MANUAL TECNICO

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

Carnet	Nombre
201503712	Oscar Rene Cuéllar Mancilla

29 DE SEPTIEMBRE DE 2019

Contenido

1. Descripción de la Aplicación y Requerimientos Mínimos 2
1.1 Descripción de la aplicación 2
1.2 Framework Cliente y Servidor 2
1.3 Generador de Analizador Léxico y Sintáctico (Irony) 3
1.4 Lenguaje de desarrollo Servidor (C#) 4
1.5 Generador de Analizador Léxico y Sintáctico (Jison) 5
1.6 Requerimientos Mínimos 5
1.7 Requerimientos mínimos para correr la aplicación 9
2. Diagrama de Clases 10
2.1 Diagrama de clases general CQL 10
2.2 Diagrama de clases general DBMS 10
3. Diagrama de Paquetes 11
3.1 Diagrama de paquetes servidor 11
3.2 Diagrama de paquetes Cliente 12
4. Descripción de Herramientas Utilizadas 13
4.1 CodeMirror 13
4.2 BlockLY 14
5. Descripción de clases y métodos principales 15
5.1 Rutas.asmx 15
5.2 Management.cs 16
5.3 AST_CQL.cs 16
6. Explicación de acciones semánticas 17
6.1 Acciones Semánticas gramática CQL 17
6.2 Acciones semánticas gramática Jison 20

1. Descripción de la Aplicación y Requerimientos Mínimos

1.1 Descripción de la aplicación

CQL-Teacher es una aplicación cliente–servidor, que permitirá interactuar con los usuarios para que puedan familiarizarse con el uso y beneficios de una base de datos, de una forma bastante fácil e intuitiva, proveerá bastantes modos para que así puedan trabajar de acuerdo con su nivel de conocimiento. Además, el sistema de base de datos también contará con instrucciones y elementos basados de un lenguaje estructurado, tales como variables, ciclos, sentencias condicionales y tipos de datos definidos por el usuario.

La aplicación cliente, CQL-Client, es una aplicación web que permitirá al usuario interactuar con el servidor de bases de datos, esta contará con una serie de componentes que le brindan al usuario una experiencia agradable al momento de utilizarlo. Tiene 3 distintos modos de dificultad para la edición de código de alto nivel, así como un login para garantizar la seguridad de la información.

1.2 Framework Cliente y Servidor

ASP.NET es un entorno para aplicaciones web desarrollado y comercializado por Microsoft. Es usado por programadores y diseñadores para construir sitios web dinámicos, aplicaciones web y servicios web XML. Apareció en enero de 2002 con la versión 1.0 del .NET Framework, y es la tecnología sucesora de la tecnología Active Server Pages (ASP). ASP.NET está construido sobre el Common Language Runtime, permitiendo a los programadores escribir código ASP.NET usando cualquier lenguaje admitido por el .NET Framework.

1.3 Generador de Analizador Léxico y Sintáctico (Irony)

Irony es un kit de desarrollo para la aplicación de las lenguas en la plataforma NET. A diferencia de la mayoría de soluciones yacc/lex-style irony no utiliza ningún escáner o el programa de análisis de generación de código a partir de especificaciones de gramática escritos en una meta-lenguaje especializada.

En Irony la gramática idioma de destino se codifica directamente en C # utilizando la sobrecarga de operadores para expresar las construcciones gramaticales. Módulos del escáner y analizador de irony utilizan la gramática codificado como c # clase para controlar el proceso de análisis.

Irony es un analizador que se caracteriza por ser uno de los pocos, o porque no decir el único que funciona en base a la generación del AST (Abstract Syntax Tree ó Árbol de Sintaxis Abstracta); El AST tiene la característica primordial, en que es la base fundamental para la ejecución de código, eso quiere decir que por medio del mismo podemos llevar el control de todo lo que hayamos generado mediante la gramática, pero tomando en cuenta que la generación del árbol implica únicamente la generación, ejecución (esto se debe implementar posteriormente por el programador) pero nunca la modificación del árbol, ya que en base a este árbol, se generará la tabla de símbolos, la cual es la que si debe ser modificable.

