CONSTRUCCIÓN DE LOS COMPONENTES ODL Y OQL PARA UN SISTEMA GENERADOR DE BASES DE DATOS REORIENTADO A OBJETOS (SGBDRO)

FRANCISCO FARITH CONTRERAS HERAZO
ARTURO SEGUNDO GARCIA PAYARES
JAVIER NORBERTO CALA URIBE

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO – MECÁNICA
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2008

CONSTRUCCIÓN DE LOS COMPONENTES ODL Y OQL PARA UN SISTEMA GENERADOR DE BASES DE DATOS REORIENTADO A OBJETOS (SGBDRO)

FRANCISCO FARITH CONTRERAS HERAZO
ARTURO SEGUNDO GARCIA PAYARES
JAVIER NORBERTO CALA URIBE

Trabajo de grado para optar al título de
Ingeniero de Sistemas

Director
JAIME OCTAVIO ALBARRACIN FERREIRA
Doctor en Informática

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO – MECÁNICA
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2008

Nota de Aceptación

_____
_____
_____
_____
_____

A Dios por todas las oportunidades que me ha dado en la vida.
A mí querida madre que siempre me apoyó y confió en mí, que siempre estuvo y estará acompañándome.
A mi padre, por su sacrificio, esfuerzo y por ser un pilar muy importante en mi vida.
A mis hermanos, quienes siempre estuvieron a mi lado a pesar de la distancia,
por todas las alegrías que me han dado.
A toda mi familia, por su respaldo incondicional,
por ser el motor de mi vida y por inspirarme a seguir siempre adelante.
A mis amigos, por su compañía y afecto, por las risas y los buenos momentos.
Y a mis socios Arturo y Javier, que la vida los colme de felicidad.

FRANCISCO

A Ludys y Arturo, mis padres, quienes me han demostrado
que con amor y dedicación se pueden hacer posibles los imposibles.
A mis hermanos por su apoyo y total confianza en mí.
A Javier y Francisco para que al igual que ésta,
logren alcanzar sus metas muy a pesar de los obstáculos.
A todos mis amigos por los buenos momentos

ARTURO

A Dios por todo.
A mis padres por su amor.
A mis hermanas por su apoyo condicional e incondicional.
A mis tíos y tías por sus consejos.
A mi familia en general por ser tan particular.
A mis amig@s por su conexión :)
A Arturo Francisco por un excelente trabajo.

JAVIER

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

UNIVERSIDAD INDUSTRIAL DE SANTANDER

JAIME OCTAVIO ALBARRACIN FERREIRA. Doctor en Informática (Universidad de Oviedo – España –) Director del proyecto. Por su constante seguimiento en el desarrollo del proyecto y su interés en la calidad del mismo.

JOSÉ DUVAN MÁRQUEZ DÍAZ. Ingeniero de Sistemas y director del Departamento de Ingeniería de Sistemas de la Universidad del Norte. Por su colaboración y asesoría a lo largo del proyecto.

SCOTT HUDSON. HCI Institute, School of Computer Science, Carnegie Mellon University. Por su asesoria en el desarrollo del analizador sintáctico.

ANDREA FLEXEDER. Technische Universitaet Manchen, Institut für Informatik. Por su colaboración a lo largo de la implementación de las estructuras de archivos para la jerarquía de clases.

A todas aquellas personas que de una u otra manera contribuyeron a la realización de este proyecto.

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN

TITULO: CONSTRUCCIÓN DE LOS COMPONENTES ODL Y OQL PARA UN SISTEMA GENERADOR DE BASES DE DATOS REORIENTADO A OBJETOS (SGBDRO)*

AUTORES: FRANCISCO FARITH CONTRERAS HERAZO
ARTURO SEGUNDO GARCIA PAYARES
JAVIER NORBERTO CALA URIBE**

PALABRAS CLAVES: MODRO, SGBDRO, Interprete, Análisis, Sintáctico, Léxico

DESCRIPCION:

Las bases de datos reorientadas a objetos (BDRO) son la implementación del MODRO o modelo de datos reorientado a objetos, presentado como tesis doctoral por el profesor Jaime Albarracín en la Universidad de Oviedo (España). La designación de reorientado a objetos nace al concebir sistemas auto-contenidos en vez de objetos auto-contenidos, como es tradicional, entendiendo que los sistemas complejos de grano grueso están compuestos de objetos elementales de grano fino.

Con dicho MODRO se busca resolver la desintegración sufrida al seno de la organización en general. Sin embargo, debido a que no existía hasta el momento un lenguaje adecuado para implementar tal MODRO, era necesario construir un nuevo lenguaje capaz de describir, generar y manipular bases de datos derivadas de MODROs específicos, el cual se ha denominado LATIN por su acrónimo en inglés Language Toward Integration, puesto que en efecto este nuevo modelo fue concebido inicialmente como una solución a la desintegración que sufren hoy las organizaciones.

A este nuevo lenguaje se le ha desarrollado un interprete para los componentes de definición y consulta de objetos (ODL y OQL) que actuarán como el embrión inicial desde el cual se desarrollará en sucesivas investigaciones el cuerpo completo del LATIN, convirtiéndolo a largo plazo en un motor de BDRO, como lo es hoy Oracle o MySQL bajo el modelo relacional.

* Trabajo de Grado.
** Facultad de Ingenierías Físico – Mecánicas, Ingeniería de Sistemas e Informática. Doctor en Informática Jaime Octavio Albarracín Ferreira.

ABSTRACT

TITLE: CONSTRUCTION OF COMPONENTS ODL AND OQL FOR AN OBJECT REORIENTED DATABASES MANAGER SYSTEM (ORDBMS).*

AUTHORS: FRANCISCO FARITH CONTRERAS HERAZO
ARTURO SEGUNDO GARCIA PAYARES
JAVIER NORBERTO CALA URIBE**

KEY WORDS: MODRO, Parsers, Lexer, Databases.

DESCRIPTION:

The object re-oriented databases (ORDB) are the implementation of MODRO or data modeling reoriented to objects, presented like doctoral thesis by professor Jaime Albarracín in the University of Oviedo (Spain). The reoriented to objects designation borned when conceiving systems contained themselves instead of objects contained themselves, like is traditional, understanding that the complex heavy grain systems are compound of elementary weak grain objects.

