PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

ESCUELA DE SISTEMAS

DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS Y COMPUTACIÓN

TEMA:

ANÁLISIS, DISEÑO Y DESARROLLO DE PROTOTIPO FUNCIONAL DE UN INTÉRPRETE PARA LA TRANSFORMACIÓN Y EJECUCIÓN DE UN LENGUAJE NATURAL ESCRITO A SENTENCIAS SQL.

AUTOR:

SEBASTIÁN PATRICIO VILLACIS MIRANDA

CHRISTIAN ROBERTO SÁNCHEZ ROMERO

QUITO, MARZO – 2020

Agradecimiento

Estoy profundamente agradecido con mi Tío Marco Lascano por su hospitalidad de llevarme a la universidad durante mi carrera. Gracias por sus historias de cómo hacer sufrir a sus estudiantes y las buenas vibras para empezar cada día.

A mi hermana que a pesar de nuestras discusiones, siempre me ha brindado la motivación, confianza y apoyo en mi crecimiento como persona, ya que sin su fuerza sería una persona muy introvertida. Además, desearle lo mejor en esta nueva etapa de su vida que es la universidad porque sé que eres una mujer de armas tomar y es por ello que alcanzaras grandes cosas.

A mis amigos por su paciencia a mi sentido del humor y personalidad que a veces admito pudo hacerles sentir incómodo. Gracias por las risas, salidas y novatadas que compartimos juntos y espero que esto no signifique un adiós sino un hasta pronto.

Sebastián Villacís Miranda

Quiero mostrar mi más profundo agradecimiento a mis padres por estar siempre presente con su paciencia y dedicación para que pueda cumplir con mis objetivos y siempre darme el ánimo

Agradezco a mis hermanos Esteban y María José quienes me apoyan en todo momento, con sus consejos y cuidado en los momentos más complicados. Gracias por sus palabras que me hacen sentir orgulloso de lo que soy.

De igual forma, agradezco a esas amistades que a través de los años se convirtieron en una hermandad. Gracias por las conversaciones, las risas y anécdotas vividas con ustedes.

Christian Sánchez

Dedicatoria

Este trabajo de disertación está dedicado a:

Mis padres por su inquebrantable paciencia, su apoyo incondicional y por sus sabios consejos y palabras. Las cuales me ayudaron a superar eventos complicados en mi vida y convertirme en la persona perseverante , valiente y honrado que soy por hoy.

A mi tía Susi que desde el cielo ve mi crecimiento, mis temores y éxitos quiero decirle que su apoyo y amor durante mi infancia y adolescencia marcaron una de las épocas más felices en mi vida. Es por ello que su partida significo un gran pesar en mi vida y hoy quiero rendirte un homenaje al gran ser que fuiste para mí.

A mi hermano que siempre está dispuesto a ayudarme ya sea académicamente como rescatándome de salidas, en las cuales las copas se me fueron de más. Por tu incondicionalidad es que quiero compartir la culminación de esta etapa de mi vida.

Sebastián Villacís Miranda

Dedico este trabajo de disertación a mi Abuelo José, por ser la persona que durante toda mi vida sus sabias palabras me han transmitido fuerza, ánimo y cariño para que consiga las metas que me he propuesto.

Christian Sánchez

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

INTRODUCCIÓN

1. Justificación

Con el avance de la tecnología, surgió la necesidad de guardar gran cantidad de información de manera organizada y coherente. Por tal motivo se desarrolló el uso de base de datos.

Actualmente, la capacidad de procesamiento de datos y recopilación de información de las computadoras es bastante ágil debido a la evolución de su arquitectura durante los años, lo que permite ayudar a las instituciones académicas y educativas, a las organizaciones y a las empresas a administrar sus sistemas y procesos de información (Singh & Solanki, 2016).

Como se sabe, las bases de datos sólo pueden responder a consultas estándar escritas en SQL y es muy poco probable que una persona común conozca SQL. Por lo que, el usuario todavía necesita dominar la base de datos lenguaje/esquema para formular completamente las consultas.

Teniendo en cuenta estas cosas, se diseña un sistema que contiene una capa inteligente que acepta las frases comunes del usuario en lenguaje natural como entrada, convierte estas frases en consultas SQL estándar y las ejecuta para recuperar datos de bases de datos relacionales.

Al obtener un nuevo nivel por encima del lenguaje SQL, se obtiene una forma pedagógica para el aprendizaje en el uso de las bases de datos, donde en un inicio, una persona puede comprender el uso básico de consultas en base de datos sin la necesidad de aprender el lenguaje SQL.

2. Planteamiento del Problema

Hoy en día, todos los sistemas de gestión de bases de datos relacionales (SGBDR) utilizan el lenguaje de consulta estructurado (SQL) para la consulta y el mantenimiento de la base de datos (Sukthankar, Maharnawar, Deshmukh, Haribhakta, & Kamble, 2017).

Provocando que, si el usuario no presenta un gran conocimiento en el lenguaje SQL, exista un gran obstáculo al momento de obtener los datos requeridos. También es posible que no conozcan el esquema de la base de datos, incluyendo nombres de tablas, formatos, sus campos y los tipos correspondientes.

Por tal motivo, es necesario que exista una interfaz inteligente que permita obtener datos de una base de datos por medio de consulta en lenguaje natural. Esto va a poder ser realizado mediante el análisis, desarrollo e implementación de un intérprete que transforme consultas en lenguaje natural a sentencias SQL.

El lenguaje natural se utiliza en la vida cotidiana, y si se puede facilitar el intercambio de información con su uso, se reducirá el coste de aprender y comprender la tecnología utilizada para ello (Pagrut, Pakmod, Kariya, Kamble, & Haribhakta, 2018).

Por lo tanto, los usuarios que no tengan conocimientos sobre bases datos puedan aprender de forma más interactiva como es la estructura del lenguaje SQL y poder obtener los datos de una base de datos de manera más dinámica.

3. Objetivos

3.1 Objetivo General

Analizar, diseñar y desarrollar un prototipo funcional de un intérprete para la transformación y ejecución de un lenguaje natural escrito a sentencias SQL.

3.2 Objetivos Específicos

- Analizar las herramientas idóneas para el desarrollo de un prototipo funcional de un intérprete p-ara la transformación y ejecución de un lenguaje natural escrito a sentencias SQL.
- Diseñar el modelo del interprete y la estructura del lenguaje natural escrito.
- Desarrollar la transformación y ejecución de un lenguaje natural escrito a sentencias SQL por medio del intérprete.
- Ejecutar el prototipo funcional del interprete.

4. Alcance

Se realizarán estudios descriptivos debido a que se describirán características del lenguaje natural escrito, ya que su base va a ser el español latinoamericano. Debido que el lenguaje natural español presenta una gran cantidad de variantes lingüísticas sólo podrá aceptar oraciones que presenten una estructura gramatical definida, la cual será sujeta al análisis y delimitación en el transcurso de la realización del trabajo de disertación.

Seguidamente, se realizarán estudios correlacionado puesto que es necesario vincular el lenguaje natural escrito con las sentencias SQL. Debido a que ambas presentan estructuras definidas, se pretende buscar su correlación para que el intérprete sea capaz de realizar la transformación previamente mencionada.

Finalmente, se realizará un prototipo funcional de un intérprete que permita la transformación del lenguaje natural escrito a sentencias SQL. Estas sentencias SQL generadas serán únicamente de consultas hacia un sistema de gestión de base de datos PostgreSQL.

CAPÍTULO 1: SISTEMA DE GESTIÓN DE BASE DE DATOS

1.1 Introducción

Con el avance de la tecnología, surgió la necesidad de guardar gran cantidad de información de manera organizada y coherente. Por tal motivo se desarrolló el sistema de gestión de base de datos o SGBD.

Silberschatz, Korth, & Sudarshan, (2011) menciona que un sistema de gestión de base de datos es: “una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos” (p.1). A esta colección de datos usualmente se la refiere como base de datos, la cual contiene información vital para la operabilidad y perdurabilidad de toda organización.

Es por ello, que el objetivo principal del SGBD es proporcionar una manera de almacenar y recuperar información de la base de datos de manera eficiente y conveniente. Ya que, esta información permitirá la toma de decisiones estratégicas sobre la dirección futura de una organización. Además, un SGBD asegura la información almacenada a pesar de fallos del sistema o de accesos no autorizados, manteniendo los datos consistentes e integro.

1.2 Antecedentes

Los SGBD más antiguos y los más recientes se encuentran estrechamente asociados con la provisión de la funcionalidad de programación API para facilitar el acceso a la base de datos para el desarrollador de software (Refaeilzadeh, Tang, Liu, Angeles, & Scientist, 2017).El primer sistema gestor de base de datos surge en 1960 y la representación de sus datos estaba basado en el modelo de red. Este es un esquema lógico basado en tablas y grafos, en donde los nodos de un grafo corresponden a entidades, las cuales se representan como conexiones (sets) entre tablas en forma de red. Posteriormente seria estandarizado como el modelo CODASYL (Conference on Data Systems Languages) y mantendría una fuerte influencia hasta el año de 1980.

A finales de la década de 1960, surgiría el modelo de datos jerárquico cuyo principal exponente fue el IMS (Information Managment System) de IBM. Como su nombre indica los datos se orientaban en una jerarquía y la forma más natural de navegación era a través de una relación padre-hijo. Es decir, la ruta por defecto es una secuencia jerárquica de arriba hacia abajo, de izquierda hacia la derecha y de adelante hacia atrás.

A inicio de la década de 1970 el modelo de datos relacional aparece gracias al trabajo teórico realizado por Edgar F. Codd y el proyecto de investigación System R en el laboratorio de investigación de San José de IBM. Este modelo se convertiría en el modelo primario para las aplicaciones comerciales de procesamiento de datos debido a su simplicidad y separación de las estructuras de almacenamiento e indexación de la vista del usuario y del programador.

1.3 Modelo Relacional

Edgar F. Codd matemático y ex IBM es el principal referente cuando se trata del modelo relacional de base de datos. En su histórico artículo “A Relational Model of Data for Large Shared Data Banks” (Un modelo relacional de datos para grandes bancos de datos compartidos) publicado en 1970 describe porque este modelo es superior en varios aspectos al modelo de red, populares en aquella época para sistemas no referenciales.

El modelo relacional proporciona un medio para describir datos con su estructura natural, permitiendo una base para un lenguaje de datos de alto nivel que maximiza la independencia entre programas y la representación y organización de los datos. Es por ello por lo que Codd considera que los datos deben ser organizados en relaciones consistentes: en tuplas, cada una con atributos consistentes. Esto permitió una evaluación más clara del alcance y las limitaciones lógicas de los sistemas de datos estructurados de aquella época, en donde tres de los principales tipos de dependencia de datos eran:

1. La dependencia del orden
2. La dependencia de indexación
3. La dependencia de la ruta de acceso

Para Donald D. Chamberlin, coinventor de SQL, esta nueva noción simplificó enormemente la especificación de las consultas y trajo una flexibilidad sin precedentes para la explotación de conjuntos de datos existentes de nuevas formas ya que los usuarios ya no deberán preocuparse por los detalles de donde o como se almacenan los datos.

Para el año 1990 Edgar F. Codd ampliaría el concepto y las nociones sobre el modelo relacional en su artículo “The Relational Model for Database Management: Version

2” (El Modelo Relacional para la Gestión de Bases de Datos : Versión 2).Desde el punto de vista de base de datos Codd clasifica los datos en dos tipos : atómicos (no puede descomponerse en piezas más pequeñas) y compuestos (combinaciones estructuradas de datos atómicos). (Codd, 1990, p.6)

En base a esta clasificación es que menciona que el único tipo de dato compuesto para el modelo relacional son las relaciones ya que todos los operadores de consulta y manipulación están sobre las relaciones, y todos ellos generan relaciones como resultado. Además, hace un énfasis en las características semánticas del modelo relacional a un nivel más detallado. Entre estas características tenemos:

- Dominios, claves primarias y claves externas
- Los valores duplicados dentro de columnas son permitidos, pero se prohíben las filas duplicadas
- Tratamiento sistemático de la información que falta, independientemente del tipo de dato que falta.

1.4 Lenguajes de Base de Datos

Los sistemas de gestión de bases de datos constan de dos lenguajes uno para especificar el esquema de la base de datos denominado lenguaje de definición de datos (LDD) y otro para reflejar las modificaciones y consultas conocido como lenguaje de manipulación de datos (LMD). Para el caso del modelo relacional Edgar F. Codd propuso que el LMD sea dividido en dos grupos: procedural (Algebra relacional) y no procedural (Calculo racional). La principal diferencia entre estos dos radica en como el usuario se formula la consulta. En el caso del algebra relacional se espera que el usuario especifique, utilizando ciertos operadores de alto nivel, como se va a obtener el resultado. Por otra parte, el cálculo relacional el usuario solo especifica las relaciones que deben mantenerse sin especificar como obtenerlos.

1.5 Lenguaje de Consulta Estructurado

El acrónimo SQL es una abreviación de “Structured Query Language.” (Lenguaje de Consulta Estructurado). El cual permite la interacción con una base de datos relacional. Donde sí se pretende obtener ciertos datos específicos, se utiliza el SQL para efectuar la petición. Este proceso se lo denomina “Query” (Groff & Weinberg, 1991, pág. 4).

El SQL no es solamente utilizado como consultor de datos para una base de datos. Además, permite la utilización de todas las funcionalidades de un DBMS como: definición de datos, recuperación de datos, manipulación de datos, control de acceso, comparación de datos, integridad de datos. (Groff & Weinberg, 1991, pág. 5).