El usuario quien implementa una gramática con Irony debe tomar en cuenta que la implementación de la misma puede ser ambigua, re-cursiva por la izquierda y demás, ya que es un analizador ascendente, pero si se aplica ambigüedad en la gramática, existe uno de sus tantos métodos llamado precedence, el cual evalúa la precedencia que le indiquemos para realizar las operaciones necesarias.

1.4 Lenguaje de desarrollo Servidor (C#)

C# (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por el signo #, el cual se lee como sharp en inglés para notación musical. Es un juego de palabras, pues "'C#" significa, musicalmente hablando, "do sostenido", donde el símbolo # indica que una nota (en este caso do, representada por C) debe ser un semitono más alta. Esto es una metáfora de la superioridad de C# sobre su antecesor C++ y a su vez hace alusión a la misma metáfora que se ideó para dar nombre a C++.1 Además, el símbolo # puede ser imaginado como la unión de cuatro símbolos +, continuando así con el sentido de progresión de los lenguajes C.

1.5 Generador de Analizador Léxico y Sintáctico (Jison)

Jison toma una gramática libre de contexto como entrada y genera un archivo JavaScript capaz de analizar el lenguaje descrito por esa gramática. Luego puede usar el script generado para analizar entradas y aceptar, rechazar o realizar acciones basadas en la entrada. Si estás familiarizado con Bison o Yacc, u otros clones, estás casi listo para rodar.

1.6 Requerimientos Mínimos

Los requerimientos mínimos del proyecto son funcionalidades del sistema que permitirán un ciclo de ejecución básica.

2 Descripción

- 2.1.5 Login
- 2.1.6 Panel de información
- 2.1.7 Salida de datos y Consola
- 2.1.8 Modos de edición
- 2.1.8.1 Principiante
- 2.1.8.2 Avanzado

2.4 Definiciones generales de CQL

- 2.4.1 Identificadores
- 2.4.2 Case Insensitive
- 2.4.3 Comentarios
- 2.4.4 Valor nulo