This MODRO is used to solve the disintegration problem in the sine of the organization in general. Nevertheless, because an suitable language did not exist until the moment to implement such MODRO, it was necessary to construct a new language able to describe, to generate and to manipulate especifics data bases derived from MODROs, which has denominated LATIN by his acronimo in English Language Toward Integration, since in effect this new model was conceived initially like a solution to the disintegration in the organizations today.

To this new language it has been developed an interpreter for the components of definition and query of objects (ODL and OQL) that will act like the initial work to complete the LATIN developing in successive research works, turning it in the long term a ORDBMS, as it is today Oracle or MySQL under the relational model.

* Degree Work.
** Faculty of Physical-Mechanical Engineering, System Engineer. Dr. Jaime Octavio Albarracín Ferreira.

INTRODUCCION

Tras una larga investigación de más de veinte años, en el espacio de varias empresas colombianas, así como en la Universidad Industrial de Santander (UIS) y en la Universidad de Oviedo (España), el profesor Albarracín ha desarrollado como tesis doctoral en ésta última, un modelo de datos dentro de un paradigma diferente, por lo que se le ha llamado Modelo de Datos Reorientado a Objetos o MODRO.

Con dicho MODRO se pretende resolver la desintegración de la Organización en general, ofreciendo simultáneamente una alternativa para las Bases de Datos Orientadas a Objetos, las cuales continúan generando muchas expectativas, insatisfechas todavía.

El desarrollo de dicha alternativa implica el análisis, diseño e implementación de un intérprete capaz de describir, generar y manipular bases de datos derivadas de MODROs específicos. Por esta razón fue necesario construir primero los componentes de definición, consulta y manipulación de objetos (ODL y OQL)[1], los cuales son generados por una gramática libre de contexto la cual esta basada en la sintaxis de SQL y en los lenguajes de programación orientados a objetos. Dicha gramática genera un lenguaje capaz de interpretar el MODRO permitiendo la definición, consulta y manipulación de clases.

También se implementaron los análisis léxico y sintáctico para esta gramática. Todo intérprete requiere de estos analizadores para su funcionamiento ya que estos verifican que las cadenas digitadas por el usuario sean válidas. El análisis léxico se encarga de leer los caracteres de entrada y clasificarlos según la gramática, para luego pasarlos al analizador sintáctico el cual comprueba la secuencia de estos caracteres.

La definición de los archivos de jerarquías de clases se realizo creando archivos de texto plano con extensión LTC (Latin Class) etiquetando las entidades y el método pertenecientes a la clase. Por último se realiza un enlace con un motor de bases de datos el cual se encarga de gestionar los datos. Éste enlace se realiza en esta primera fase del SGBDRO para probar el funcionamiento del intérprete basado en los componentes ODL y OQL, posteriormente será reemplazado por un gestor de bases de datos propio.

Una vez terminado el trabajo anterior se procedió a efectuar las pruebas de los componentes, así como también las pruebas del software en general. Estas

[1] Object Definition Languaje (Lenguaje de Definición de Objetos)
Object Query Language (Lenguaje de Consulta de Objetos)

pruebas permitieron corroborar el trabajo realizado y corregir las fallas presentadas. Por último se implementó un MODRO utilizando dos ejemplos desarrollados en la tesis doctoral del profesor Albarracin, los cuales probaron los diferentes componentes del intérprete desarrollado.

Este trabajo de grado esta dividido en tres capítulos. En el primer capitulo se describe la gramática desarrollada para los componentes ODL y OQL, los análisis realizados, la creación de los archivos y el enlace con el motor de bases de datos. El segundo capitulo explica las pruebas realizadas a los componentes y al software en general. En el último capítulo se explica el uso de un MODRO específico. Al final del documento se presentan las conclusiones, recomendaciones, bibliografía utilizada y los anexos.

1. DESARROLLO DE LOS ANALIZADORES Y ESTRUCTURA DE DATOS DEL SISTEMA GENERADOR DE BASES DE DATOS REORIENTADO A OBJETOS (SGBDRO)

Para el desarrollo de los componentes ODL y OQL fue necesario definir una gramática libre de contexto la cual se encarga de generar el lenguaje del MODRO. Una vez obtenida la gramática se procedió a realizar los analizadores léxico y sintáctico encargados de verificar las entradas del usuario. A partir del análisis sintáctico se realiza una traducción dirigida por sintaxis para la creación de los archivos de jerarquía de clases así como también de un archivo encargado del enlace con el motor de bases de datos. El motor de almacenamiento utilizado es InnoDB del sistema de bases de datos MySQL.

1.1. DEFINICION DE UNA GRAMATICA LIBRE DE CONTEXTO PARA LOS COMPONENTES ODL Y OQL

Un nuevo modelo de datos trae consigo la generación de un lenguaje que pueda interpretar dicho modelo. El MODRO requiere de los componentes de definición, manipulación y consulta de objetos (ODL y OQL). Por tanto es menester definir una gramática libre de contexto para este modelo capaz de generar el ODL y OQL.

Para la definición de la gramática se utilizo el sistema formal “BNF” o Forma de Backus-Naur. En el desarrollo de la gramática para los componentes de definición, manipulación y consulta de objetos se tuvo en cuenta la sintaxis de SQL y la empleada por lenguajes de programación orientados a objetos como Java.

A continuación se describe la estructura del lenguaje implementado para los componentes. (Para más información sobre la gramática consultar el Anexo A y B).

1.1.1. Valores Literales

- **Variables**: Una variable comienza con una letra y puede estar seguida de uno o más caracteres alfa-numéricos. También se acepta el carácter “_”.

  Variable = [A-Za-z_][A-Za-z_0-9]*

  Ejemplo
  nombreClase, parametro_1, _variable2

- **Números**: Los enteros se representan como secuencias de dígitos. Los flotantes utilizan "." como separador decimal.