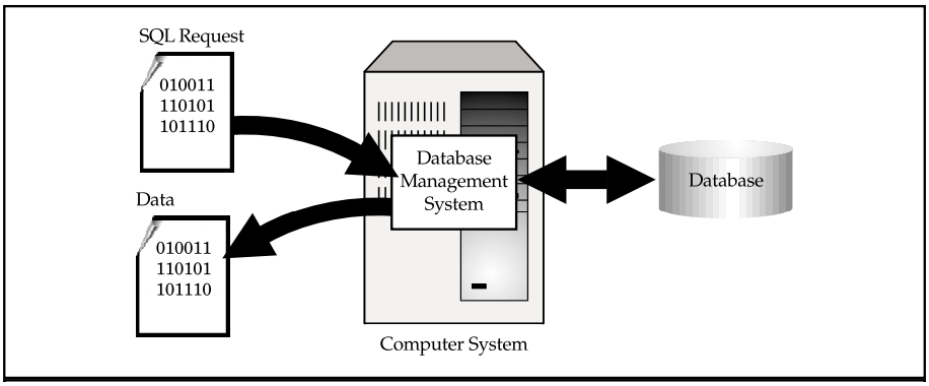

Ilustración 1. Using SQL for database access

Fuente: (Groof & Weinberg, 2002, pág. 5)

La ilustración 1 muestra como es la utilización de sentencias SQL para la obtención de información específica dentro de una base de datos. En primer lugar, se puede apreciar un sistema informático el cuál presenta una base de datos donde se almacena información importante. Por ejemplo, existe un negocio donde es necesario llevar un inventario de productos y es necesario realizar la consulta de un producto específico, se utiliza una sentencia SQL para la solicitud y por medio de un sistema de manejo de base de datos o SGBD; (Groof & Weinberg, 2002, pág. 4) se procesa

la petición SQL, recupera los datos solicitados, es devuelto la información requerida. Este proceso de solicitar datos de una base de datos y recibir de vuelta se denomina consulta de base de datos, de ahí el nombre de Lenguaje de consulta estructurado. (Groof & Weinberg, 2002, pág. 4)

SQL presenta varias ciertas características las cuales son:

- Fundamentos relacional: SQL es un lenguaje cuyo principal uso es en bases de datos relacionales, por lo que su uso es intuitivo para los usuarios, manteniendo el lenguaje SQL simple y fácil de entender. El modelo relacional presenta un fuerte fundamento teórico donde ha sido un pilar para la evolución de la aplicación de bases de datos relacionales. (Groof & Weinberg, 2002, pág. 10)
- Estructura de alto nivel, similar a la del inglés: Las sentencias SQL están estructuradas para parecer simples frases en inglés, lo que permite un fácil entendimiento y aprendizaje de dicho lenguaje. El objetivo de una sentencia SQL es describir los datos que se buscan, mas no especificar la ubicación de dichos datos. (Groof & Weinberg, 2002, pág. 10)
- Acceso a la programación de la base de datos: Uno de los principales usos del SQL es en la programación, ya que al utilizar sentencias SQL permite a los programadores desarrollar aplicaciones. Ya que permite poder ser probadas previamente a su implementación en el programa. (Groof & Weinberg, 2002, pág. 11)
- Lenguaje completo de la base de datos: SQL fue desarrollado en un principio como un lenguaje de consulta, sin embargo, al continuar con su desarrollo, SQL acepta la creación de una base de datos, gestionar su seguridad, actualizar su contenido, recuperar datos y compartir datos entre muchos usuarios concurrentes. Los conceptos de SQL que se aprenden en una parte del lenguaje

pueden aplicarse a otros comandos de SQL, lo que hace que los usuarios sean más productivos. (Groof & Weinberg, 2002, pág. 11)

- Definición de datos dinámicos: Mediante el uso de SQL, la estructura puede ser modificada de manera flexible, incluso mientras los usuarios acceden al contenido de la base de datos. Se trata de un gran avance con respecto a los lenguajes de definición de datos estáticos, que impedían el acceso a la base de datos mientras se modificaba su estructura. Así pues, el SQL proporciona la máxima flexibilidad, lo que permite que una base de datos se adapte a las necesidades cambiantes mientras las aplicaciones en línea continúan sin interrupción. (Groof & Weinberg, 2002, pág. 11)

1.5.1 Estructura

El lenguaje SQL presenta un esquema claro de cómo se debe realizar una sentencia, la cuál debe ser una solicitud para que el SGBD realice una acción específica. Por ejemplo, con una sentencia SQL se puede consultar datos, insertarlos o eliminarlos. (Groof & Weinberg, 2002, pág. 72)

Una sentencia SQL va a comenzar con un verbo que describe la acción que debe ser realizada. Para el caso de LMD dicho verbos van a ser: INSERT, UPDATE, DELETE y SELECT. Los cuales realizan diferentes acciones hacia la BDD y presentan una estructura definida.

- Sentencia SELECT: Es una sentencia SQL que sirve para consultar registros a una tabla.

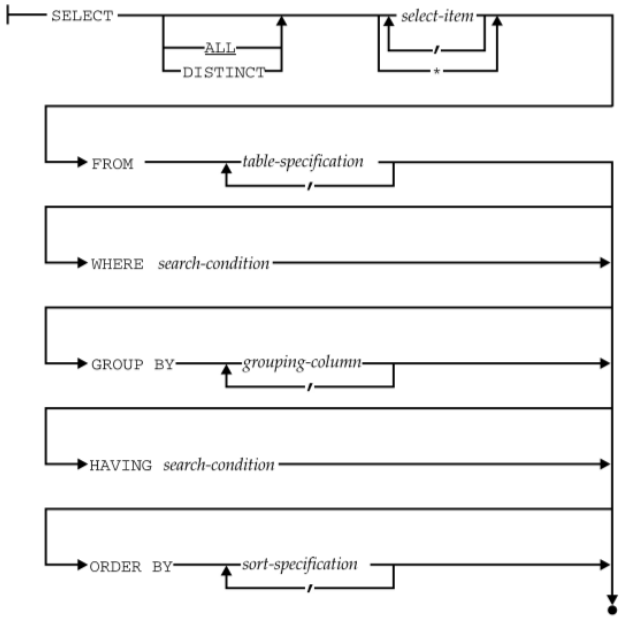

Ilustración 2. SELECT statement syntax diagram

Fuente: (Groof & Weinberg, 2002, pág. 98)

En la ilustración 2 se puede apreciar cómo es la estructura y sintaxis para una sentencia Select se comienza con el verbo y posteriormente se especifica los atributos de la tabla, o se utiliza * para especificar que se pide todos los atributos. Posteriormente se escribe la cláusula FROM precedido por el nombre de la tabla, esta sintaxis sirve para especificar la tabla de la cual se requiere.

La cláusula WHERE se usa para delimitar los registros que se desean obtener a través de la sentencia SQL. (Groof & Weinberg, 2002, pág. 109). En dicha cláusula se da los parámetros que debe cumplir los registros necesarios. GROUP BY es una cláusula que se utiliza para agrupar los registros en base a una condición específica provista por la cláusula HAVING. (Groof & Weinberg, 2002, pág. 98). Por último, la cláusula ORDER BY sirve para ordenar los registros obtenidos a través de una sentencia SQL, dicha cláusula va precedida por el

nombre de la columna por la que se organiza y por las palabras ASC o DESC que especifican si se ordena ascendente o descendentemente respectivamente.

- Sentencia UPDATE: Sentencia SQL que permite actualizar los registros de una tabla.

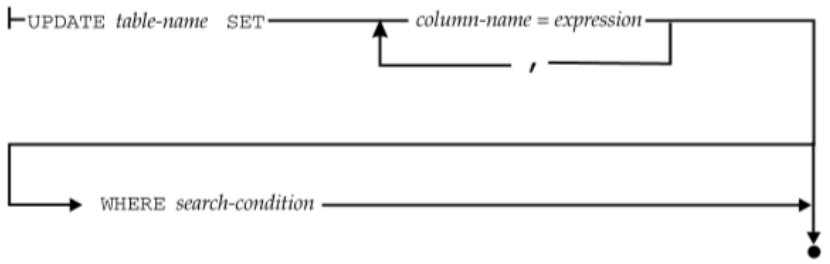

Ilustración 3 UPDATE statement syntax diagram

Fuente: (Groof & Weinberg, 2002, pág. 285)

En la ilustración 3 se puede apreciar como la cláusula UPDATE sirve para realizar una modificación a un registro o registros de una tabla. En la declaración se nombra la tabla que se va a actualizar. La cláusula WHERE proporciona la condición para saber los registros a modificar (Groof & Weinberg, 2002, pág. 285). La cláusula SET especifica qué columnas se van a actualizar y se detalla los nuevos valores de los registros.

- Sentencia DELETE: Es una sentencia SQL que permite la eliminación de un registro de la tabla.

Ilustración 4. DELETE statement syntax diagram

Fuente: (Groof & Weinberg, 2002, pág. 280)

En la ilustración 4 se muestra que la cláusula DELETE permite remover registros de una tabla, el cuál va precedido por la cláusula FROM que especifica la tabla a utilizar. Por último, la cláusula WHERE impone la condición que debe cumplir los registros para ser eliminados. (Groof & Weinberg, 2002, pág. 279)

- Sentencia INSERT: Sentencia SQL para la inserción de un registro a una tabla.

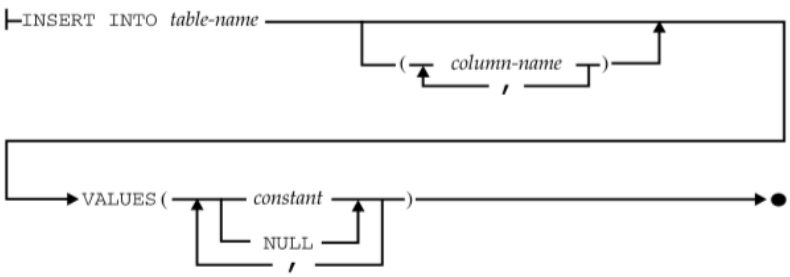

Ilustración 5. Single-row INSERT statement syntax diagram

Fuente: (Groof & Weinberg, 2002, pág. 280)

La cláusula INSERT mostrada en la ilustración 5 especifica que se va a añadir un nuevo registro a una tabla. Va precedido por la cláusula INTO donde se detalla el nombre de la tabla a utilizar. Posteriormente se especifica las columnas a ser ingresadas. Por último, la cláusula VALUES define los valores que va a presentar el nuevo registro. (Groof & Weinberg, 2002, pág. 279).

1.6 Lenguaje Natural Escrito

La investigación sobre Interfaz de Lenguaje Natural para Sistemas de Bases de Datos (NLIDB) tiene sus origines en el siglo 20. Desde entonces, existe un gran interés en esta área a pesar de sus adversidades. Si bien el lenguaje natural es fácil de aprender y usar por la gente, se ha demostrado que es difícil de dominar para una computadora. Por tal motivo, el procesamiento del lenguaje natural es considerado un esfuerzo prometedor e importante para el campo de la investigación informática (Singh & Solanki, 2016).

La primera NLIDB fue lanzada en 1973 por William Aaron Woods, denominado LUNAR, el cual involucró un sistema que respondía consultas sobre las muestras de roca traídas de la luna. Para ello se utilizó dos bases de datos: uno para el análisis químico y el segundo para las referencias bibliográfica. LUNAR funcionaba con un analizador de la Red de Transición Aumentada (ATN) y la Semántica Procesal de Woods (Sukthankar, Maharnawar, Deshmukh, Haribhakta, & Kamble, 2017).

Cinco años después surgiría LIFER/LADDER diseñado por Hendrix. Considerado como uno de los mejores sistemas de procesamiento de lenguaje de base de datos. Su función era la recuperación de información sobre los barcos de la marina de los EE. UU. Utilizó gramática semántica para analizar las consultas de los usuarios en lenguaje natural, no obstante, presentaba limitaciones ya que solo soportaba consultas simples de una tabla de una base de datos específica (Singh & Solanki, 2016).

CAPÍTULO 2: INTÉRPRETES

2.1 Estructura

Un intérprete tiene parecido a un compilador ya que, “la parte de análisis puede realizarse de manera idéntica a como se lleva a cabo en los compiladores. Es la parte de síntesis la que se diferencia sustancialmente. En el caso de la interpretación, se parte del árbol de sintaxis abstracta y se recorre, junto con los datos de entrada, para obtener los resultados” (Universitat t Jaume, 2011).

2.2 Diseño

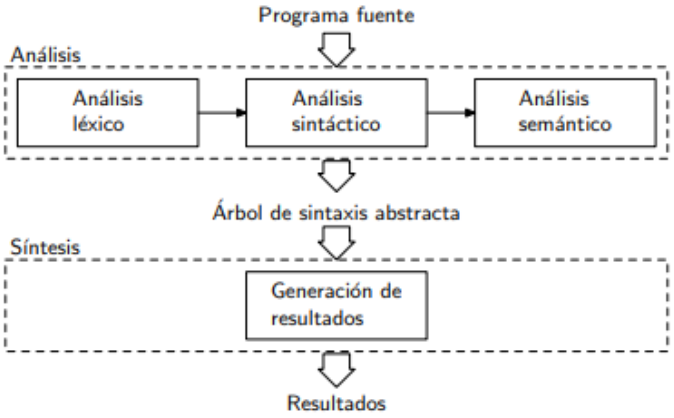

Ilustración 6 Estructura de un intérprete.

Fuente: Universitat t Jaume, 2011

2.3 Funcionamiento

Como se muestra en la ilustración 6, en la fase del análisis léxico “se analiza la entrada carácter a carácter y se divide en una serie de unidades elementales: los componentes léxicos. Cada uno de estos componentes se clasifica en una categoría y puede recibir uno o más atributos con información relevante para otras fases” (Universitat t Jaume, 2011). Cada componente léxico se le denomina token. En este proceso, existe un filtrado de caracteres especiales como espacios, comas, puntos, etc.

Esto permite que exista una división de toda la oración de entrada y obtener cuales son las palabras reservadas dentro de esta oración. Este proceso de lectura de los caracteres de entrada y construcción de tokens que posteriormente serán utilizados por el analizador sintáctico puede ser automatizado, como son el caso de los generadores de analizadores léxicos, por ejemplo, LEX, FLEX, JFLEX, entre otros. Estas herramientas presentan la licencia pública general de GNU, la cual permite y garantiza a los usuarios finales el poder de usar, modificar, compartir el software.

La función del analizador sintáctico “no es comprobar que la sintaxis del programa fuente sea correcta, sino construir una representación intermedia de ese programa (necesaria para la traducción) y, en el caso en que sea un programa incorrecto, dar un mensaje de error” (Garrido Alenda, Iñesta Quereda, & Moreno Seco, 2002, p.46). Al igual que en la fase de análisis léxico este proceso puede ser automatizado, como es el caso de los generadores de análisis sintáctico, por ejemplo, Yacc, Cup, GNU Bison, entre otros. Estas herramientas son software libre por lo que garantiza su utilización, modificación y comercialización del software.

CAPÍTULO 3: ANÁLISIS DE LA IMPLEMENTACIÓN DE UN INTÉRPRETE PARA LA TRANSFORMACIÓN DE UN LENGUAJE SEMÁNTICO A SENTENCIAS SQL

-3.1 Estructura del lenguaje natural escrito

El lenguaje natural es un conductor por el cuál de manera cotidiana se establece la comunicación entre las personas (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009, pág. 46). Por lo que al pasar los años su complejidad ha incrementado a medida que la experiencia de cada individuo aumenta, permitiendo que pueda ser utilizada para el análisis y resolución de problema complejos. Sin embargo, el lenguaje natural puede ser modelado de manera que presente una estructura simple en la resolución de problemas matemáticos o de lógica. (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009, pág. 46)

Por tal motivo cuando se utiliza el lenguaje natural se debe considerar las siguientes características:

- “Un lenguaje natural se define a partir de una gramática G, sin embargo, este se enriquece progresivamente modificando así también la gramática que la define. Esto dificulta la formalización de la definición de G.” (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009, pág. 47). Por lo tanto, la gramática no puede presentar reglas estáticas, para que siga nutriéndose y no se limite al momento de ser utilizado.