2.5 Tipos de dato

2.5.1 Tipos primitivos

2.5.2 Secuencias de escape

2.5.3 User Types

2.5.4 Casteos primitivos en CQL

3 Lenguaje de Definición de Datos (DDL).

3.1 Crear Base de datos

3.2 Sentencia Use 3.4 Crear Tabla

3.4.1 Tipo Dato Counter

3.4.2 Llaves Primarias

3.4.3 Tipos de datos para definición de columnas

3.5 Alter Table

4 Lenguaje de Control de Transacciones (TCL).

4.1 Commit

4.2 Rollback

5 Lenguaje de Control de Datos (DCL).

5.1 Crear Usuario

6 Lenguaje de Manipulación de datos (DML).

6.1 Insertar

6.2 Actualizar

6.4 Seleccionar

6.6 Funciones de Agregación

6.7 Clausula Where

6.7.1 Operaciones Lógicas

6.8 Uso de colecciones y objetos como componentes de una tabla

6.8.3 Listas

6.8.4 User Types

7 Lenguaje de Control de Flujo (FCL)

7.2 Sintaxis

7.2.3 Variables

7.2.4 Declaración de variables

7.2.5 Asignación de variables

7.2.6 Valores predeterminados para las variables

7.2.7 Operadores aritméticos

7.2.8 Operadores relacionales

7.2.9 Operaciones lógicas

7.2.12 Sentencias de Selección

7.2.12.1 Sentencia If

7.2.12.2 Sentencia else

7.2.13 Sentencias cíclicas o bucles

7.2.13.1 Sentencia While

7.2.13.2 Sentencia For

7.2.14 Collections

7.12.14.1 Lists

7.2.15 Funciones

7.2.16 Procedimientos

7.2.19 Sentencias de transferencia

7.2.19.1 Break

7.2.19.2 Return

7.2.20 Cursores

7.2.21 Log

8 Lenguaje Unificado de Paquetes (LUP)

8.1 Solicitudes del cliente al servidor:

8.1.1 Paquete de Inicio de Sesión

8.1.2 Paquete de Fin de Sesión

8.1.3 Paquete de Consulta:

8.2 Respuestas del servidor al cliente:

8.2.1 Paquete de Datos

8.2.2 Paquete de Error

8.2.3 Paquete de Mensaje

8.2.4 Paquete de Estructura

9 El Lenguaje de Almacenamiento de Datos (CHISON)

9.1 Notación

9.1.1 Datos Primitivos

9.2 Definición de los archivos

9.2.1 Archivo Principal

9.3 Importar Archivo

10 Excepciones

11 Apéndice A: Precedencia y asociatividad de operadores

12 Manejo de errores

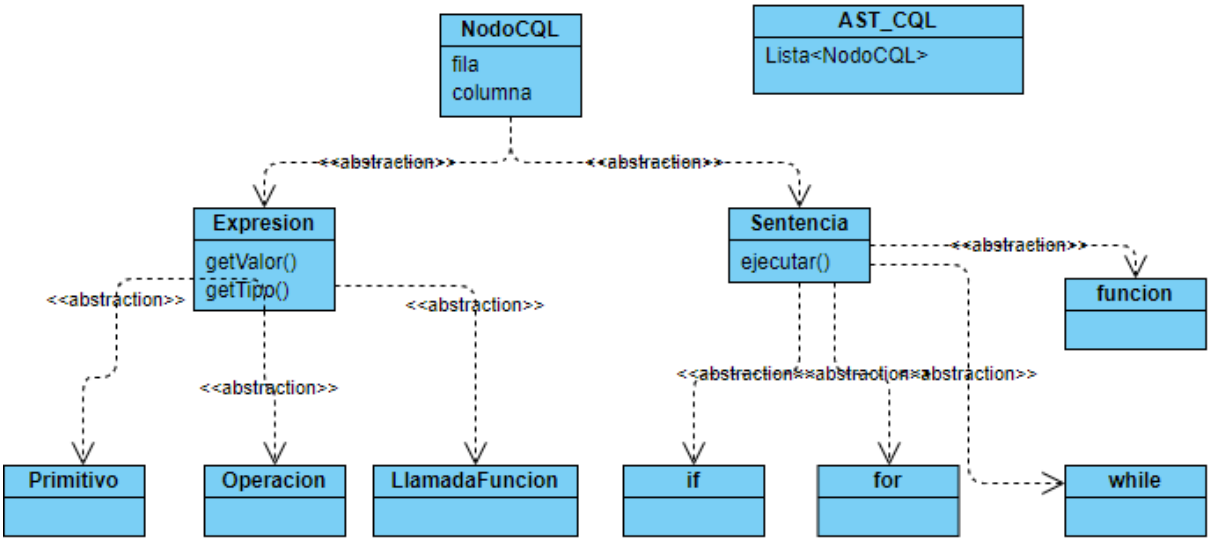
1.7 Requerimientos mínimos para correr la aplicación

Para poder ejecutar dicha aplicación se necesitan tener instaladas las siguientes herramientas y frameworks.