  Numero = 0
  | [1-9][0-9]*
  | [0-9]+"."[0-9]*
  | "."[0-9]*

  Ejemplo
  123, 0.5, 5, .99

- **Fecha**: El rango soportado es de '1000-01-01' a '9999-12-31'. El formato es "YYYY-MM-DD"

  Fecha = [1-9][0-9][0-9][0-9]"-"[0-1][0-9]"-"[0-3][0-9]

  Ejemplo
  2008-03-15, 1985-09-21

- **Hora**: El rango soportado es de '00:00:00' a '23:59:59'. El formato es "HH:MM:SS"

  Fecha = [0-2][0-9]":"[0-5][0-9]":"[0-5][0-9]

  Ejemplo
  01:55:21, 14:03:15

- **Valores NULL:** El valor null significa "no hay dato". Solo se debe escribir null en minúscula, no se aceptan combinaciones de mayúscula y minúscula.

1.1.2. Caracteres permitidos

;	**+**	**-**
*****	**/**	**(**
)	**{**	**}**
,	**.**	**:**
'	**&**	**=**
<	**>**	

Tabla No 1. Caracteres Permitidos

1.1.3. Palabras reservadas

all	and	any	avg
autoincrement	between	by	cascade
count	check	class	date
decimal	default	delete	distinct
double	drop	entity	escape
exists	extend	float	foreign
from	group	having	in
indicator	insert	integer	into
is	key	like	method
min	max	not	null
numeric	on	or	precision
primary	real	references	select
set	smallint	some	show
sum	table	time	unique
update	values	varchar	where

Tabla No 2. Palabras Reservadas

Estas palabras reservadas no aceptan combinación de mayúsculas y minúsculas, solo son aceptadas en minúscula.

1.1.4. Comentarios

Los comentarios son aceptados desde una secuencia '/*' hasta la próxima secuencia '*/'. La secuencia de cierre no necesita estar en la misma línea, lo que permite tener comentarios que abarquen múltiples líneas. También se permiten comentarios por línea de tipo “// “

1.1.5. Sintaxis iniciales

Solo se acepta un comando de entrada por vez y siempre deben terminar en “;”. La sintaxis de entrada pueden ser para:

- Definición de clases
- Manipulación y consulta de entidades
- Consulta de clases
- Ver gráfico de una clase

1.1.6. Sintaxis de sentencias ODL

- **Definición de Clase**

```
class nombre_clase [herencia][cuerpo_clase];

herencia  :
          /*vacio */
          |extend nombre_clase_padre

cuerpo_clase :
               {
               [definicion_entidad]
               [definicion_metodo]
               }
```

class *nombre_clase* … crea una nueva clase. Esta clase puede heredar las entidades y el método de otra clase con el comando **extend** *nombre_clase_padre*.

nombre_clase, y *nombre_clase_padre* deben ser literales de tipo variable.

- **Definición de Entidades**

```
definicion_entidad  :/* vacio */
                    |entity nombre_entidad
                    {
                    [listado_atributos]
                    }
                    [definicion_entidad]


listado_atributos   :/* vacio */
                    |[elemento_tabla]
                    |[listado_atributo],[elemento_tabla]


elemento_tabla :[def_columna]
               |[def_interrelacion]


def_columna    :
               [columna] [tipo_dato] [opc_def_columna]
```

```
tipo_dato       :varchar
                |varchar(numero)
                |numeric
                |numeric(numero)
                |numeric(numero,numero)
                |decimal
                |decimal(numero)
                |decimal(numero,numero)
                |integer
                |smallint
                |float
                |float(numero)
                |real
                |double precision
                |date
                |time

opc_def_columna     :/* vacio */
                    |[opc_def_columna]
                    [listado_opc_columna]

listado_opc_columna :not null
                    |not null autoincrement
                    |not null unique
                    |not null primary key
                    |default numero
                    |default null
                    |default variable
                    |default ‘ nombre ‘
                    |check ([condicion_busqueda])
                    |references nombre_tabla
                    |references
                    nombre_tabla([listado_columna])

def_interrelacion   :unique ( [listado_columna] )
                    |primary key ([listado_columna])
                    |foreign key ([listado_columna])
                    references nombre_tabla
                    |foreign key ([listado_columna])
                    references nombre_tabla
                    ([listado_columna])
```

```
                    |foreign key (listado_columna)
                    references nombre_tabla
                    (listado_columna)
                    on update cascade
                    |foreign key (listado_columna)
                    references nombre_tabla
                    (listado_columna)
                    on delete cascade
                    |check (condicion_busqueda)

listado_columna     :nombre_columna
                    |listado_columna , nombre_columna


condicion_busqueda  :condicion_busqueda
                    or condicion_busqueda
                    |condicion_busqueda
                    and condicion_busqueda
                    |not condicion_busqueda
                    |(condicion_busqueda)
                    |predicado

predicado           :comparacion_predicado
                    |entre_predicado
                    |like_predicado
                    |prueba_para_null
                    |en_predicado
                    |todo_o_algun_predicado
                    |prueba_existencia

comparacion_predicado   :escalar_exp comparación
                        escalar_exp
                        |escalar_exp comparación
                        subquery

entre_predicado     :escalar_exp not
                    between escalar_exp
                    and escalar_exp
                    |escalar_exp between escalar_exp
                    and escalar_exp

like_predicado      :escalar_exp not
                    like atom opt_escape
                    |escalar_exp like
                    atom opt_escape
```

```
opt_escape          :/* vacio */
                    |escape [atom]

prueba_para_null    :[column_ref] is not null
                    |[column_ref] is null

en_predicado        :[escalar_exp] not in (subquery)
                    |[escalar_exp] in ( subquery )
                    |[escalar_exp] not
                    in ([atom_commalist])
                    |[escalar_exp] in ([atom_commalist])


atom_commalist      :[atom]
                    |[atom_commalist] , [atom]

todo_o_algun_predicado   :[escalar_exp][comparación]
                         [any_all_some] subquery

any_all_some        :any
                    |all
                    |some

prueba_existencia   :exists [subquery]

subquery            :(select [opt_all_distinct]
                    [selection] [table_exp])

opt_all_distinct    :|all
                    |distinct

atom                :parametro
                    |numero
                    |&variable
                    |‘ nombre ‘
                    |‘ literal_hora ‘
                    |‘ literal_fecha ‘

Comparación         :=
                    |<>
                    |<
                    |>
                    |<=
                    |>=
```

```
escalar_exp         :[escalar_exp] + [escalar_exp]
                    |[escalar_exp] – [escalar_exp]
                    |[escalar_exp] * [escalar_exp]
                    |[escalar_exp] / [escalar_exp]
                    |+ [escalar_exp]
                    |- [escalar_exp]
                    |[atom]
                    |nombre_tabla . [nombre_tabla]
                    |[function_ref]
                    |( [escalar_exp] )


function_ref        :[ammsc] ( * )
                    |[ammsc] ( distinct nombre_tabla )
                    |[ammsc] ( all [escalar_exp] )
                    |[ammsc] ( [escalar_exp] )

ammsc               :avg
                    |min
                    |max
                    |sum
                    |count
```

Como se puede apreciar el listado_atributos mantiene la sintaxis de SQL al momento de crear los atributos de las tablas.