- “Un LN tiene un gran poder expresivo debido a la riqueza del componente semántico (polisemántica). Esto dificulta aún más la formalización completa de su gramática.” (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009, pág. 47) Es decir que el lenguaje natural permite expresar una misma oración de diferentes maneras, lo cual provoca que la gramática deba poder aceptar cambios en su estructura.

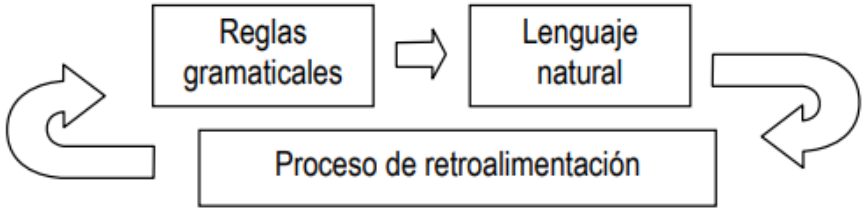

Ilustración 7. Estructura del proceso del Lenguaje Natural

Fuente: Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009, pág. 47

3.2 Definición de la Gramática

Cortez Vásquez, Vega Huerta, & Pariona Quispe (2009) mencionan que : “Una gramática G es un modelo lingüístico-matemático que describe el orden sintáctico que den cumplir las frases bien formadas de un lenguaje” (p. 49). En la ilustración 8 se puede observar la definición formal de la gramática:

$G = (V_t, V_N, P, S)$ donde:

V_T : conjunto finito de símbolos terminales del lenguaje

V_N : conjunto finito de símbolos no terminales

P : conjunto finito de reglas de producción

S : Símbolo distinguido o axioma inicial a partir del cual se reconocerán las secuencias de L aplicando sucesivamente las reglas de producción.

Ilustración 8 Definición formal de la Gramática

Fuente: (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009)

A su vez en la ilustración 9 muestra la aplicación de esta gramática para la construcción un árbol capaz de entender y procesar el lenguaje natural escrito.

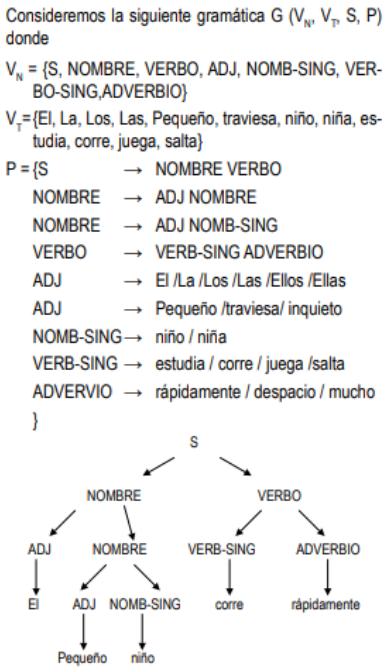

Ilustración 9. Aplicación de la gramática para el procesamiento del Lenguaje Natural Escrito

Fuente: (Cortez Vásquez, Vega Huerta, & Pariona Quispe, 2009)

3.3 Relación entre el lenguaje natural y Sentencias SQL

El diseño de modelos de procesamiento automático de la semántica del lenguaje natural escrita ha sido extensamente estudiado desde la década de 1970 y se ha convertido en un reto importante e interesante para la rama de la ciencia de la computación ya que tendría un impacto directo al ámbito industrial como social. Su relevancia se debe a dos razones principales:

1. Permite el diseño de aplicaciones dinámicas basadas en base de datos

2. Ofrece la posibilidad de comprender el papel de la sintaxis en la derivación de una semántica compartida entre un lenguaje natural y un lenguaje artificial.

En el libro Mapping Natural language into SQL in NLIDB (Mapeo del lenguaje natural en SQL en un NLIDB) de Giordani (2008) menciona que los mejores enfoques para abordar este problema pueden clasificarse en tres categorías:

- Comprensión del lenguaje hablado
- Búsqueda basada en palabras clave
- Respondiendo a la pregunta de lenguaje natural

En este estudio hemos escogido el enfoque “Búsqueda basada en palabras clave” como medio para crear la relación entre el lenguaje natural escrito y las sentencias SQL. Es decir, a través de una interacción cuidadosamente limitada con el usuario, existe la capacidad de interpretar correctamente consultas complejas de lenguaje natural, de manera genérica a través de un diccionario de sinónimos. En el cual se buscará la equivalencia de la palabra clave capturada en palabras reservadas del SQL. De esta manera, una frase en español lógicamente estructurada se traduce correctamente en una consulta SQL, que puede incluir uniones, entre otras cosas y puede ser evaluada contra un SGBDR (Sistema de gestión de base de datos relacionales). En definitiva, se desarrollará un sistema NaLIR (Interfaz de lenguaje Natural para base de datos relacionales) que incorpore el enfoque de “Búsqueda basa en palabras clave”.

3.4 Análisis comparativo de las herramientas

Analizador Léxico	Lenguaje Base	Compatibilidad Sistema Operativo	Soporte JDK	Tipo de Software	Soporte de depuración	Soporte parser generator	Modo de integración al proyecto
LISA	JAVA	LINUX/UNIX WINDOWS	JRE 1.5 o superior	Libre	NO	-	En modo grafico - IDE En modo texto – ejecutando solo el compilador
JFLEX	JAVA	UNIX WINDOWS	JDK 7 o superior	Libre	SI	CUP/CUP2 BYACC/JJAY	Maven plugin ant task standalone binary Liberia

LEX	C	UNIX	-	Propietario	NO	YACC	En modo texto – ejecutando solo el compilador
Analizador Sintáctico	Lenguaje Base	Compatibilidad Sistema Operativo	Tipo de parser generator		Gramática que soporta		
LISA	JAVA	LINUX/UNIX WINDOWS	(grammar parser and parser generator)		LL, SLR, LALR and LR		
CUP	JAVA	UNIX WINDOWS	LALR		LALR (1)		
YACC	C	UNIX	(grammar parser and parser generator)		LALR(1)		

Tabla 1 Comparación de Herramientas

LISA es un sistema que genera automáticamente un compilador/interprete a partir de especificaciones formales de lenguaje basadas en atributos gramaticales. Presenta un entorno de desarrollo integrado escrito en java lo que permite al desarrollador trabajar en un entorno visual o textual. La versión actual soporta generadores de analizadores como LALR, LL, SLR, LR, no obstante, a pesar de presentar una funcionalidad completa no permite conectar con un sistema gestor de base de datos.

LEX es un generador de programas UNIX diseñado para el procesamiento léxico de flujo de entrada de caracteres. Él cual a su vez puede generar analizadores en C o Raftor, un lenguaje que puede ser traducido automáticamente en fortran portátil. Por otra parte, JFLEX es una versión java de LEX, pero proporciona un tiempo de ejecución más rápido ya que ha sido optimizado para manejar grandes conjuntos de reglas.

Yacc es un analizador gramatical y un generador de análisis desarrollado en C, cuya función es leer una especificación gramatical (reglas que describen la estructura de la entrada) y generar código capaz de organizar los tokens en un árbol sintáctico de acuerdo con la gramática. En cambio, Cup esta desarrollado en java y realiza la mayoría de las características de Yacc con la diferencia que implementa código java embebido y produce analizadores que se implementan en java.

Como JFlex está diseñado para trabajar junto con el generador de análisis CUP e adicionalmente ambas herramientas están desarrolladas en java. Por ende, son compatibles para su integración con Netbeans, estas se integrarán a Netbeans a manera de librerías. Por su parte Netbeans permitirá crear la interfaz para la interacción con el usuario y realizará la conexión con SGBDR para la ejecución de la sentencia obtenida tras la transformación de la cadena ingresada a sentencias SQL mediante el analizador léxico (JFlex) Y el analizador sintáctico (CUP).

CAPÍTULO 4: DISEÑO DEL INTÉRPRETE Y MODELO DEL SISTEMA.

4.1 Diseño de la estructura del intérprete.

4.1.1 Análisis Léxico

Para el diseño del intérprete es necesario utilizar expresiones regulares que permitan obtener tokens válidos al tokenizar la oración de entrada. El intérprete solo va a aceptar cualquier número, sustantivos y fechas. Las expresiones regulares que se utilizan son la siguientes:

- numero = [0-9]+

  Dicha expresión regular acepta cualquier número entero positivos.
- verbo = [a-zA-Z]+r

  Se utiliza esta expresión para aceptar cualquier verbo que sea terminado en infinitivo.
- adverbio = [D|d]onde

  Solo se reconoce como adverbio a la palabra donde tanto en mayúsculas como en minúsculas.
- preposicion= ([E|e]ntre|a|desde|hasta|con|en|de|datos)

  Acepta a las preposiciones: entre, a, desde, hasta, con, en, de, datos.
- proposicion = ([l|L][o|a]s|[U|u]n[o|a]s)

  Expresión regular que acepta proposiciones como: los, las, unos, unas.
- sustantivo = [A-Z][a-z]*

  Esta expresión regular acepta a cualquier palabra que empiece con Mayúscula y se la reconoce como sustantivo.
- conjuncion=([,|y|o|e|u])

Se acepta conjunciones como: y, o, e, u, “,”.

- fecha=([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))

  Se reconoce fecha en el formato “YYY/MM/DD” con esta expresión regular.

- adjetivo = [O|o]rdenado

  Se reconoce como adjetivo a “ordenado” con esta expresión regular.

- operador = “="|">"|"<"|"<="|">=".

  Se reconoce con la expresión regular como operador a los siguientes caracteres:”=”, ">", "<", "<=", ">=”.

Además, se va a reconocer caracteres especiales como: ‘(’, ‘)’.

Ejemplo 1:

Oración de entrada: “*Buscar los Clientes donde Nombre = Kevin y Sexo = P”*

TOKEN	**LEXEMA**
Verbo	Buscar
Proposición	los
Sustantivo	Clientes
Adverbio	donde
Sustantivo	Nombre
Igual	=
Sustantivo	Kevin
Conjunción	y
Sustantivo	Sexo
Igual	=
Sustantivo	P

Tabla 2 Ejemplo 1 de tokenización de una oración

En la tabla 2 se puede apreciar que, por medio de las expresiones regulares utilizadas, la oración de entrada es tokenizado y reconoce a los verbos, proposiciones, sustantivos, adverbios y operadores lógicos como el igual.

Ejemplo 2:

Oración de entrada: "*Insertar Clientes datos (10003, Michael, Colimes, Quito, M, 2000-08-08)*"

TOKEN	**LEXEMA**
Verbo	Insertar
Sustantivo	Clientes
Preposición	datos
Paréntesis	(
Número	10003
Conjunción	,
Sustantivo	Michael
Conjunción	,
Sustantivo	Colimes
Conjunción	,
Sustantivo	Quito
Sustantivo	Sexo
Conjunción	,
Sustantivo	M
Conjunción	,
Fecha	*2000-08-08*
Paréntesis	)

Tabla 3: Ejemplo 2 de tokenización de una oración

En la tabla 3 se puede apreciar cómo tras la tokenización de la oración de entrada, se reconoce a verbos, sustantivos, preposición, número, fecha y caracteres especiales como paréntesis. La tokenización del carácter ‘,’ se reconoce como conjunción, ya que en este caso se lo puede utilizar como unión de tablas y agregación de datos como en este ejemplo.

4.1.2 Análisis Sintáctico

Una vez tokenizado a la oración del lenguaje natural, se procede a utilizar los tokens obtenidos y se realiza el árbol sintáctico con un análisis de abajo hacia arriba. Para poder desarrollar dicho árbol es necesario utilizar la siguiente gramática:

$$G \rightarrow VP\ NP$$

$$VP \rightarrow V\ R \mid V$$

$$NP \rightarrow Pre\ R \mid Pre\ R\ NP \mid Pr\ R\ NP \mid Pr\ R \mid Ad\ I \mid Adj\ I \mid Pre\ \ Pd\ Z\ Pi \mid Ad\ I\ W \mid Ad\ I\ Pre\ Z \mid Ad\ I\ Adj\ R$$

$$R \rightarrow I\ Co\ R \mid S \mid S\ Op\ S \mid S\ Op\ Num \mid S\ Op\ Fe$$

$$I \rightarrow S \mid Fe \mid num \mid S\ Op\ I$$

$$Z \rightarrow S\ Co\ Z \mid S \mid Num\ Co\ Z \mid Num \mid Fe\ Co\ Z \mid Fe$$

$$W \rightarrow Co\ W \mid S\ Op\ S \mid S\ Op\ Num$$

Símbolos Terminales: Pr, Pd, Pi, Op, Num, V, Ad, Pre, S, Co, Fe, Adj.

Símbolos No Terminales: G, VP, NP, R, I, Z, W.

Donde:

Pr: Proposición

Pd: Paréntesis derecho “(”

Pi: Paréntesis izquierdo “)”

Op: Operador

Num: Número

V: Verbo

Ad: Adverbio

Pre: Preposición

S: Sustantivo

Co: Conjunción

Fe: Fecha

Adj: Adjetivo

Ejemplo 1:

Oración de entrada: “*Buscar los Clientes donde Id = 626 y Sexo = M”*

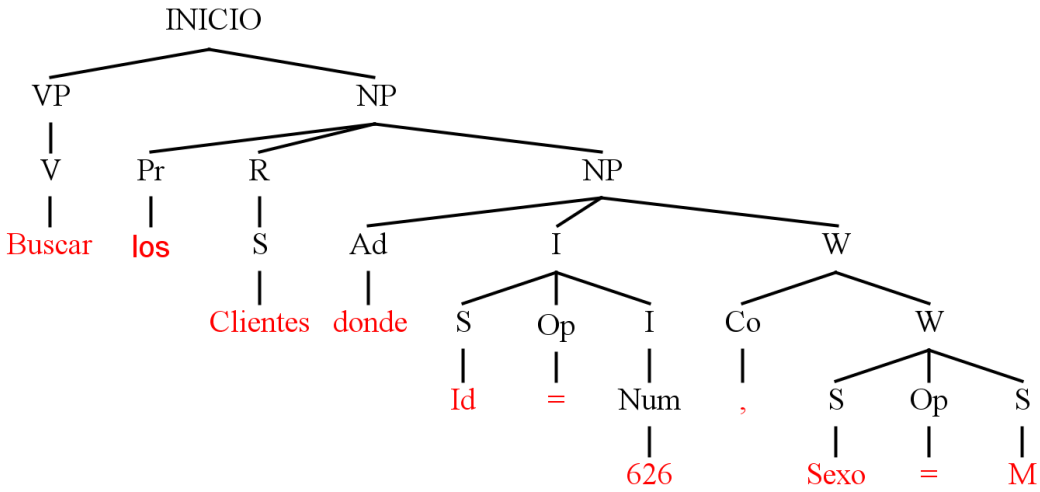

Ilustración 10. Árbol Sintáctico del Ejemplo 1

En la ilustración 10 se aprecia como se realiza el árbol sintáctico para la oración de entrada, la cual presenta una estructura de una sentencia SELECT, además el árbol sintáctico permite la conjunción de la búsqueda con dos parámetros los cuales pueden ser de diferente tipo de dato como número o sustantivo.