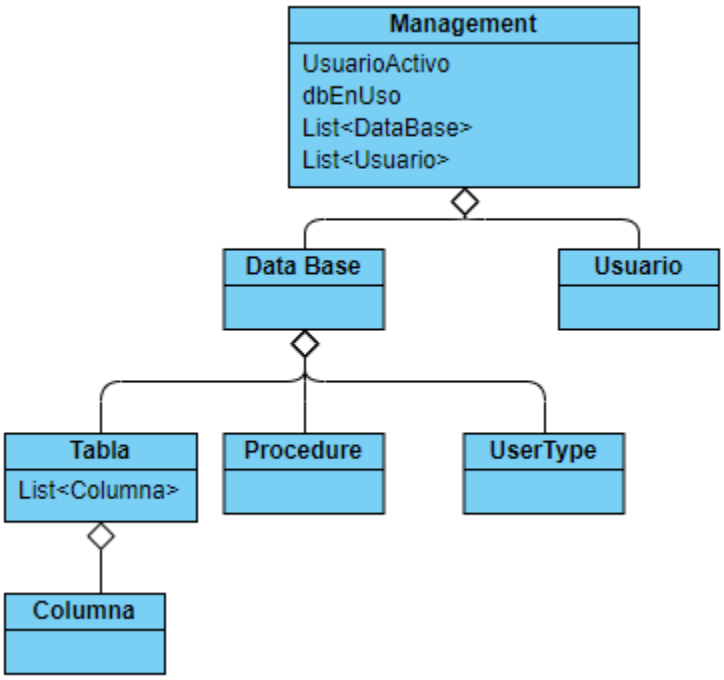
- Internet Information Services (IIS) Microsoft
- Framework .Net 4.0 o superior
- Navegador Google Chrome o Microsoft Edge