Si una entidad ya fue definida en otra clase no se deben colocar atributos. En su caso se debe escribir:

entity *nombre_entidad* ***{ }***

En caso de colocar otros atributos se agregarán a la entidad creada en el padre. Se debe tener cuidado de no agregar nuevos atributos con nombres iguales a los ya definidos con anterioridad. De ser así, el analizador retornará un error indicando que ya existe el campo que se intenta crear.

- **Definición de Métodos**

```
method nombre_metodo ([paramet_lista]) [cuerpo_metodo];

paramet_lista       : /*vacio*/
                    |[insert_paramet]
                    |[paramet_lista] , [insert_paramet]
```

```
insert_paramet      :nombre_parametro
                    |null

cuerpo_metodo       :{
                    |[definicion_funciones] ;
                    aux_definicion_funciones
                    }

aux_definicion_funciones :/* vacio */
                         |[definicion_funciones] ;
                         [aux_definicion_funciones]


definicion_funciones     :[delete_statement_searched]
                         |[insert_statement]
                         |[update_statement_searched]

delete_statement_searched     :delete from nombre_tabla
                              [opt_where_clause]

opt_where_clause    :/* vacio */
                    |[where_clause]

where_clause        :where [condicion_busqueda]

insert_statement    :insert into nombre_tabla
                    [opt_column_commalist]
                    [values_or_query_spec]

opt_column_commalist     :/* vacio */
                         |([listado_columna])

values_or_query_spec :   values
                         ([insert_atom_commalist])
                         |[query_spec]

query_spec          :select [opt_all_distinct]
                    [selection] nombre_tabla

selection           :[escalar_exp_commalist]
                    |*

escalar_exp_commalist    :[escalar_exp]
                         |[escalar_exp_commalist] ,
                         [escalar_exp]
```

```
insert_atom_commalist    :[insert_atom]
                         |[insert_atom_commalist] ,
                         [insert_atom]

insert_atom              :[atom]
                         |null

update_statement_searched     :update nombre_tabla
                              set [assignment_commalist]
                              [opt_where_clause]


assignment_commalist     :[assignment]
                         |[assignment_commalist] ,
                         [assignment]


assignment     :nombre_columna [comparacion]
               [escalar_exp]
               |nombre_columna [comparacion] null
               |nombre_columna [comparacion]
               [subquery] [escalar_exp]
```

IMPORTANTE: Los nombres de las variables (parámetros) utilizadas en la definición de los métodos deben terminar en "__". Esto es para evitar posibles problemas al momento de implementar los métodos y para que el motor de bases de datos pueda diferenciar las variables_parametros de las variables de la base de datos.

Ejemplo:

method *nombre_metodo* **(** *parametro1___, parametro2___, …* **);**

Para la sentencia de inserción (**insert**) los valores a insertar de tipo numérico deben iniciar con "&". Esto es solo para la sentencia **insert** dentro de los métodos, no aplica para usar la sentencia como una consulta independiente. (Ver más adelante OQL). Los demás tipos de datos (fecha, hora, varchar) inician y terminan con (') - (comilla simple).

Ejemplo:

insert into *nombre_tabla* **values** (*&num__*,'char__', null);

A continuación dos ejemplos completos que muestran una correcta definición de clases, entidades y metodos.

Ejemplo No 1 :: ODL

```
/*implementacion de un ejemplo de definicion de clase*/

// Definicion Clase Compra_Mercancia
class compra_mercancia
{
 // inicia la definicion de entidades pertenecientes
 // a la clase Compra_mercancia

 entity mercancia
 {
  cod_mercancia varchar(5) not null primary key,
  valor_existencia numeric
 }
 entity diario
 {
  cod_trans    numeric,
  fecha        date,
  hora         time,
  cod_mercancia varchar(5),
  foreign key(cod_mercancia)
  references mercancia(cod_mercancia) on delete cascade
 }
 /*Finaliza la definicion de entidades pertenecientes a
 la clase Compra_mercancia*/

 method comprar (cod_trans__, valor__,
                 cod_mercancia__, hora__)
 {
   insert into diario values (&cod_trans__, null,
                              'hora__', &cod_mercancia);
   update mercancia
   set valor_existencia = valor_existencia + valor__
   where cod_mercancia = cod_mercancia__ ;
  }
 };
```

Ejemplo No 2 :: ODL

```
// Definicion Clase Compra_Mercancia_Contado
class compra_mercancia_contado extend compra_mercancia
{

 entity bancos
 {
  nit_banco         numeric not null primary key,
  nom_banco         varchar(30),
  saldo             numeric,
  cod_cuenta   numeric(4),
  foreign key(cod_cuenta) references mayor(cod_cuenta)
 }



 entity cuentas
 {
  cuenta_ban        numeric not null primary key,
  saldo             numeric,
  nit_banco         numeric,
  foreign key(nit_banco) references bancos(nit_banco)
 }

 method comprar
  // Parametros Entidad Cuentas
  (cuenta_ban__, saldo__, nit_banco__ , valor__)
 {
   insert into cuentas values
   (&cuenta_ban__, &saldo__, &nit_banco__);

   update cuentas set
   saldo = saldo - valor__
   where cuenta_ban = cuenta_ban__;
 }
};
```

1.1.7. Sintaxis de sentencias OQL

- **Consultar entidades**

```
consulta_entidades  :[select_statement]
                    |[delete_statement_searched]
                    |[insert_statement]
                    |[update_statement_searched]
                    |[drop_statement]
```

```
select_statement    :select [opt_all_distinct]
                    [selection] nombre_tabla

drop_statement      :drop table nombre_tabla ,
                    opc_tabla... ;
```

Las sentencias ***delete, insert, update*** *fueron definidas anteriormente.*

Las consultas a entidades conservan la sintaxis SQL. Esto con el fin de facilitar al usuario la comprensión del MODRO así como su manipulación y consulta, permitiendo a su vez mantener algunas características importantes del modelo entidad-relación. Además debido al enlace que se realiza con el motor de bases de datos la estructura tradicional de tablas interrelacionadas por el momento se mantiene, se espera en las siguientes fases de desarrollo realizar los cambios necesarios a esta estructura.