Ejemplo 2:

Oración de entrada: "*Insertar Clientes datos (10003, Michael, Colimes, Quito, M , 2000-08-08)"*

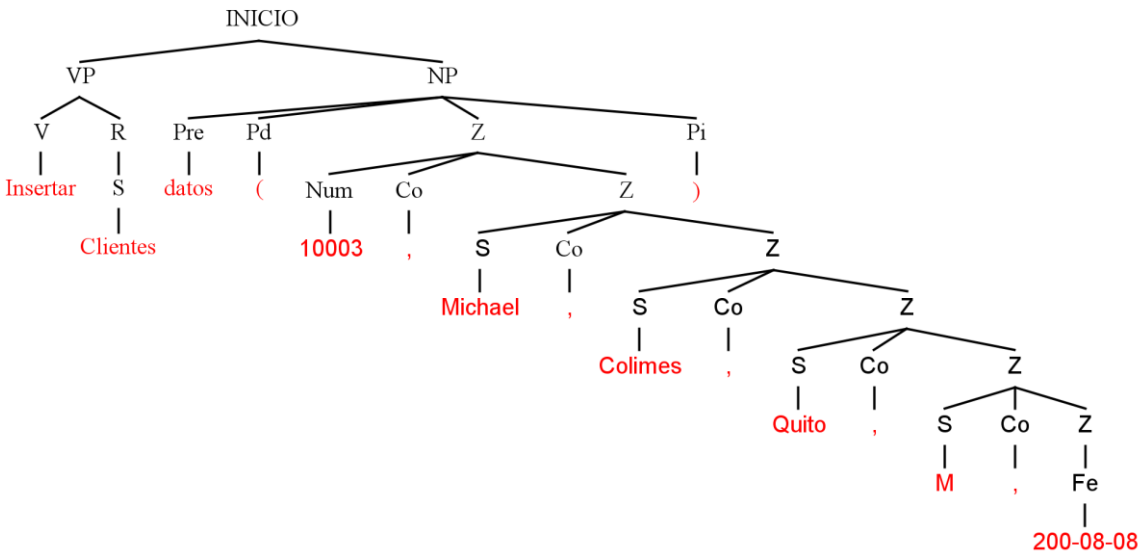

Ilustración 11. Árbol Sintáctico del Ejemplo 2

En la ilustración 11 se aprecia un árbol sintáctico creado por la oración de entrada, creando la estructura para una sentencia INSERT de un registro. Donde se acepta todos los datos para el registro.

Ejemplo 3:

Oración de entrada: "*Actualizar Clientes con Clinombre = Kevin, Clicalle = Republica donde Cliid = 10003"*

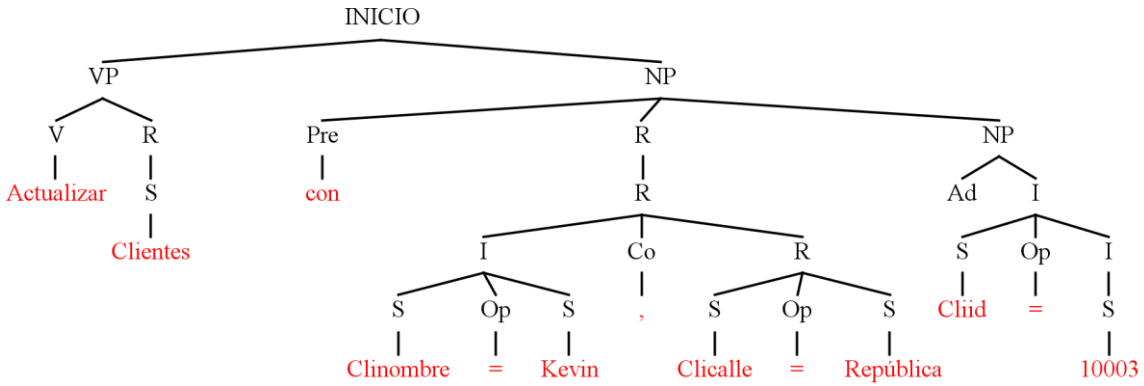

Ilustración 12. Árbol Sintáctico del Ejemplo 3

En la ilustración 12 se aprecia un árbol sintáctico creado por la oración de entrada, que genera una estructura para una sentencia de UPDATE. El árbol sintáctico permite seguir agregando los datos a ser modificados en el registro y que los registros a modificar sean específicos.

Ejemplo 4:

Oración de entrada: “*Eliminar en Clientes donde Id = 10003”*

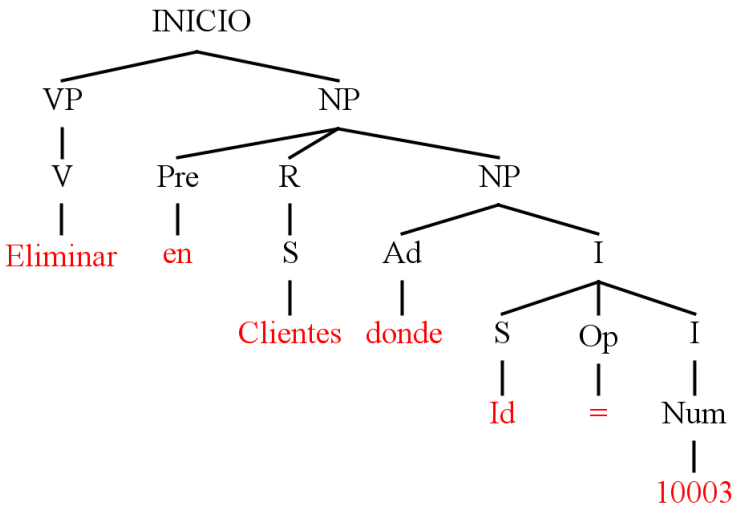

Ilustración 13. Árbol Sintáctico del Ejemplo *4*

En la ilustración 13 se aprecia un árbol sintáctico creado por la oración de entrada, que genera una estructura para una sentencia de DELETE. El árbol sintáctico permite seguir agregando los datos a ser modificados en el registro y que los registros a modificar sean específicos.

4.1.3 Diccionario de sinónimos

Una vez identificado los tokens con su respectivo lexema y realizado el árbol sintáctico, se procede a realizar la traducción. Para ello, se busca dentro del diccionario de sinónimos el valor del lexema y si existe alguna coincidencia se realiza el cambio en el árbol sintáctico.

Clave	Valor
SELECT	Buscar, visualizar, ver, hallar, seleccionar, mostrar, indicar, señalar, enseñar, visibilizar
UPDATE	Actualizar, reemplazar, modificar
DELETE	Eliminar, descartar, excluir, suprimir, quitar
INSERT	Insertar, introducir, incluir, agregar
WHERE	Donde, cual
FROM	En, de,
SET	Con
AND	Y, e
OR	O, u
*** FROM**	Los, las, unos, unas,
VALUES	Datos
BETWEEN	Entre
ORDER BY	Ordenado

Tabla 4 Diccionario de datos

4.2 Diseño de los procesos del intérprete

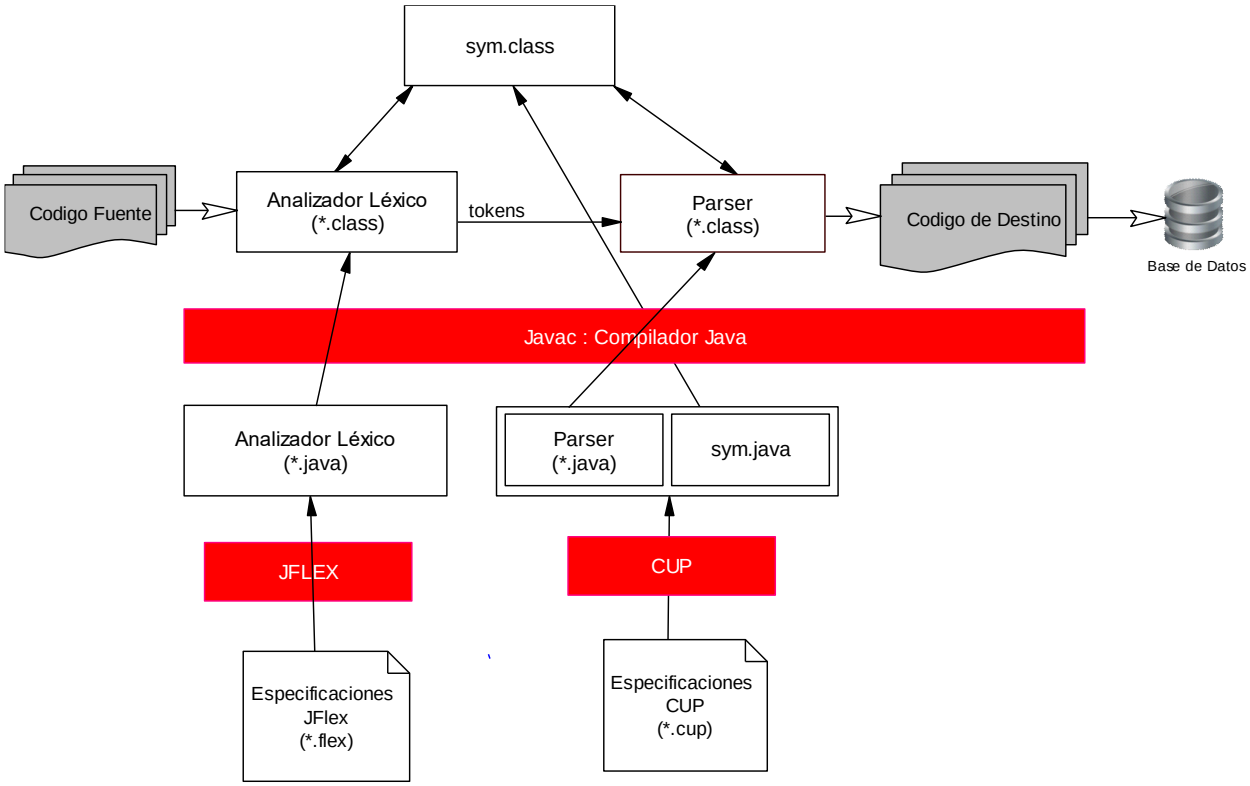

Ilustración 14. Interoperabilidad del Prototipo

4.2.1 Especificaciones Programa

El programa está compuesto por 4 secciones o paquetes los cuales cumplen una funcionalidad del interprete.

- Conexión

  El siguiente archivo presentará los métodos de conexión y ejecución de consultas con el sistema gestor de base de datos, que en este caso se manejará PostgreSQL

  - Conectar.java
- Analizadores

Los dos primeros archivos contendrán la especificación de sintaxis tanto para el analizador léxico (JFLEX) como para el analizador sintáctico (CUP).

- A_Lexico.flex
- A_Sintactico.cup

Los siguientes archivos son autogenerados, es decir, se crearán inmediatamente al momento de ejecutar y compilar los archivos batch ubicados en la sección de generadores.

- Analizador_Lexico.java
- Simbolos.java
- TERROR.java
- Análisis_sintactico.java

- Generadores

  Los siguientes archivos contendrán las especificaciones de compilación tanto para JFLEX (A_Lexico.flex) como para CUP (A_Sintactico.cup).

  - A_Lexico.bat
  - A_Sintactico.bat

- Interfaz

  Se implementará el patrón de diseño MVC en donde la vista vendrá hacer el archivo VentanaGUI, el controlador pasa hacer el archivo VentanaDP y el modelo viene hacer el archivo VentanaMD. Con esto se busca tener una mejor división del trabajo y que nuestro prototipo cumpla con escalabilidad y mantenibilidad.

  - VentanaGUI
  - VentanaDP
  - VentanaMD

4.2.2 Especificaciones JFLEX

JFLex toma como entrada una especificación de conjuntos de expresiones regulares y acciones correspondientes, la cual se va definirla en el archivo A_Lexico.flex. La especificación de este archivo va a constar de tres partes divididas por %%:

- Código de usuario:
    - Declaración de librerías y paquetes
- Opciones y declaraciones:
    - Se especifican las directrices como %cup,%class,%column,%line y %unicode. Klein, Rowe, & Décamps (2018) en su manual de usuarios de JFlex mencionan el funcionamiento de las directrices previamente mencionadas:

      %cup. – Cambia al modo de compatibilidad CUP para interactuar con un analizador generado por CUP.

      %class Lexer.- Le dice a JFlex que le dé a la clase generada el nombre de Lexer y que escriba el código en un archivo Lexer.java.

      %column.- Activa el contador de columnas (la columns actual es accedida via la variable yycolumn) .

      %line.- Activa el contador de filas (la fila actual es accedida via la variable yyline).

      %unicode.- Define el conjunto de caracteres con los que trabajará el analizador. (pp. 14-15)

- El código contenido en %{…%} será automáticamente incluida dentro del lexer generado. Aquí se puede declarar las variables y funciones que se usan dentro de las acciones del analizador léxico.
- Definición de los macros, los cuales son abreviaciones de las expresiones regulares usadas para facilitar la lectura y comprensión de las especificaciones léxicas.

- Reglas léxicas:
  - Contiene las expresiones y acciones regulares, las cuales se ejecutarán cuando el analizador léxico coincida con alguna expresión regular asociada. Es decir, a medida que el analizador va leyendo la entrada va llevando un registro de todas las expresiones regulares y al momento que encuentra la expresión con mayor coincidencia activa su acción correspondiente.

    Klein, Rowe, & Décamps (2018) en su manual de usuarios de JFLEX presenta un ejemplo de una regla léxica:

    “<YYINITIAL> "abstract" { return symbol(sym.ABSTRACT); }”

    El estado léxico YYINITIAL está predefinido e indica al analizador léxico donde comenzar a escanear.

    En este caso coincide con la entrada abstract y al ser emparejada con esta regla se ejecuta su función (acción) asociada, por lo que en este caso devolverá el símbolo CUP sym.ABSTRACT. Sin embargo, si en el caso que una acción no devuelva un valor se reanudara inmediatamente el analizador.

A partir de esta especificación y compilación generará un archivo .java al cual llamaremos Analizador_Lexico.java (un lexer) con una clase que contiene el código para el analizador y tendrá un constructor que tomará un java.io.Reader del cual la entrada es leida. A su vez, se encargará de comparar la cadena ingresada por el usuario con las expresiones regulares del lexer y ejecuta la acción correspondiente si una expresión regular coincide. Por último, dentro del código de este lexer se encontrará una función next_token() que ejecutará el analizador para leer el siguiente token de la entrada.