2. Diagrama de Clases

2.1 Diagrama de clases general CQL

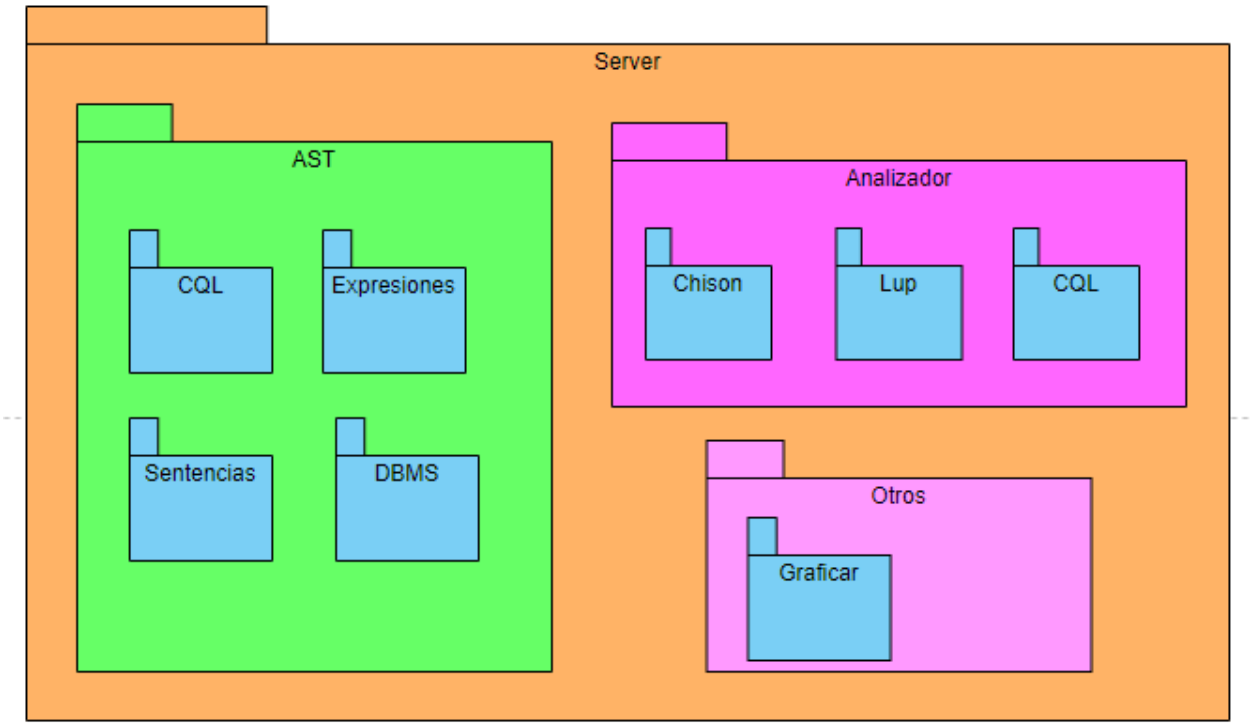


2.2 Diagrama de clases general DBMS

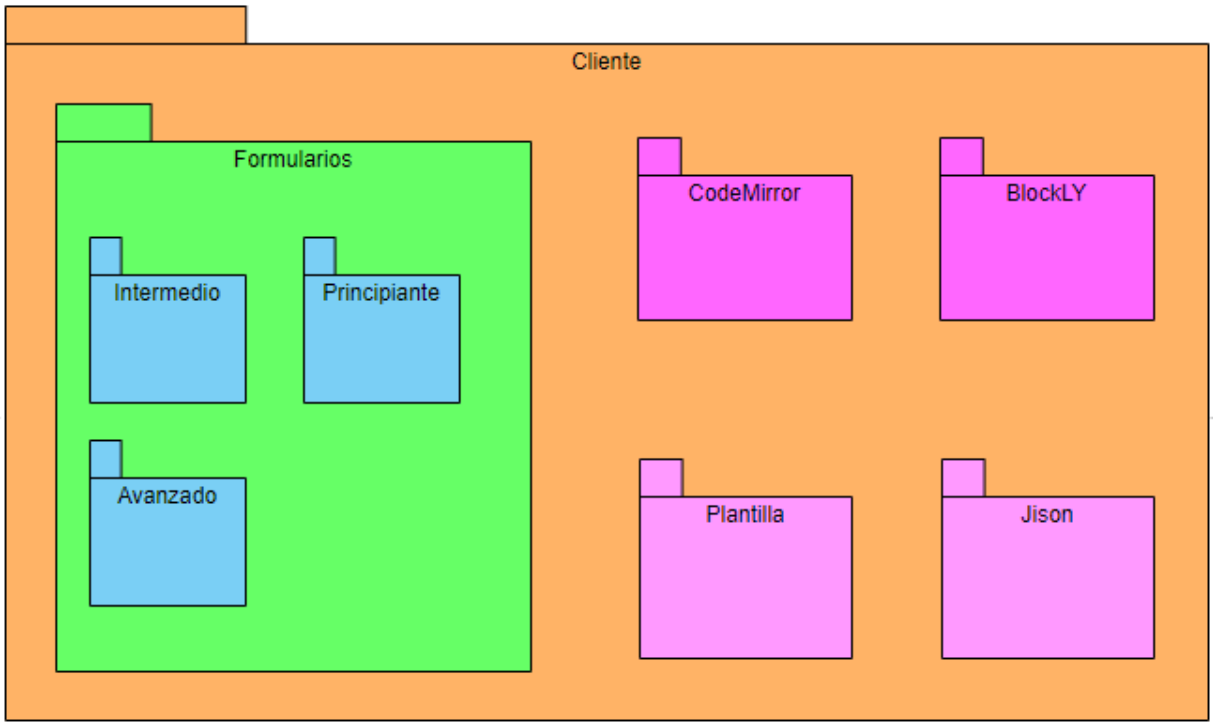


3. Diagrama de Paquetes

3.1 Diagrama de paquetes servidor



3.2 Diagrama de paquetes Cliente



4. Descripción de Herramientas Utilizadas

4.1 CodeMirror

CodeMirror es un editor de texto versátil implementado en JavaScript para el navegador. Está especializado para editar código y viene con varios modos de lenguaje y complementos que implementan una funcionalidad de edición más avanzada.

Una rica API de programación y un sistema de temas CSS están disponibles para personalizar CodeMirror para que se ajuste a su aplicación y ampliarla con nuevas funciones.

