implementar el reconocimiento de funciones como ‘COUNT‘, ‘SUM‘, ‘AVG‘, ‘MIN‘ y ‘MAX‘ dentro de las consultas SQL-like.
- **Optimización del rendimiento:** Evaluar técnicas de optimización para reducir la complejidad computacional del parser, especialmente en consultas extensas o anidadas.
- **Interfaz gráfica o API REST:** Integrar el analizador con una interfaz gráfica o exponerlo como un servicio mediante una API REST para facilitar su uso.
- **Implementación de pruebas automatizadas:** Ampliar la cobertura de pruebas unitarias y de integración para garantizar la estabilidad del código ante futuras modificaciones.

Este trabajo sienta las bases para el desarrollo de sistemas más complejos de procesamiento de consultas, permitiendo futuras mejoras y adaptaciones según las necesidades del usuario o del sistema en el que se integre.

8. Bibliografía

Referencias

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools.* Addison-Wesley.

[2] Appel, A. W. (1998). *Modern Compiler Implementation in ML.* Cambridge University Press.

[3] Bird, R. (2014). *Thinking Functionally with Haskell*. Cambridge University Press.

[4] Leijen, D., & Meijer, E. (2001). *Parsec: Direct style monadic parser combinators for the real world*. Utrecht University.