- **Usar un método**

  method *nombre_clase (parámetro_1, parámetro_2,...)* **;**

  La sintaxis anterior muestra la forma apropiada de acceder al método de una clase. Los parámetros pueden ser de tipo variable, fecha, hora o numérico.

  La utilización de los métodos es una de las características principales del MODRO permitiendo estos la activación de manera automática de los procedimientos almacenados en la clase. En el capitulo 3 se explica con mas detalle el funcionamiento de los mismos.

Según el profesor Albarracin en su tesis doctoral el concepto de clase lo define de la siguiente manera:

"Convencionalmente una clase es definida como el grupo de objetos que comparten el mismo conjunto de atributos, interrelaciones y métodos. Sin embargo, esta definición se extiende en esta tesis, pues el concepto de clase supera a la entidad individual de grano fino, implícita en la anterior definición (al mencionar atributos). En efecto, aquí una clase es planteada, no como una entidad, sino como un conjunto de entidades enlazadas e inter-actuantes de acuerdo a un mismo método, el cual no es particular a cada entidad sino común a todas ellas."

Por lo tanto la manipulación de las clases creadas se sintetiza con el comando **method** *nombre_clase (param1. param2, …)* porque este realiza todo el trabajo de inserción y actualización de las entidades relacionadas con la clase.

1.2. IMPLEMENTACION DEL ANALIZADOR LEXICO

Una vez definida la gramática que permite la generación de los componentes ODL y OQL el siguiente paso es la implementación de un intérprete que soporte dichos componentes. Para ello se hace necesario construir los analizadores léxico y sintáctico.

Los analizadores permiten controlar las entradas digitadas por el usuario y dan paso a la generación de la estructura de los archivos de jerarquía de clases. Un analizador léxico es aquel que toma las entradas digitadas y las divide en componentes léxicos, "que son secuencias de caracteres que tienen un significado atómico"[2]. Una vez hecha esta clasificación las envía al analizador sintáctico como TOKENS. La siguiente figura muestra el proceso llevado a cabo.

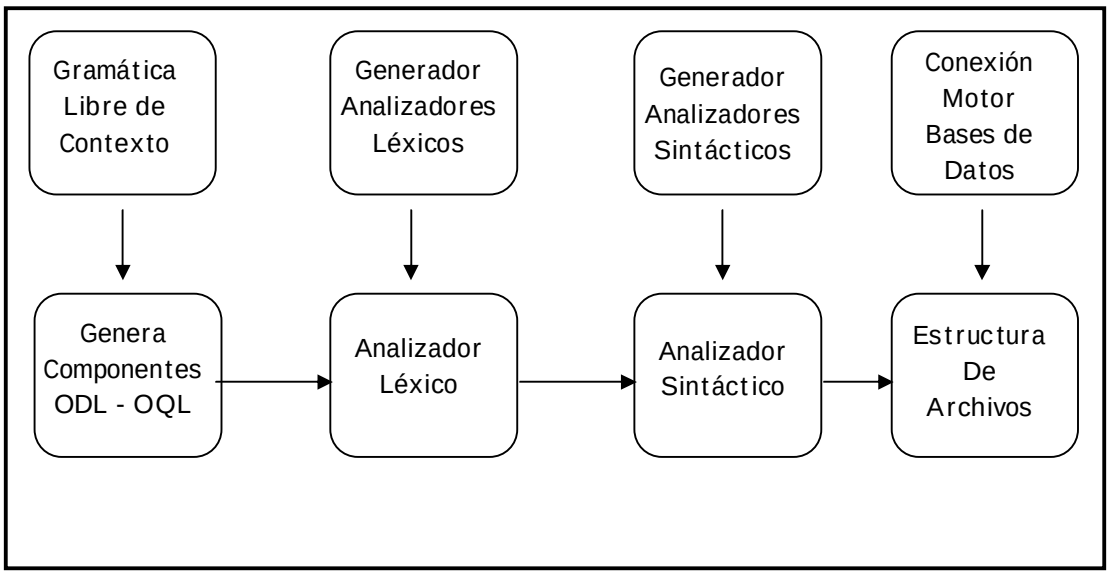


Figura No 1. Diseño de Interprete del MODRO

El lenguaje escogido para el desarrollo del intérprete fue JAVA. Entre las principales razones se encuentran:

[2] Galves Rojas Sergio, Mata Miguel. Java a Tope : Traductores y Compiladores. Universidad de Malaga, 2005. Pag 17.

- Ofrece la potencia del diseño orientado a objetos con una sintaxis sencilla y un entorno robusto y agradable.

- Cuenta con un conjunto de clases potentes y flexibles.

- Permite ejecutar aplicaciones en diferentes plataformas (Macintosh, Windows, Unix, …).

- Excelente desempeño en trabajo sobre la red.

- Altos niveles de seguridad como: fuertes restricciones al acceso a memoria, rutinas de verificación de los códigos de byte que asegura que no se viole ninguna construcción del lenguaje, verificación del nombre de clase y de restricciones de acceso durante la carga, entre otras.

Lo anterior permite ver las características principales que convierten a JAVA en una excelente solución para el desarrollo del intérprete.
Para la creación del analizador léxico se utilizo JFlex 1.4.1 el cual es un generador de analizadores léxicos. Un generador es un programa que convierte una notación formal por medio de expresiones regulares, en un programa (analizador léxico) que puede reconocer los componentes léxicos de un programa fuente[3].