4.2.3 Especificaciones CUP

La especificación de la sintaxis se encuentra dividida en cuatro secciones, las cuales se definirán en el archivo A_Sintactico.cup. A continuación, se detallan la funcionalidad de estas cuatro secciones en el archivo:

1. Declaración de librerías y paquetes. – Cup emplea las mismas convenciones léxicas de java, es decir, tienen la misma sintaxis y funcionalidad que la declaración de paquetes y librerías que se encuentran un programa normal de java.
2. Componentes del código de usuario. - En esta sección se encuentran los métodos del analizador y la sobreescritura o no de las rutinas predeterminadas de informe de errores. Estos se encuentran declarados dentro de la siguiente línea: parser code {: …:}.
3. Declaraciones. - Esta sección se divide en dos partes:
    a. Lista de símbolos (terminales y no terminales).- En donde para el caso de los terminales, estas son devueltas por el analizador y colocadas en la pila del analizador. Por otra parte, con los símbolos no terminales sustituyen a una serie de objetos Symbol dentro de la pila del

analizador siempre que se reconozca el lado derecho de alguna producción

b. Declaraciones de precedencia y asociatividad. – Esta parte es opcional pero muy útil para analizar gramáticas ambiguas ya que especifica la precedencia y asociatividad de los terminales.

4. La gramática – Aquí se especifican las producciones de la gramática en donde el lado izquierdo es un no terminal seguido por el símbolo ":==", usado para asignar la acción del lado derecho. En el cual puede ir una serie de cero o más acciones, símbolos terminales o no terminales.

A partir de esta especificación y su compilación se generará un archivo .java al cual llamaremos análisis_sintactico.java con una clase que contiene el código para el analizador y tendrá un constructor que tomará como parámetro una clase Analizador_Lexico.java. Además, mediante el método parse() se realizará la transformación del lenguaje natural escrito a sentencias SQL y cuyo resultado será mostrado en un componente de texto dentro de la interfaz del programa.

CAPÍTULO 5: DESARROLLO DEL PROTOTIPO

5.1 Conexión a la base de datos

Para el prototipo desarrollado se va a utilizar una conexión a la base de datos PostgreSQL. Para ello se utiliza JDBC como paquete de la conexión de la base de datos.

```
public class conectar {
    private Connection contacto = null;
    static String cadena;
    static String user;
    static String password;
    public conectar() {
    }
    public conectar(String DatabaseName, String us, String pass) {
        cadena = "jdbc:postgresql://localhost:5432/" + DatabaseName;
        user = us;
        password = pass;
    }
```

En un principio se realiza la clase conectar con la cual se va a realizar las diferentes acciones al llamado de la base de datos. En esta clase se tiene como parámetros al usuario, contraseña y cadena, los que sirven para la identificación de la base de datos a utilizar, y así poder conectarse de manera correcta.

Posteriormente se tiene el constructor, donde se tiene como parámetros el nombre de la base de datos, usuario y contraseña. En este método es donde se instancia la conexión a la base de datos y se identifica al usuario con el que se va a realizar la conexión.

```
public ResultSet consulta(String query) throws SQLException {
```

```
        Connection con = getConexion();
        Statement declara;
        declara = con.createStatement();
        ResultSet respuesta = declara.executeQuery(query);
        con.close();
        return respuesta;
    }
```

En el método consulta se tiene como parámetro a la sentencia que se obtiene para consultarla en la base de datos. Se realiza la conexión con la base de datos, se ejecuta la sentencia y se obtiene una respuesta la cual se envía como resultado dicha respuesta.

5.2 Uso del JFlex en el análisis léxico

5.2.1 A_Lexico.flex (Especificación de sintaxis)

Como se había explicado en el capítulo 3 la estructura del archivo consta de tres 3 secciones:

Código de Usuario (Declaración de paquetes y librerías)

```
package Analizadores;
import java_cup.runtime.*;
import java.util.LinkedList;
```

Opciones y Declaraciones

Directrices

```
%public
%class Analizador_Lexico
%cupsym Simbolos
%cup
%char
%column
%full
%ignorecase
%line
```

%unicode

Funciones

```
%{
   public static LinkedList<TError> TablaEL = new LinkedList<TError>();
%}
```

Macros

```
numero = [0-9]+
verbo = [a-zA-Z]+r
adverbio = [D|d]onde
preposicion= ([E|e]ntre|a|desde|hasta|con|en|de|datos)
proposicion = ([l|L][o|a]s|[U|u]n[o|a]s)
sustantivo = [A-Z][a-z]*
conjuncion=([,|y|o|e|u])
fecha=([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))
adjetivo = [O|o]rdenado
operador = "="|">"|"<"|"<="|">="
```

Reglas Léxicas

```
<YYINITIAL> "+"         { System.out.println("Reconocio "+yytext()+" mas"); return new Symbol(Simbolos.mas, yycolumn, yyline, yytext()); }
<YYINITIAL> "*"         { System.out.println("Reconocio "+yytext()+" por"); return new Symbol(Simbolos.por, yycolumn, yyline, yytext()); }
<YYINITIAL> "/"         { System.out.println("Reconocio "+yytext()+" div"); return new Symbol(Simbolos.div, yycolumn, yyline, yytext()); }
<YYINITIAL> "("         { System.out.println("Reconocio "+yytext()+" para"); return new Symbol(Simbolos.para, yycolumn, yyline, yytext()); }
<YYINITIAL> ")"         { System.out.println("Reconocio "+yytext()+" parc"); return new Symbol(Simbolos.parc, yycolumn, yyline, yytext()); }
//-------> Simbolos ER
<YYINITIAL> {numero}    { System.out.println("Reconocio "+yytext()+" num"); return new Symbol(Simbolos.num, yycolumn, yyline, yytext()); }
<YYINITIAL> {verbo}    { System.out.println("Reconocio "+yytext()+" verbo"); return new Symbol(Simbolos.verbo, yycolumn, yyline, yytext()); }
<YYINITIAL> {adverbio}    { System.out.println("Reconocio "+yytext()+" adverbio"); return new Symbol(Simbolos.adverbio, yycolumn, yyline, yytext()); }
<YYINITIAL> {preposicion}    { System.out.println("Reconocio "+yytext()+" preposicion"); return new Symbol(Simbolos.preposicion, yycolumn, yyline, yytext()); }
<YYINITIAL> {proposicion}    { System.out.println("Reconocio "+yytext()+" proposicion"); return new Symbol(Simbolos.proposicion, yycolumn, yyline, yytext()); }
<YYINITIAL> {sustantivo}    { System.out.println("Reconocio "+yytext()+" sustantivo"); return new Symbol(Simbolos.sustantivo, yycolumn, yyline, yytext()); }
<YYINITIAL> {conjuncion}    { System.out.println("Reconocio "+yytext()+" conjuncion"); return new Symbol(Simbolos.conjuncion, yycolumn, yyline, yytext()); }
<YYINITIAL> {fecha}    { System.out.println("Reconocio "+yytext()+" fecha"); return new Symbol(Simbolos.fecha, yycolumn, yyline, yytext()); }
<YYINITIAL> {adjetivo}    { System.out.println("Reconocio "+yytext()+" adjetivo"); return new Symbol(Simbolos.adjetivo, yycolumn, yyline, yytext()); }
<YYINITIAL> {operador}    { System.out.println("Reconocio "+yytext()+" operador"); return new Symbol(Simbolos.operador, yycolumn, yyline, yytext()); }
//------> Espacios
[ \t\r\n\f]             {/* Espacios en blanco, se ignoran */}

//------> Errores Lexicos
.                       { System.out.println("Error Lexico"+yytext()+" Linea "+yyline+" Columna "+yycolumn);
                          TError datos = new TError(yytext(),yyline,yycolumn,"Error Lexico","Simbolo no existe en el lenguaje");
                          TablaEL.add(datos);}
```

5.3 Uso del JCup en el análisis sintáctico

El analizador sintáctico esta realizado en base al Jcup, donde está estructurado en cuatro secciones las cuales son:

Declaración de librerías y paquetes

```
import java_cup.runtime.Symbol;
import java.util.LinkedList;
import java.util.HashMap;
import java.lang.StringBuilder;
```

Componentes del código de usuario

Dentro de los componentes del código de usuario se realiza un HashMap del diccionario de sinónimos que se ha definido previamente. Donde se agrega en un a la palabra en LN y su traducción como un solo elemento.

```
parser code
{:
   public String resultado="";
   public static LinkedList<TError> TablaES = new LinkedList<TError>();
   public static HashMap<String, String> TablaSQL  = new HashMap<String,
String>() {{
//Palabras mostradas con Select
        put("buscar", "SELECT ");
        put("visualizar", "SELECT ");
        put("ver", "SELECT ");
        put("hallar", "SELECT ");
        put("seleccionar", "SELECT ");
        put("mostrar", "SELECT ");
        put("indicar", "SELECT ");
        put("señalar", "SELECT ");
        put("enseñar", "SELECT ");
        put("visibilizar", "SELECT ");

        //Palabras mostradas con Update
   put("actualizar", "UPDATE ");
        put("reemplazar", "UPDATE ");
        put("modificar", "UPDATE ");

        //Palabras mostradas con Delete
```

```
    put("eliminar", "DELETE ");
        put("descartar", "DELETE ");
    put("excluir", "DELETE ");
        put("suprimir", "DELETE ");
        put("quitar", "DELETE ");

        // Palabras mostradas con Insert
    put("insertar", "INSERT INTO ");
        put("introducir", "INSERT INTO ");
        put("incluir", "INSERT INTO ");
        put("agregar", "INSERT INTO ");

    put("donde", " WHERE ");
        put("cual", " WHERE ");
    put("en", "FROM ");
    put("de", "FROM ");
    put("con", "SET ");
    put("y", " AND ");
        put("e", "AND ");
    put("o", "OR ");
        put("u", "OR ");
    put("los", "* FROM ");
        put("las", "* FROM ");
        put("unos", "* FROM ");
        put("unas", "* FROM ");
    put("datos", "VALUES");
    put("entre", " BETWEEN ");
    put("ordenado", " ORDER BY ");
    put(",", ",");
}};
```

Declaraciones.

En esta sección del analizador sintáctico se realiza las declaraciones de los símbolos terminales y no terminales. Además, se declara cual va a ser el símbolo con el que comienza a realizarse el árbol sintáctico.

```
terminal mas,por,div,para,parc;
terminal String num, verbo, adverbio, preposicion, proposicion, sustantivo,
conjuncion, fecha, adjetivo, operador ;
//------> declaracion de no terminales
non terminal String INICIO,VP,NP,R,I,Z,W;
start with INICIO;
```

Gramática

Dentro de esta sección se introduce las reglas sintácticas que se han definido anteriormente. Así mismo se utiliza la tabla Hash que contiene el diccionario de sinónimos, ya que al momento de la construcción del árbol sintáctico se realiza la traducción de las palabras del LN a lenguaje SQL.

Por lo tanto, el árbol está configurado para que primero reconozca que tipo de lexema es la palabra, si no es un sustantivo, se busca dentro de la tabla Hash el lexema para encontrar su traducción y ser reemplazado en el momento de la construcción de un nodo en el árbol sintáctico.

```
INICIO::= VP:a NP:c {: resultado=a+" "+c; :};
VP::= verbo:a R:r {: RESULT=TablaSQL.get(a.toLowerCase())+r;
                  :}
    |verbo:a {:  if(RESULT==null){
                   RESULT=TablaSQL.get(a.toLowerCase());
                       }else{
                       RESULT+=TablaSQL.get(a.toLowerCase());
}
                  :};
NP::= preposicion:pr R:r{: if(RESULT==null){
                            RESULT=TablaSQL.get(pr.toLowerCase())+r;
                             }
                            else{
                              RESULT+=TablaSQL.get(pr.toLowerCase())+r;
                               } :}
    |proposicion:pr R:r {: if(RESULT==null){
                            RESULT=TablaSQL.get(pr.toLowerCase())+r;
                            }
                            else{ RESULT+=TablaSQL.get(pr.toLowerCase())+r;
                              } :}
    |preposicion:pr R:r NP:np  {: if(RESULT==null){
                              RESULT=TablaSQL.get(pr.toLowerCase())+r+np;
                            }
                            else{
RESULT+=TablaSQL.get(pr.toLowerCase())+r+np;
                              } :}
   | proposicion: pr R:r NP:np {: if(RESULT==null){
                              RESULT=TablaSQL.get(pr.toLowerCase())+r+np;

                            }
                            else{
RESULT+=TablaSQL.get(pr.toLowerCase())+r+np;
                              } :}
    |adverbio:ad I:i {: if(RESULT==null){
```

```
                                RESULT=TablaSQL.get(ad.toLowerCase())+i;
                                }
                                else{ RESULT+=TablaSQL.get(ad.toLowerCase())+i;
                                  } :}
    |adjetivo:adj I:i {: if(RESULT==null){
                                RESULT=TablaSQL.get(adj.toLowerCase())+i;
                                }
                                else{
RESULT+=TablaSQL.get(adj.toLowerCase())+i;
                                  } :}
    |preposicion:pr para:pa Z:z parc: pc {: if(RESULT==null){

RESULT=TablaSQL.get(pr.toLowerCase())+pa+z+pc;
                                }
                                else{
RESULT+=TablaSQL.get(pr.toLowerCase())+pa+z+pc;
                                  } :}
       |adverbio:ad I:i W:w  {: if(RESULT==null){
                                RESULT=TablaSQL.get(ad.toLowerCase())+i+   "   "+
w;
                                }
                                else{
RESULT+=TablaSQL.get(ad.toLowerCase())+i+ " "+ w;
                                  } :}
       |adverbio:ad I:i preposicion:pr Z:z {: if(RESULT==null){
                                RESULT=TablaSQL.get(ad.toLowerCase())+i+         "
"+TablaSQL.get(pr.toLowerCase())+" "+z;
                                }
                                else{
RESULT+=TablaSQL.get(ad.toLowerCase())+i+                                        "
"+TablaSQL.get(pr.toLowerCase())+" "+z;
                                  } :}
       |adverbio:ad I:i adjetivo:adj R:r {: if(RESULT==null){
                                    RESULT=TablaSQL.get(ad.toLowerCase())+i+  "
"+TablaSQL.get(adj.toLowerCase())+" "+r;
                                   }
                                   else{
RESULT+=TablaSQL.get(ad.toLowerCase())+i+                                        "
"+TablaSQL.get(adj.toLowerCase())+" "+r;
                                     } :}
;


R ::=  I:i conjuncion:co R:r {: if(RESULT==null){
                                  RESULT=i+TablaSQL.get(co.toLowerCase())+r;
                                }
                                else{
RESULT+=i+TablaSQL.get(co.toLowerCase())+r;
                                  } :}
    |sustantivo:s {: if(RESULT==null){RESULT=s;
                           }else{RESULT+=s;} :}
```

```
    |sustantivo:s operador:op sustantivo:su {: if(RESULT==null){
                                RESULT=s+op+" "+quote(su);
                               }
                               else{ RESULT+=s+op+" "+quote(su);
                                } :}
   |sustantivo:s operador:op num:n {: if(RESULT==null){
                                RESULT=s+op+" "+quote(n);
                               }
                               else{ RESULT+=s+op+" "+quote(n);
                                } :}
   |sustantivo:s operador:op fecha:fe {: if(RESULT==null){
                                RESULT=s+op+" "+quote(fe);
                               }
                               else{ RESULT+=s+op+" "+quote(fe);
                                } :}
    ;
I ::= sustantivo:su {: if(RESULT==null){RESULT=su;}
                    else{RESULT+=su;} :}
    | fecha:fe {:if(RESULT==null){RESULT=fe;}
                    else{RESULT+=fe;} :}
    | num:n {: if(RESULT==null){RESULT=n;}
                    else{RESULT+=n;} :}
    |sustantivo:su operador:op I:i {: if(RESULT==null){RESULT=su+" "+op+"
"+quote(i);}
                    else{RESULT+=su+" "+op+" "+quote(i);} :}
;
Z ::= sustantivo:su conjuncion:co Z:z {: if(RESULT==null){

RESULT=quote(su)+TablaSQL.get(co.toLowerCase())+z;
                               }
                               else{
RESULT+=quote(su)+TablaSQL.get(co.toLowerCase())+z;
                                } :}
   | sustantivo:su {: if(RESULT==null){
                                RESULT=quote(su);
                               }
                               else{ RESULT+=quote(su);
                                } :}
   | num:n conjuncion:co Z:z {: if(RESULT==null){
                                RESULT=n+TablaSQL.get(co.toLowerCase())+z;
                               }
                               else{
RESULT+=n+TablaSQL.get(co.toLowerCase())+z;
                                } :}
   | num: n {: if(RESULT==null){
                                RESULT=n;

                               }
                               else{ RESULT+=n;
                                } :}
| fecha:fe conjuncion:co Z:z {: if(RESULT==null){
```

```
RESULT=quote(fe)+TablaSQL.get(co.toLowerCase())+z;
                                }
                                else{
RESULT+=quote(fe)+TablaSQL.get(co.toLowerCase())+z;
                                } :}
   | fecha:fe {: if(RESULT==null){
                                 RESULT=quote(fe);
                                }
                                else{ RESULT+=quote(fe);
                                  } :}
;
W ::=   conjuncion:co W:w {: if(RESULT==null){
                                 RESULT= " "+TablaSQL.get(co.toLowerCase())+w;
                                }
                                else{              RESULT+=              "              "+
TablaSQL.get(co.toLowerCase())+w;
                                 } :}
     |sustantivo:s operador:op sustantivo:su {: if(RESULT==null){
                                 RESULT=s+op+" "+quote(su);
                                }
                                else{ RESULT+=s+op+" "+quote(su);
                                  } :}
|sustantivo:s operador:op num:nu {: if(RESULT==null){
                                 RESULT=s+op+" "+quote(nu);
                                }
                                else{ RESULT+=s+op+" "+quote(nu);
                                  } :}
;
```