Características:

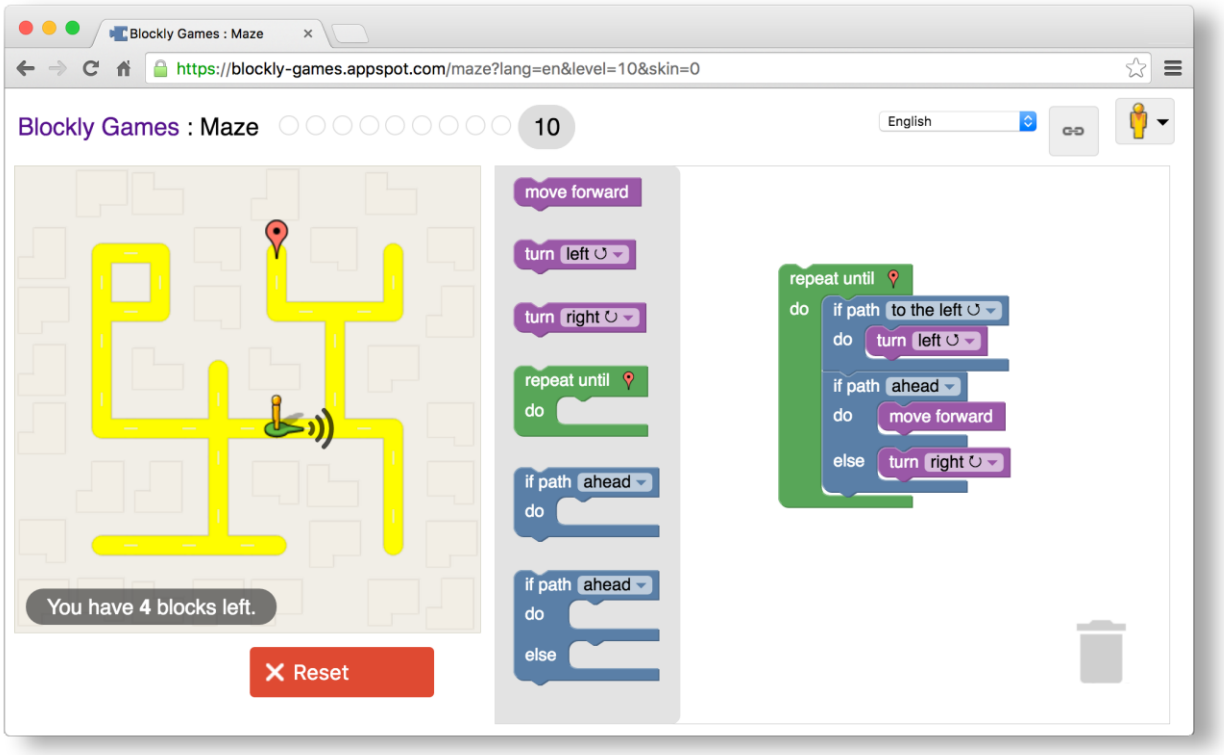
- Soporte para más de 100 idiomas listos para usar
- Un sistema de modo de lenguaje potente y composable
- Autocompletado (XML)
- Código plegable
- Combinaciones de teclas configurables
- Enlaces de Vim , Emacs y Sublime Text
- Buscar y reemplazar interfaz
- Soporte y etiqueta a juego
- Soporte para vistas divididas
- Integración de Linter
- Mezclando tamaños de fuente y estilos
- Varios temas
- Capaz de cambiar el tamaño para adaptarse al contenido
- Widgets en línea y en bloque
- Canaletas programables
- Hacer rangos de texto con estilo, solo lectura o atómico
- Soporte de texto bidireccional
- Muchos otros métodos y complementos.

4.2 BlockLY

Blockly en un navegador permite que las páginas web incluyan un editor de código visual para cualquiera de los cinco lenguajes de programación compatibles de Blockly, o el suyo.

Blockly es:

- Biblioteca de JavaScript puro. Menos de 150 kb sobre el cable.
- 100% del lado del cliente. Sin dependencias del lado del servidor.
- Compatible con todos los principales navegadores: Chrome, Firefox, Safari, Opera e IE.
- Altamente personalizable y extensible.