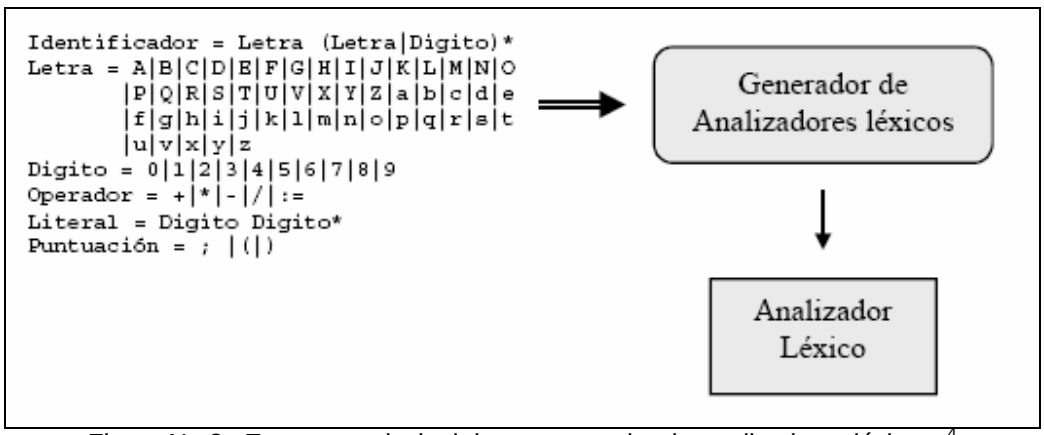


Figura No 2. Esquema principal de un generador de analizadores léxicos. [4]

[3] Luengo Maria. Desarrollo y evaluación de técnicas de construcción de procesadores de lenguaje para máquinas abstractas orientadas a objetos. Universidad de Oviedo, 2002. Pag 31.
[4] Ibid. Pag 32.

A continuación se muestra un ejemplo de una entrada y la salida de un analizador léxico.

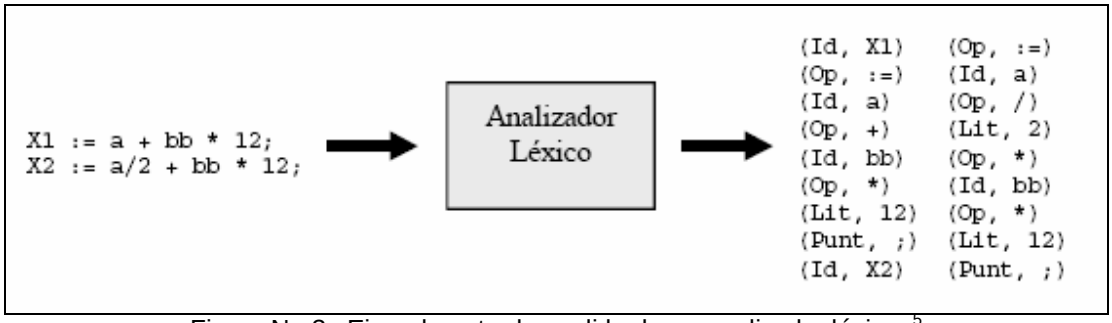


Figura No 3. Ejemplo entrada –salida de un analizador léxico. [5]

JFlex esta escrito en Java y genera los analizadores léxicos en Java. Fue escrito tomando como referencia el generador Jlex el cual fue desarrollado por Elliot Berk en la Universidad de Princeton. Entre sus principales características se encuentran:

- Soporta Unicode completamente.
- Rápida generación de escaneo.
- Apropiada especificación de sintaxis.
- Independencia de plataforma.
- Compatibilidad con JLex.

Para generar el analizador léxico JFlex requiere de un archivo que contiene las expresiones regulares y la clasificación de los tokens según la gramática a implementar. En el Anexo A se encuentra el archivo utilizado para el interprete.

Una vez escrito el archivo para JFlex se procede a generar el analizador léxico. El siguiente comando genera el analizador léxico desde la consola de Windows (MS-DOS):

```
java -jar jflex.jar lex_odl_oql.flex
```

[5] Ibid. Pag 32.

Donde **jflex.jar** es el archivo que contiene los archivos ejecutables de JFlex y **lex_odl_oql.flex** es el archivo creado con las expresiones regulares y la clasificación de los tokens de los componentes ODL y OQL.

La siguiente es la salida que genera JFlex si el proceso se hizo correctamente:

```
Reading "lex_odl_oql.flex"
Constructing NFA : 554 states in NFA
Converting NFA to DFA :
306 states before minimization, 281 states in minimized DFA
Writing code to "Lexer.java"
```

Como se puede apreciar en la salida del programa se generó el archivo **Lexer.java,** el cual contiene el programa en Java del analizador léxico de los componentes ODL y OQL para el intérprete.

Para poder realizar el análisis léxico se requiere utilizar autómatas finitos deterministicos, por eso se ve como JFlex construye primero los autómatas finitos no deterministicos los cuales luego convierte en autómatas finitos deterministicos AFD (o en ingles DFA). Esto se debe a que la mejor forma de implementar un analizador léxico computacionalmente es utilizando autómatas finitos.

Otra forma de poder generar el analizador léxico con JFlex es utilizando el entorno grafico que trae consigo esta herramienta. A continuación se muestra su interfaz:

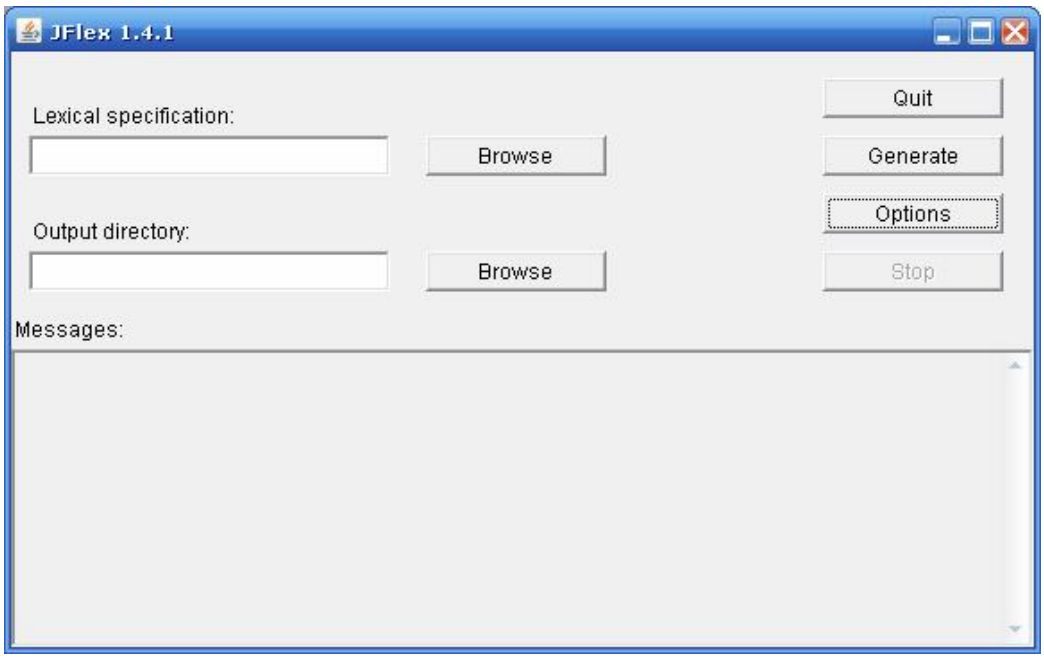


Figura No 4. Interfaz Grafica de JFlex 1.4.1.

1.3. IMPLEMENTACION DEL ANALIZADOR SINTACTICO

Un analizador sintáctico es: "un programa que recibe como entrada los elementos individuales o componentes léxicos (tokens) del programa fuente y reconoce la estructura de las sentencias o instrucciones que lo conforman"[6]. Es por esto que las principales funciones del analizador son detectar e informar de los errores sintácticos y la generación de un árbol sintáctico por cada una de las expresiones digitadas por el usuario.

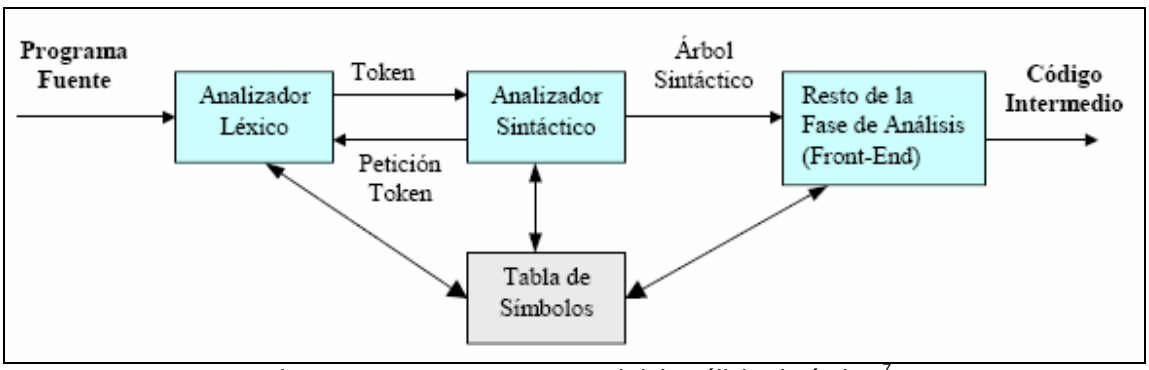


Figura No 5. Esquema general del análisis sintáctico.[7]

Las siguientes producciones de una gramática libre de contexto describen expresiones aritméticas simples en notación BNF

```
exp       :=   [term] "+" [exp]
           |   [term] "-" [exp]
           |   [term]
term      :=   [factor] "*" [term]
           |   [factor] "/" [term]
           |   [factor]
factor    :=   [number]
           |   "("  [exp]  ")"
```

[6] Ibid. Pag 24.
[7] Ibid. Pag 24.

La expresión “ 9 + 5 * 2 “ generaría el siguiente árbol sintáctico:

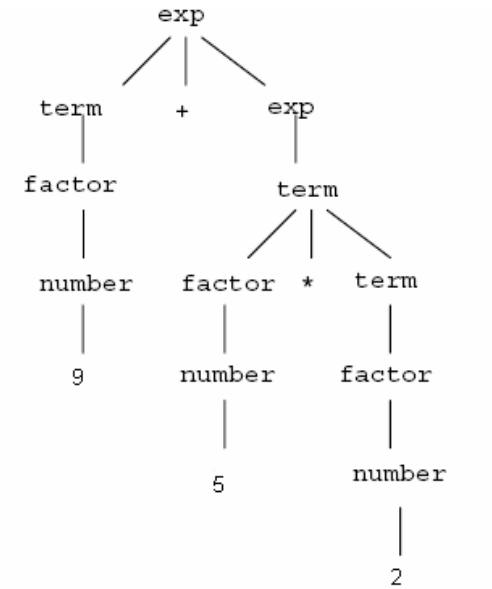


Figura No 6. Árbol Sintáctico de la expresión “9 + 5 * 2”.

A diferencia de un analizador léxico, el analizador sintáctico se puede implementar de diferentes maneras. Un analizador sintáctico puede ser ascendente o descendente, el análisis ascendente consiste en partir de la cadena de componentes léxicos y mediante reducciones se llega a la cima del árbol sintáctico. Por su parte el análisis descendente parte de la cima del árbol y por medio de derivaciones de las reglas gramaticales se genera el árbol sintáctico para la cadena de componentes léxicos ingresada. Como se aprecia ambos análisis generan un árbol sintáctico el cual es utilizado para hacer luego la generación de la estructura de los archivos de jerarquía de clases por medio de la traducción dirigida por sintaxis.