5.4 Interfaces

5.4.1 Ventanas Gráficas

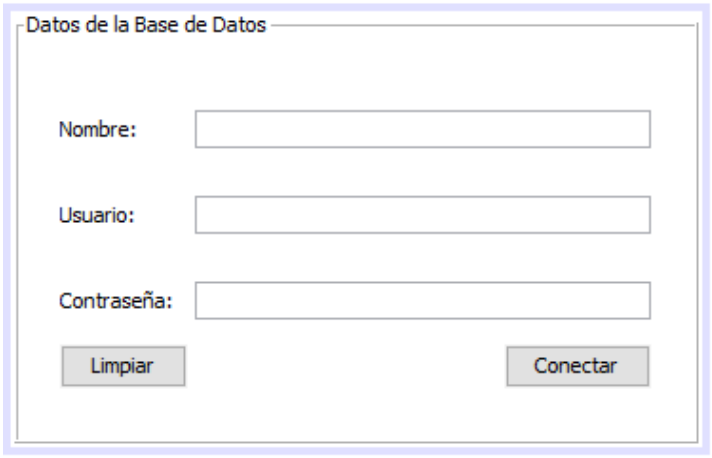

Ilustración 15. Interfaz de conexión a base de datos

Los métodos asociados a la ilustración 15 son los siguientes:

JButtonLimpiarActionPerformed

Este método se encargará de borrar toda información que se encuentra en la interfaz y dejar los campos vacíos.

```
private void jButtonLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        Limpiar();
    }
```

jButtonConectarActionPerformed

Este método captura los datos que se encuentran en los distintos campos de la interfaz para ser enviados como parámetros a la clase conectar.java. Una vez creado una instancia de esta clase se valida que se haya creado dicha conexión mediante el método getContacto(). Si pasa dicha validación se desplegará otra ventana en la cual el usuario podrá interactuar con la base de datos, caso contrario, se desplegará una ventana emergente indicando que los datos ingresados no son correctos.

```
 private void jButtonConectarActionPerformed(java.awt.event.ActionEvent evt) {
        connection = new conectar(jTextNombre.getText().trim(),
jTextUser.getText().trim(), String.valueOf(jPassword.getPassword()).trim());
        connection.getConexion();
        if (connection.getContacto() != null) {
            Limpiar();
            jDialog1.pack();
            jDialog1.setLocationRelativeTo(null);
            jDialog1.setPreferredSize(new Dimension(800, 730));
            jDialog1.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(null, "Revisar los campos de
configuración");
        }
```

}

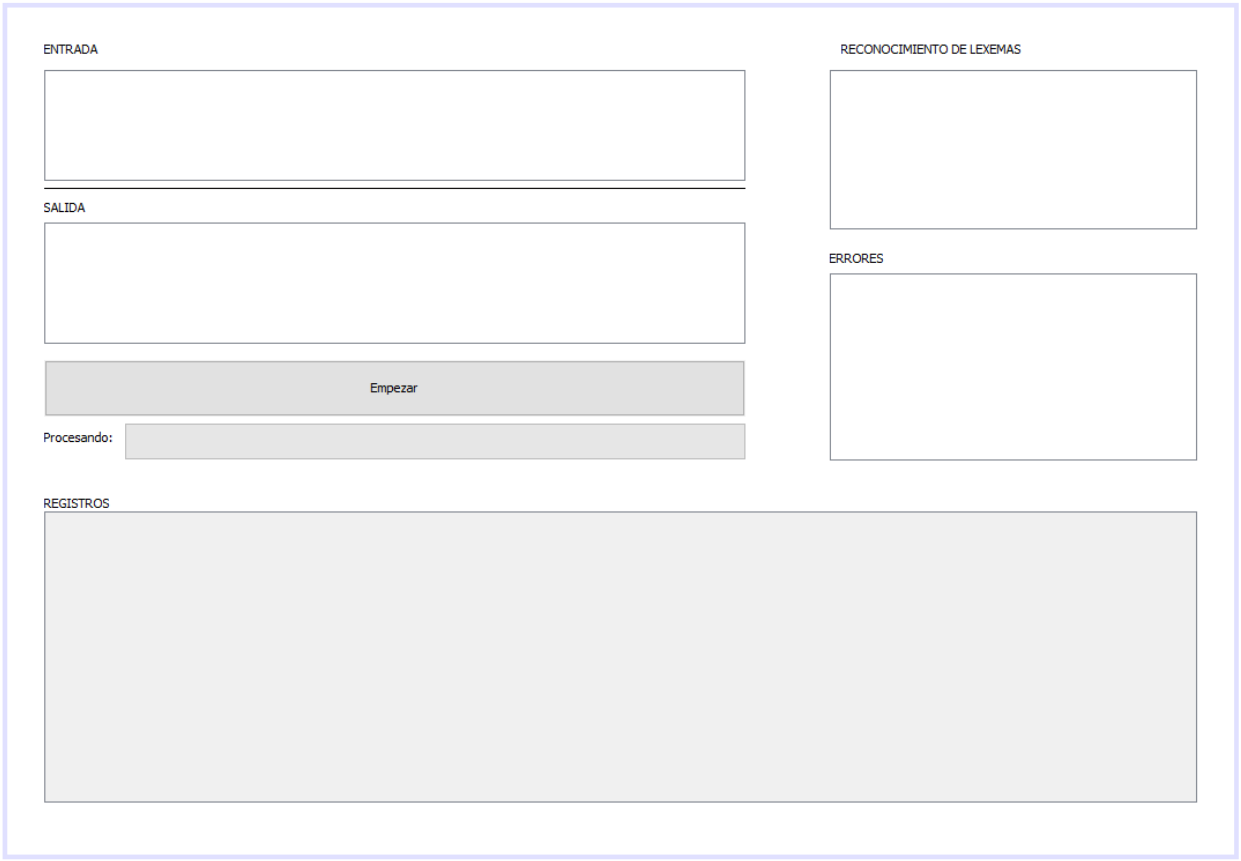

Ilustración 16. Interfaz del interprete

Las clases asociados a la ilustración 16 son las siguientes:

Task

Esta clase extiende la clase SwingWorker lo que permite realizar largas tareas de interacción con la interfaz en un hilo de fondo o en segundo plano. Esta tenie dos métodos que son los siguientes: doInBackground() y done().

El primero método deshabilitará el botón Empezar para evitar que el usuario siga ingresando consultas mientras se encuentra en ejecución el proceso actual. Al mismo

tiempo, se habilitará la animación del componente del progressbar cuyo tiempo de duración está ligado al tiempo de ejecución y recuperación de información de la sentencia obtenida del proceso de transformación de lenguaje natural escrito a sentencias SQL. Su implementación se muestra a continuación:

```
@Override
    public Void doInBackground() {
        startButton.setEnabled(false);
        jProgressBar1.setEnabled(true);
        jProgressBar1.setIndeterminate(true);
        vtDP.executeQuery();
        return null;
    }
```

El segundo método entra en acción al instante que el método doInBackground() es finalizado. Dentro de este método se llama a la clase Toolkit previamente declarada , la cual se encarga de generar un sonido al momento que se haya finalizado la tarea en segundo plano. A su vez, se restaura el estado del componente progressbar a su estado inicial, es decir, sin ninguna animación. Por último, se habilita el botón Empezar para que el usuario continúe interactuando con la interfaz. Su implementación se encuentra a continuación:

```
public void done() {
        Toolkit.getDefaultToolkit().beep();
        startButton.setEnabled(true);
        jProgressBar1.setValue(jProgressBar1.getMinimum());
        jProgressBar1.setIndeterminate(false);
        jProgressBar1.setEnabled(false);
    }
```

Task2

Esta clase extiende la clase SwingWorker cuya funcionalidad fue explicada previamente. Esta tiene dos métodos que son los siguientes: doInBackground() y done().

El primer método se encargará de ir imprimiendo los lexemas reconocidos durante el proceso de transformación de lenguaje natural escrito a sentencias SQL. El resultado del reconocimiento de lexemas es visible en el componente jTextAreaLexemas, en donde el usuario podrá ver de manera visual el proceso de análisis léxico. Además, se inhabilitará el botón de Empezar para evitar que el usuario siga generado mas procesos. La implementación de este método se encuentra a continuación:

```
public Void doInBackground() {
        jTextAreaLexemas.setText("");
        startButton.setEnabled(false);
        String datos = jTextArea1.getText();
        Analizador_Lexico lexico = new Analizador_Lexico(new
BufferedReader(new StringReader(datos)));
        try {
            while (true) {
                Symbol token = lexico.next_token();
                if (jTextAreaLexemas.getText() != "") {
                    jTextAreaLexemas.append("\n");
                }
                if (token.value == null) {
                    jTextAreaLexemas.append("EOF");
                    break;
                }
                switch (token.sym) {
                    case 1:
                        jTextAreaLexemas.append("Reconocio: " + token.value + " Error");
                        break;
                    case 2:
                        jTextAreaLexemas.append("Reconocio: " + token.value + " Mas");
                        break;
                    case 3:
                        jTextAreaLexemas.append("Reconocio: " + token.value + " Por");
```

```
                    break;
                case 4:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
División");
                    break;

                case 5:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Parentesis Derecho");
                    break;
                case 6:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Parentesis Izquierdo");
                    break;
                case 7:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Número");
                    break;
                case 8:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Verbo");
                    break;
                case 9:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Adverbio");
                    break;
                case 10:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Preposicion");
                    break;
                case 11:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Proposicion");
                    break;
                case 12:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Sustantivo");
                    break;
                case 13:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Conjuncion");
                    break;
                case 14:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Fecha");
                    break;
                case 15:
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Adjetivo");
                    break;
                case 16:
```

```
                    jTextAreaLexemas.append("Reconocio: " + token.value + "
Operador");
                    break;

                }
            }
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
        return null;
    }
```

El segundo método habilita el botón de Empezar al instante que finaliza el proceso del primer método. Su implementación se muestra a continuación:

```
    public void done() {
            startButton.setEnabled(true);
        }
```

Además de las clases explicadas previamente, la interfaz consta del siguiente método:

private void startButtonActionPerformed

Es aquí en donde inicia el proceso de transformación del lenguaje natural escrito a sentencias SQL. En primer lugar se captura la cadena ingresada por el usuario en el componente jTextArea1 para luego ser almacena en una variable de tipo String. La cual es pasada como parámetro al constructor de la clase StringReader. Este a su vez es un parámetro del constructor de la clase BufferedReader y este a su vez es el parámetro del constructor de la clase Analizador_Lexico. El cual se encarga del reconocimiento de lexemas. Una vez instanciado un objeto de esta clase podrá ser utilizado para inicializar el proceso del análisis sintactico, para ello se usa el objeto instanciado de la clase Analizador_Lexico como parámetro del constructor de la clase análisis_sintactico.

Una vez instancio el objeto de la clase análisis_sintactico se ejecuta el método parse(), el cual a partir de los lexemas reconocidos va construyendo el árbol sintactico hasta obtener la equivalencia de la cadena ingresada en sentencias SQL. Dicho resultado será presentando en el componente jTextArea2 y al mismo instante es ejecutado en la base de datos. Durante este proceso se llevan a cabo otros dos procesos en segundo plano los cuales son realizados por las clases Task y Task2, cuya funcionalidad ya fueron mencionadas previamente.