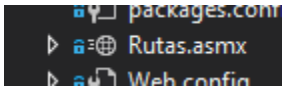


Referencia: https://developers.google.com/blockly

5. Descripción de clases y métodos principales

5.1 Rutas.asmx

El archivo ASMX es uno de archivos de la categoría Archivos de Internet. Su nombre completo es ASP.NET Web Service Format.



Este es el archivo encargado de almacenar los métodos REST que son consumidos por nuestro cliente.

```
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Script.Serialization;
using System.Threading;
using Server.Analizador.LUP;
using Server.AST.DBMS;

namespace Server
{
    /// <summary>
    /// Descripción breve de Rutas
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // Para permitir que se llame a este servicio web desde un script, usando ASP.NET AJAX, quite la marca de comentario de la línea siguiente.
    // [System.Web.Script.Services.ScriptService]
    public class Rutas : System.Web.Services.WebService
    {
        public static Management dbms;

        public Rutas() {
            if (dbms==null)
            {
                dbms = new Management();
                dbms.analizarChison("");
            }
        }
```

5.2 Management.cs

Este archivo es el encargado de manejar todo lo relacionado con la base de datos, ejecutar instrucciones DML, DDL, etc. Es el corazón de la base de datos.

```
namespace Server.AST.DBMS
{
    public class Management
    {
        public DataBase system;
        public User usuarioActivo { get; set; }
        List<User> users;
        public List<DataBase> bases;
        public List<clsToken> errores;

        public Management() {
            this.bases = new List<DataBase>();
            this.users = new List<User>();
            this.errores = new List<clsToken>();
        }
```

5.3 AST_CQL.cs

En esta clase se encuentra todo lo necesario para realizar la ejecución de las instrucciones CQL.

```
namespace Server.AST
{
    public class AST_CQL
    {
        public List<NodoCQL> nodos { get; set; }
        public List<String> mensajes { get; set; }
        private List<List<ColumnCQL>> res_consultas { get; set; }
        public List<String> result_consultas { get; set; }
        public List<clsToken> errores { get; set; }
        public Entorno entorno { get; set; }
        public List<Funcion> funciones { get; set; }
        public Boolean finalizado { get; set; }
        public Management dbms { get; set; }

        public AST_CQL() {
            this.funciones = new List<Funcion>();
            this.nodos = new List<NodoCQL>();
            this.mensajes = new List<String>();
            this.res_consultas = new List<List<ColumnCQL>>();
            this.result_consultas = new List<string>();
            this.errores = new List<clsToken>();
            this.entorno = new Entorno(null);
            this.finalizado = false;
            this.dbms = new Management();
        }
```