La implementación se muestra a continuación:

```
private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (!jTextArea1.getText().isEmpty()) {
        try {
            String datos = jTextArea1.getText();
            Analizador_Lexico lexico = new Analizador_Lexico(new
BufferedReader(new StringReader(datos)));
            analisis_sintactico sintactico = new analisis_sintactico(lexico);
            sintactico.parse();
            jTextArea3.append(sintactico.printErrors());
            jTextArea2.setText(sintactico.resultado);
            vtDP.setQuery(jTextArea2.getText());
            resetTabla();
            vtDP.setResultTable((DefaultTableModel) Table.getModel());
            startButton.setEnabled(false);
            task2 = new Task2();
            task2.execute();
            task = new Task();
            jProgressBar1.setEnabled(false);
            task.execute();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
        }
    } else {
```

```
            JOptionPane.showMessageDialog(null, "Ingrese una consulta");
        }
    }
```

5.4.2 Controlador

El principal método para la conexión y ejecución de la sentencia SQL a la base de datos es el siguiente:

public boolean executeQuery()

En este método se inicializa un objeto de la clase VentanaMD (Modelo) que tiene como parámetro la instancia actual del objeto VentanaDP (Controlador). Una vez inicializado este objeto(VentanaMD) se ejecuta su función asociada que es executeQuery, en la cual se encuentra la lógica de manejo de la base de datos. Su implementación es la siguiente:

```
public boolean executeQuery() {
        vtMD = new VentanaMD(this);
        return vtMD.executeQuery();
    }
```

5.4.3 Modelo

El principal método para la ejecución y retorno de información es el siguiente:

public boolean executeQuery()

Este método consta de 4 casos los cuales representan a SELECT, UPDATE , DELETE y INSERT. La implementación del método se muestra a continuación:

```
public boolean executeQuery() {
    String queryType = vtDP.getQuery().substring(0, vtDP.getQuery().indexOf(" "));
    switch (queryType) {
```

```
case "SELECT":
   try {
      ResultSet rs = con.consulta(vtDP.getQuery());
      DefaultTableModel tabla = (DefaultTableModel) vtDP.getResultTable();
      ResultSetMetaData rsmd = rs.getMetaData();
      int columnCount = rsmd.getColumnCount();
      for (int i = 1; i <= columnCount; i++) {
         String name = rsmd.getColumnName(i);
         aux.add(name);
      }
      Object[] items = aux.toArray(new String[aux.size()]);
      tabla.setColumnIdentifiers(items);
      while (rs.next()) {
         Vector v = new Vector();
         for (int i = 1; i <= columnCount; i++) {
            int colType = rsmd.getColumnType(i);
            switch (colType) {
               case -7:
                  v.add(rs.getBoolean(i));
                  break;
               case -6:
                  v.add(rs.getInt(i));
                  break;
               case -5:
                  v.add(rs.getInt(i));
                  break;
               case -1:
                  v.add(rs.getString(i));
                  break;
               case 1:
                  v.add(rs.getString(i));
                  break;
```

```
            case 2:
                v.add(rs.getBigDecimal(i));
                break;
            case 3:
                v.add(rs.getBigDecimal(i));
                break;
            case 4:
                v.add(rs.getInt(i));
                break;
            case 5:
                v.add(rs.getInt(i));
                break;
            case 6:
                v.add(rs.getFloat(i));
                break;
            case 8:
                v.add(rs.getDouble(i));
                break;
            case 12:
                v.add(rs.getString(i));
                break;
            case 91:
                v.add(rs.getDate(i));
                break;
            case 92:
                v.add(rs.getTime(i));
                break;
            case 93:
                v.add(rs.getTimestamp(i));
                break;
        }
```

```
                }
                tabla.addRow(v);
            }
            vtDP.setResultTable(tabla);
        } catch (SQLException sqlEx) {
            printSQLException(sqlEx);
            JOptionPane.showMessageDialog(null,msg.toString());
        }
        break;
      case "UPDATE":
        try {
          ResultSet rs = con.consulta(vtDP.getQuery());
        } catch (SQLException sqlEx) {
           if(sqlEx.getSQLState().equals("00000") ||
sqlEx.getSQLState().equals("02000") ){
            JOptionPane.showMessageDialog(null,"Actualizado éxitosamente");
          }else{
            printSQLException(sqlEx);
            JOptionPane.showMessageDialog(null,msg.toString());
          }
        }
        break;
      case "DELETE":
        try {
          ResultSet rs = con.consulta(vtDP.getQuery());
        } catch (SQLException sqlEx) {

          if(sqlEx.getSQLState().equals("00000") ||
sqlEx.getSQLState().equals("02000")){
            JOptionPane.showMessageDialog(null,"Eliminado éxitosamente");
          }else{
```

```
                    printSQLException(sqlEx);
                    JOptionPane.showMessageDialog(null,msg.toString());
                }
            }
            break;
        case "INSERT":
            try {
                ResultSet rs = con.consulta(vtDP.getQuery());
            } catch (SQLException sqlEx) {
                if(sqlEx.getSQLState().equals("00000") ||
sqlEx.getSQLState().equals("02000") ){
                    JOptionPane.showMessageDialog(null,"Insertado éxitosamente");
                }else{
                    printSQLException(sqlEx);
                    JOptionPane.showMessageDialog(null,msg.toString());
                }
            }
            break;
      }
    return true;
   }
```

CAPÍTULO 6: PROTOTIPO FUNCIONAL DE SISTEMA

6.1 Pruebas del intérprete y resultados obtenidos

6.1.1 Pruebas con sentencias select

Prueba 1:

Oración de entrada: "*Buscar los Clientes donde Clinombre = Michael y Cliid > 900*"

Resultado

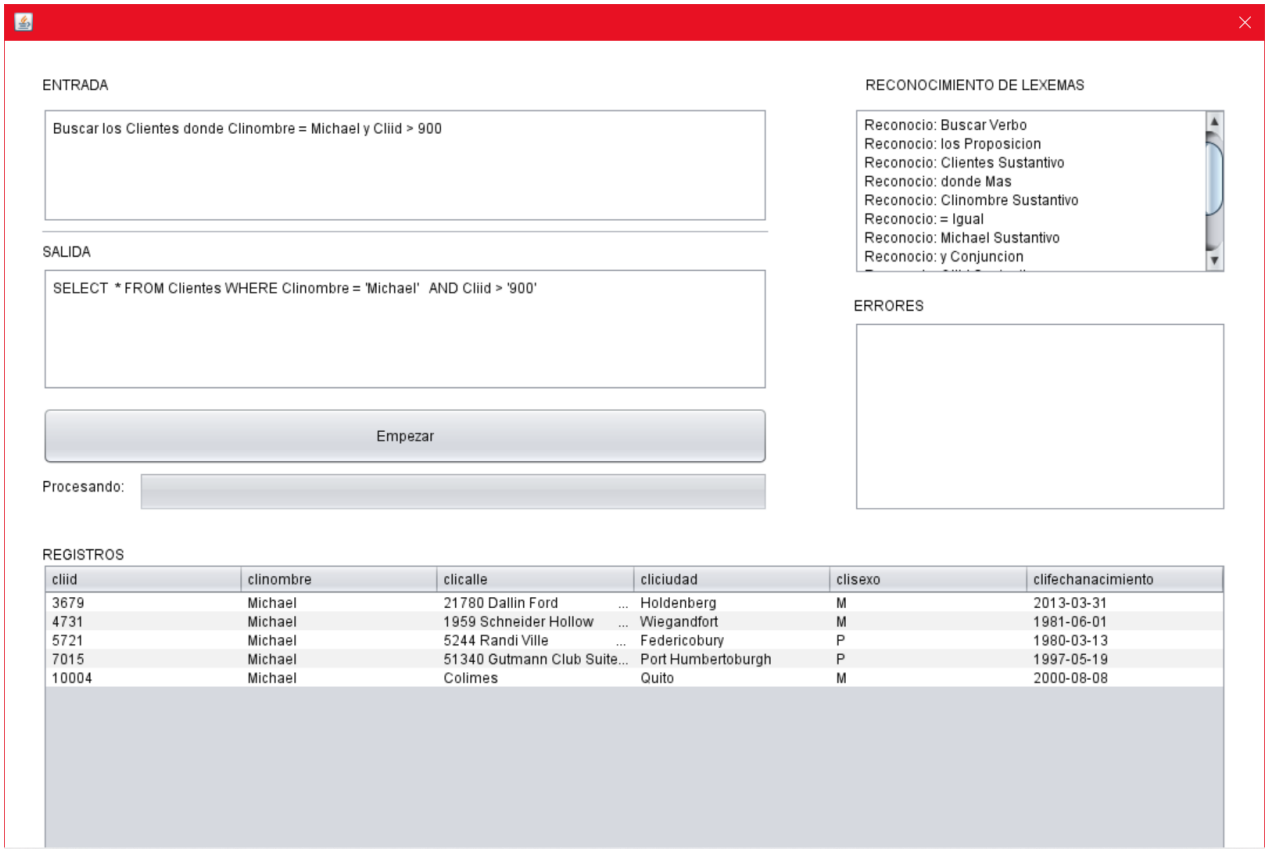

Ilustración 17. Resultado del Intérprete de la Sentencia de la Prueba 1

Prueba 2:

Oración de entrada: “*Buscar Clinombre, Clicalle en Clientes*”

Resultado

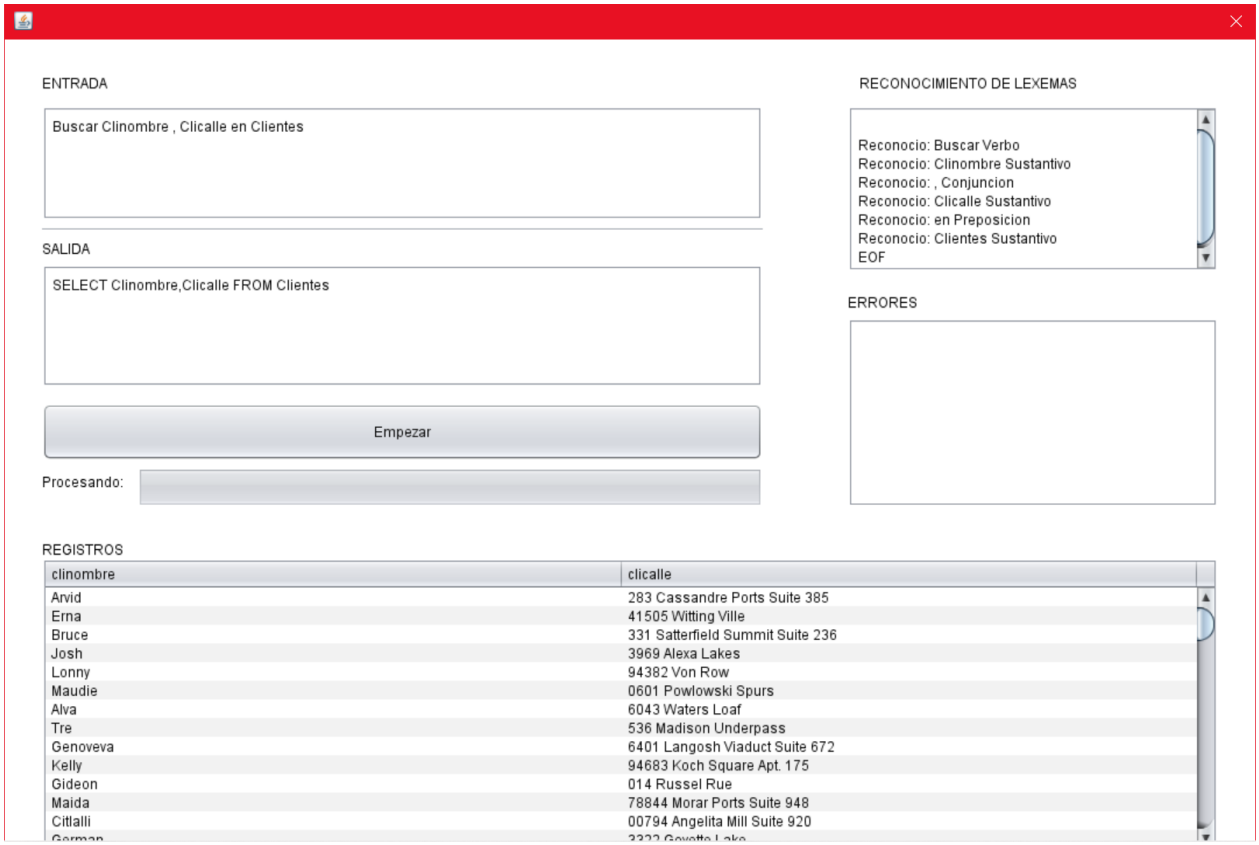

Ilustración 18. Resultado del Intérprete de la Sentencia de la Prueba 2

Prueba 3:

Oración de entrada: “Hallar unas Sucursales, Cuentas”

Resultado:

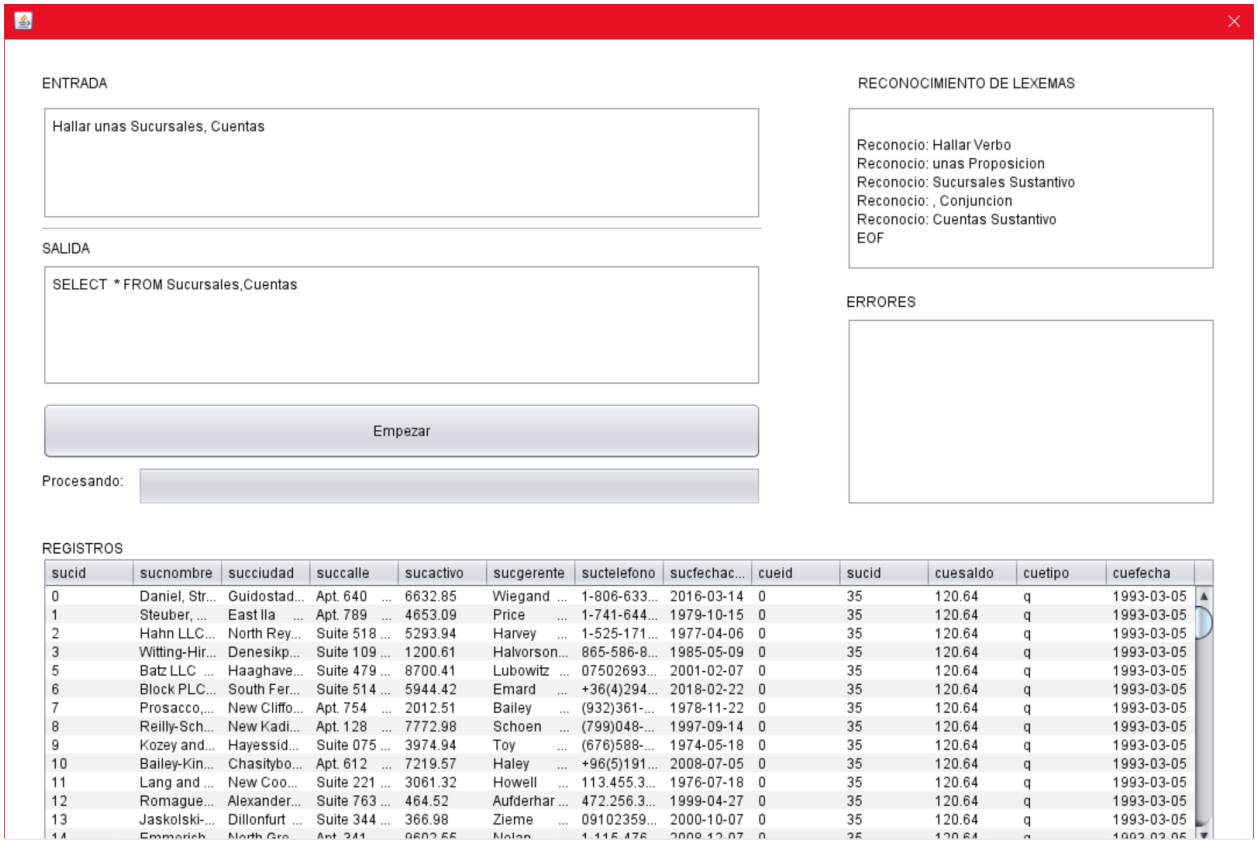

Ilustración 19. Resultado del Intérprete de la Sentencia de la Prueba 3

6.1.2 Pruebas con sentencias insert

Prueba 1:

Oración de Entrada: "*Insertar Clientes datos (10005, Michael, Colimes, Quito, M , 2000-08-08)"*

Resultado

Ilustración 20. Resultado del Intérprete de la Sentencia INSERT

6.1.3 Pruebas con sentencias update

Prueba 1:

Oración de Entrada: "*Actualizar Clientes con Cliciudad = Loja, Clicalle = Republica donde Cliid = 10005"*

Resultado

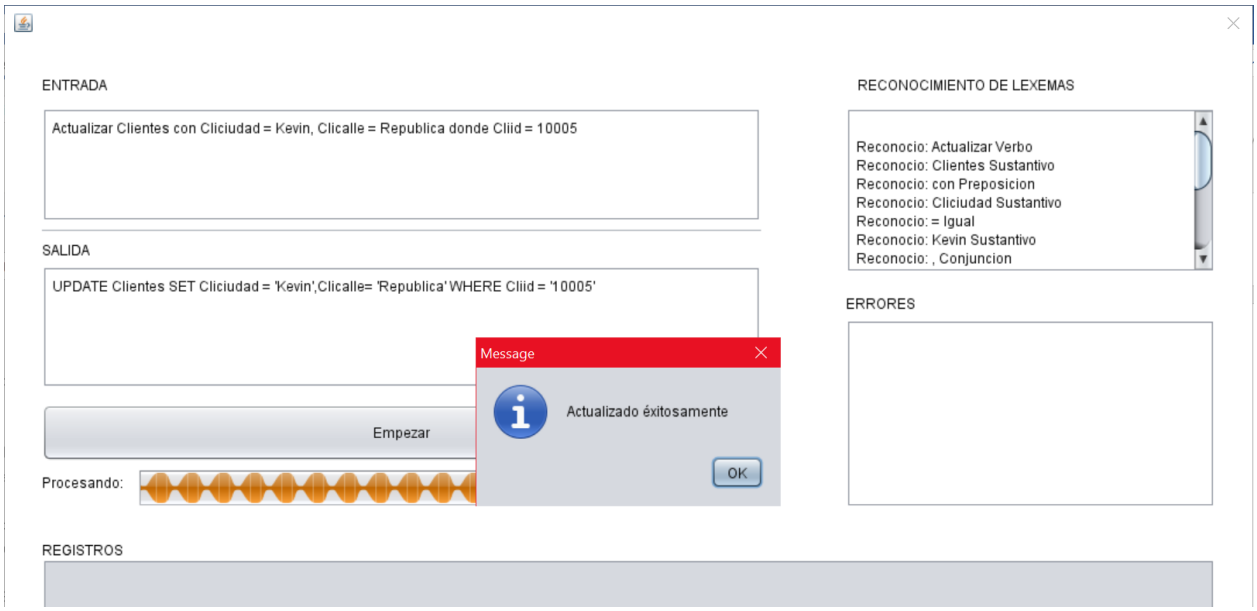

Ilustración 21. Resultado del Intérprete de la Sentencia UPDATE

6.1.4 Pruebas con sentencias delete

Prueba 1:

Oración de Entrada: “Quitar en Clientes donde Cliid = 10005”

Resultado

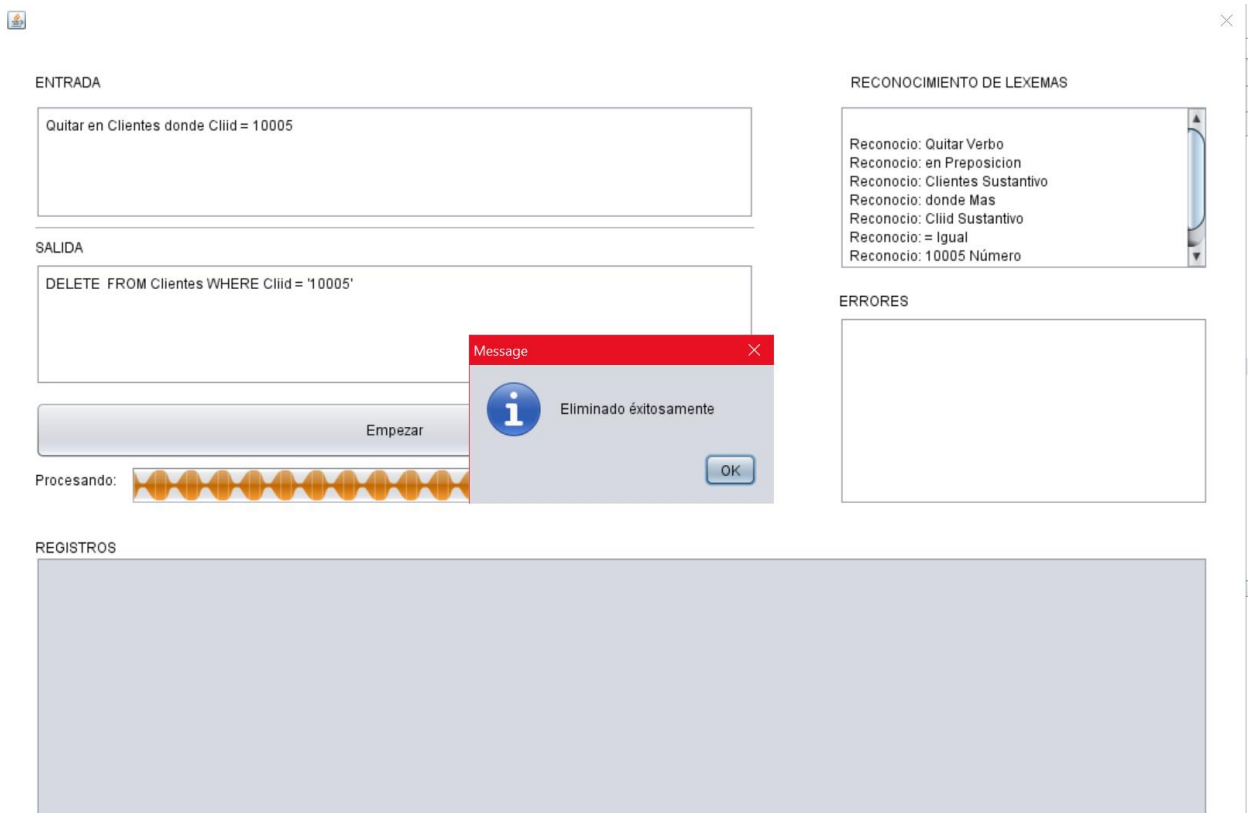

Ilustración 22. Resultado del Intérprete de la Sentencia DELETE

CAPÍTULO 7: CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones

Tras el análisis de las herramientas para la construcción de un intérprete se concluye que JFlex y Cup son las más idóneas para trabajar e implementar en un entorno de desarrollo basado en interfaces graficas. Esto se debe a que ambas están desarrolladas en java y son utilizadas como librerías.

A diferencia de otras herramientas como LIZA, Lex y Yacc; JFlex y Cup generan un resultado simple, el cual es utilizado para las consultas de la base de datos. Además, ambas herramientas presentan una sintaxis simple para su manipulación e interpretación.

La estructura del lenguaje natural escrito presenta una flexibilidad en su construcción, ya que una oración puede presentar diferentes estructuras sin perder sus significado. Por lo que al momento de la utilización del LN se estableció una gramática estática, donde la oración está dividida en dos secciones: la primera presenta el verbo que especifica la acción que se debe realizar con la sentencia SQL y la segunda sección especifica las condiciones con las que se debe realizar la acción.

El intérprete presenta un diseño basado en el análisis léxico y sintáctico. Donde se realizó expresiones regulares para el reconocimiento de lexemas que provienen del LN. Posteriormente, se utilizó una gramática donde se encuentra la estructura válida para que el LN pueda ser transformado en sentencia SQL. Por lo tanto, el intérprete solo permite una estructura específica del LN para su transformación y uso.

Se concluye que el LN tras ser validado por la gramática es transformado a sentencias SQL si los lexemas están dentro del diccionario de sinónimos. Por lo que el intérprete limita la transformación del LN y solo no permite la traducción de sentencias SQL de gran complejidad.

La operabilidad del interprete está dirigido para el lenguaje de manipulación de datos, en este caso: SELECT, INSERT, UPDATE, y DELETE. Los cuales presentan una estructura no tan compleja al solo permitir el uso de ciertos criterios simples de búsqueda como son el where, and, or , entre otros. Por lo que, el usuario podrá realizar solo sentencias SQL básicas.

7.2 Recomendaciones

Este interprete está dirigido para el uso de enseñanza de base de datos a personas sin conocimientos de la sintaxis de SQL. Debido que presenta ciertas limitaciones en la complejidad de formulación de las sentencias SQL.

Al ser un prototipo el sistema no presenta una cobertura extensa del lenguaje natural escrito ya que solo abarca ciertos sinónimos para las palabras reservadas. Por lo que, se recomienda implementar una base de datos que contenga todos los sinónimos y pueda ser utilizada a través del CUP.

Se recomienda seguir explorando el dinamismo que puede presentar las reglas gramaticales, con el objetivo de expandir el reconocimiento y traducción de diferentes oraciones con una estructura diferente a la establecida. Con el objetivo de que el intérprete sea más flexible y presente un proceso de retroalimentación como lo tiene el lenguaje natural.

BIBLIOGRAFÍA

Cortez Vásquez, A., Vega Huerta, H., & Pariona Quispe, J. (2009). *Procesamiento de lenguaje natural.* Obtenido de http://200.62.146.19/bibvirtual/Publicaciones/risi/2009_n2/v6n2/a06v6n2.pdf

Silberschatz, A., Korth, H., & Sudarshan, S. (2011). *DATABASE SYSTEM CONCEPTS, SIXTH EDITION.* Nueva York: McGraw-Hil.

Giordani, A. (2008). *Mapping Natural Language into SQL in a NLIDB.* Obtenido de https://link.springer.com/chapter/10.1007/978-3-540-69858-6_46

Groof, J., & Weinberg, P. (2002). *SQL: The Complete Reference .* Osborne: McGraw-Hill.

Klein, G., Rowe, S., & Décamps, R. (21 de Septiembre de 2018). *JFlex User's Manual.* Obtenido de https://www.jflex.de/manual.html

Pagrut, A., Pakmod, I., Kariya, S., Kamble, V., & Haribhakta, Y. (Junio de 2018). *AUTOMATED SQL QUERY GENERATOR BY UNDERSTANDING A NATURAL LANGUAGE STATEMENT.* Obtenido de http://aircconline.com/ijnlc/V7N3/7318ijnlc01.pdf?fbclid=IwAR292V

Singh, G., & Solanki, A. (1 de Septiembre de 2016). *An algorithm to transform natural language into SQL queries for relational databases.* Obtenido de http://www.iaees.org/publications/journals/selforganizology/articles/2016-3(3)/algorithm-to-transform-natural-language-into-SQL-queries.pdf?fbclid=IwAR1h7G_hZhX5GT_mGpKufGhPJfrqlAa0aGaKWJpr63cDQw8zNQAPO8et26U

Sukthankar, N., Maharnawar, S., Deshmukh, P., Haribhakta, Y., & Kamble, V. (Julio de 2017). *nQuery - A Natural Language Statement to SQL Query Generator.* Obtenido de https://www.aclweb.org/anthology/P17-3004.pdf

Universidad de Málaga. (s.f.). *JavaCUP - Análisis Léxico: Directivas JLex.* Obtenido de Directivas de JLex: http://www.lcc.uma.es/~galvez/theme/cup/anlexcup/especificacion/directivas.html

Viescas, J., & Hernandez, M. (2014). *SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL.* Addison-Wesley Professional.

Codd, E. F. (1990). The Relational Model for Database Management : Version 2. In *Database*. Retrieved from http://books.google.co.kr/books?q=9780201141924

Refaeilzadeh, P., Tang, L., Liu, H., Angeles, L., & Scientist, C. D. (2017). *Cross Validation: Encyclopedia of Database Systems*. 1–2.

https://doi.org/10.1007/978-1-4899-7993-3

Lake, P., & Crowther, P. (2013). Concise Guide to Databases. In *A History of Databases*. https://doi.org/10.1007/978-1-4471-5601-7

Codd, E. F. (1983). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *26*(1), 64–69. https://doi.org/10.1145/357980.358007

Türker, C., & Gertz, M. (2001). Semantic integrity support in sql:1999 and commercial (object-)relational database management systems. *VLDB Journal*, *10*(4), 241–269. https://doi.org/10.1007/s007780100050

In Reply: BEHAVIOUR THERAPY. (1965). In *The British Journal of Psychiatry* (Vol. 111). https://doi.org/10.1192/bjp.111.479.1009-a

Allesina, S., & Wilmes, M. (2019). Relational Databases. *Computing Skills for Biologists*, (11), 337–365. https://doi.org/10.2307/j.ctvc77jrc.15

Nagare, P., Indhe, S., Sabale, D., Thorat, G. P. D. Y., & Chaturvedi, P. G. K. (2017). Automatic SQL Query Formation from Natural Language Query. *International Research Journal of Engineering and Technology(IRJET)*, *4*(5), 1589–1591. Retrieved from https://www.irjet.net/archives/V4/i5/IRJET-V4I5310.pdf

Kovács, L. (2016). SQL generation for natural language interface. *Computer Technology and Computer Programming: New Research and Strategies*, (March), 90–98. https://doi.org/10.1201/b13124-6

LEMONE, K. A. (1996). FUNDAMENTOS DE COMPILADORES; COMO TRADUCIR AL LENGUAJE DE COMPUTADORA (1a. ed.). MEXICO: COMPAÑIA EDITORIAL CONTINENTAL.