6. Explicación de acciones semánticas

6.1 Acciones Semánticas gramática CQL

```
else if (CompararNombre(raiz, "GLOBAL"))
{
    return recorrido(raiz.ChildNodes[0]);
}
else if (CompararNombre(raiz, "DECLARACION"))
{
    return new Declaracion(recorrido(raiz.ChildNodes[0]), (List<KeyValuePair<String, Expresion>>)recorrido(raiz.ChildNodes[1]), getFi
}
else if (CompararNombre(raiz, "LISTA_DECLARACION_E"))
{
    List<KeyValuePair<String, Expresion>> lista = new List<KeyValuePair<String, Expresion>>();
    foreach (ParseTreeNode nodo in raiz.ChildNodes)
    {
        lista.Add((KeyValuePair<String, Expresion>)recorrido(nodo));
    }
    return lista;
}
else if (CompararNombre(raiz, "KEY_VALUE_LIST"))
{
    List<KeyValuePair<Expresion, Expresion>> lista = new List<KeyValuePair<Expresion, Expresion>>();
    foreach (ParseTreeNode nodo in raiz.ChildNodes)
    {
        lista.Add((KeyValuePair<Expresion, Expresion>)recorrido(nodo));
    }
    return lista;
}
else if (CompararNombre(raiz, "LISTA_CQLTIPOS"))
{
    List<KeyValuePair<String, Object>> list = new List<KeyValuePair<string, object>>();
    foreach (ParseTreeNode nodo in raiz.ChildNodes)
    {
        list.Add((KeyValuePair<String, Object>)recorrido(nodo));
    }
    return list;
}
```

```
if (ContainsString(getLexema(raiz, 0), "create") && ContainsString(getLexema(raiz, 1), "database"))
{
    return new CreateDataBase(Convert.ToBoolean(recorrido(raiz.ChildNodes[2])), getLexema(raiz, 3),
        getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "drop") && ContainsString(getLexema(raiz, 1), "database"))
{
    return new DropDataBase(getLexema(raiz, 2), getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "drop") && ContainsString(getLexema(raiz, 1), "table"))
{
    return new DropTable(Convert.ToBoolean(recorrido(raiz.ChildNodes[2])), getLexema(raiz, 3),
        getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "create") && ContainsString(getLexema(raiz, 1), "table"))
{
    return new CreateTable(Convert.ToBoolean(recorrido(raiz.ChildNodes[2])), getLexema(raiz, 3),
        (List<ColumnCQL>)recorrido(raiz.ChildNodes[5]), getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "truncate") && ContainsString(getLexema(raiz, 1), "table"))
{
    return new TruncateTable(getLexema(raiz, 2), getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "use"))
{
    return new UseDataBase(getLexema(raiz, 1), getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "alter") && ContainsString(getLexema(raiz, 3), "add"))
{
    return new AlterTableAdd(getLexema(raiz, 2), (List<ColumnCQL>)recorrido(raiz.ChildNodes[4])
        , getFila(raiz, 0), getColumna(raiz, 0));
}
else if (ContainsString(getLexema(raiz, 0), "alter") && ContainsString(getLexema(raiz, 3), "drop"))
{
    return new AlterTableDrop(getLexema(raiz, 2), (List<String>)recorrido(raiz.ChildNodes[4])
        , getFila(raiz, 0), getColumna(raiz, 0));
}
```

```
else if (CompararNombre(raiz, "LIMIT_Q"))
{
    if (raiz.ChildNodes.Count == 0)
    {
        return null;
    }
    else
    {
        return recorrido(raiz.ChildNodes[0]);
    }
}
else if (CompararNombre(raiz, "LIMIT"))
{
    //res_limit + E;
    return (Expresion)recorrido(raiz.ChildNodes[1]);
}
else if (CompararNombre(raiz, "ORDER"))
{
    /* id
    | id + res_desc
    | id + res_asc;*/
    if (raiz.ChildNodes.Count == 2 && getLexema(raiz, 1).Equals("desc"))
    {
        return new Order(getLexema(raiz, 0), Order.ORDER.DESC);
    }
    else if (raiz.ChildNodes.Count == 2 && getLexema(raiz, 1).Equals("asc"))
    {
        return new Order(getLexema(raiz, 0), Order.ORDER.ASC);
    }
```

Como se puede observar anteriormente se muestra la generación del árbol de análisis semántico para las acciones CQL.

6.2 Acciones semánticas gramática Jison

Para poder realizar el análisis de la información obtenida como respuesta del servidor, se utilizaron las siguientes estructuras en Javascript dentro del analizador de Jison.

```
%{
    function TokenError(){
        this.lexema = "";
        this.descripcion = "";
        this.fila = "";
        this.columna = "";
        this.tipo = "";
    }

    function AST_LUP(){
        this.login = false;
        this.logout = false;
        this.contMess = 0;
        this.contData = 0;
        this.contErr = 0;
        this.contDBMS = 0;
        this.data = {};
        this.errores = {};
        this.mensajes = {};
        this.dbms = new Estruct();
    }

    function Estruct(){
        this.contProcedures = 0;
        this.contAtrs = 0;
        this.contColumns = 0;
        this.contTypes = 0;
        this.contTab = 0;
        this.contDataBase = 0;
        this.dataBases = {};
    }

    var ast = new AST_LUP();
%}
```