Dentro de los análisis descendentes se encuentran el análisis descendente con retroceso y el análisis de gramáticas LL (Left to rigth, Left most). El análisis descendente con retroceso realiza una prueba sistemática de todas las alternativas posibles, su construcción es fácil pero emplea más tiempo para el análisis que otros analizadores y no da un buen diagnostico de los errores que encuentra. Por su parte el análisis de gramáticas LL lee la cadena de entrada de izquierda a derecha utilizando reglas de producciones de izquierda e inspeccionando un solo símbolo de entrada para elegir la regla conveniente, su principal inconveniente es el no aplicar para todas las gramáticas y estar limitado a gramáticas de tipo LL.

Por su parte los análisis ascendentes sin retroceso se dividen en análisis SLR, LALR y LR Canónico. El análisis SLR (Simple LR – Left to right, right most –) es el más sencillo y menos potente. Por otro lado el análisis LR-Canónico es el más costoso y potente, en cambio el LALR (Look-Ahead LR) está entre los dos, en términos de su complejidad y potencia. Cada análisis se diferencia solamente en la construcción de la tabla que utilizan para el análisis. A continuación se muestra el algoritmo genérico de análisis LR.

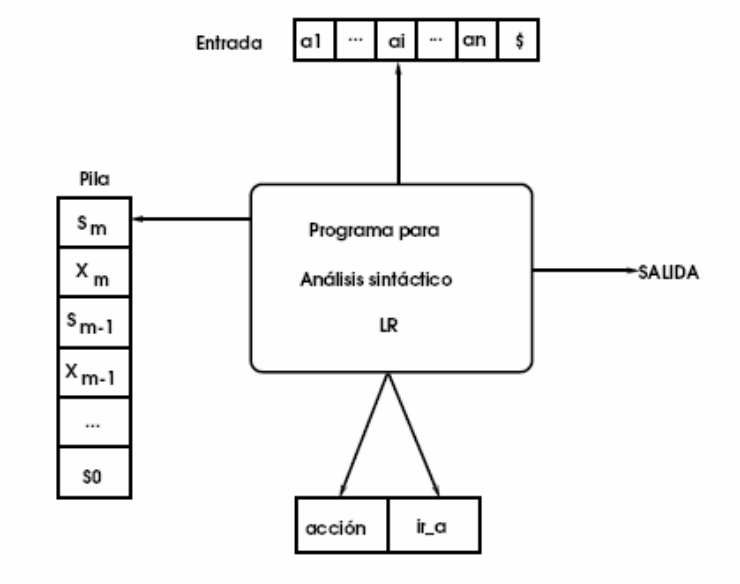


Figura No 7. Algoritmo genérico de análisis LR.

El conjunto de gramáticas LL es un subconjunto de gramáticas LR. Los analizadores sintácticos LR pueden reconocer prácticamente la totalidad de las construcciones de los lenguajes que se pueden definir mediante gramáticas libres de contexto. El método de análisis LR es de tipo reducción – desplazamiento, sin retroceso, más general que se conoce pero además su aplicación es tan eficiente como la de otros métodos reducción – desplazamiento menos generales. El manejo de errores en los analizadores ascendentes se realiza tan pronto como sea posible hacerlo.

Con base en lo anterior se definió implementar un analizador sintáctico de tipo LR Canónico por ser el más potente. El generador de analizadores sintácticos más común es el YACC (Yet Another Compiler Compiler) el cual implementa el método de análisis sintáctico LALR(1). Actualmente existen versiones mejoradas de este generador como lo es BISON que también utiliza el análisis LALR(1). Un inconveniente con estas herramientas es que generan los analizadores sintácticos en C/C++, por lo que se hizo necesario conseguir un generador de analizadores para el lenguaje de programación Java.

Nombre	Tipo de Análisis	Lenguaje Generado	Tratamiento de errores	Especifi-cación	Otras Características Destacables
Lex/Yacc	LALR(1)	C	Malo	BNF mezclado con C	- Falta de integración entre analizadores léxicos y sintácticos. - Ciclo de desarrollo largo. - No permite generar ASTs.
Lemon	LALR(1)	C	Malo	BNF mezclado con C	- Mantiene la misma estructura y modo de operar de Yacc. - Ciclo de desarrollo largo. - No permite generar ASTs.
Accent	AE/AP	C	Malo	EBNF mezclado con C	- Procesa cualquier gramática Libre de Contexto. - Mantiene la misma estructura que los generadores clásicos. - Ciclo de desarrollo largo. - No permite generar ASTs.
Cocktail	LALR(1) LL(1)	C Modula-2	Regular	BNF mezclado con C o Módula-2	- Mantiene características similares a los generadores clásicos. - Formado por un conjunto de herramientas . - Falta de integración entre analizadores léxicos y sintácticos. - Ciclo de desarrollo largo.
Antlr	LL(k)	C++/Java	Bueno	EBNF mezclado con C++/Java	- Flexibilidad de predicados costosa. - Ciclo de desarrollo largo. - Falta de integridad en los ASTs generados.
JavaCC	LL(k)	Java	Bueno	EBNF mezclado con Java	- Ciclo de desarrollo largo. - Dispone de una herramienta adicional para generar ASTs.

Tabla No 3. Características de Generadores de Analizadores.[8]

Como se puede apreciar son pocos los generadores de analizadores sintácticos para Java y aun más que trabajen con el análisis LALR(1). El grupo mostró interés en desarrollar un generador de analizadores sintáctico LR-Canónico para Java, y se encontró con CUP el cual es un generador de análisis sintácticos LALR(1).

[8] Ibid. Pag 79.

Consultando con los desarrolladores del programa CUP se tomó la decisión de utilizar este generador para el analizador sintáctico que se estaba necesitando. El profesor Scott Hudson, quien es uno de los desarrolladores de este programa, nos comentó que inicialmente el proyecto CUP esperaba desarrollar un generador de analizadores sintácticos LR-Canónico para el lenguaje de programación Java, pero el costo en recursos del sistema para su implementación era muy elevado debido a la gran cantidad de estados necesarios para lograr un análisis de este tipo, por lo tanto se enfocaron en optimizar los estados generados por el analizador y así lograron implementar el análisis LALR(1).

Se utilizo la versión 0.11a de CUP. (Marzo de 2006). La cual recibe como entrada un archivo con las reglas gramaticales, los símbolos terminales y no terminales, así como también algunos métodos definidos por el usuario para ser implementados con las clases de Java. Para más información sobre este archivo ver el Anexo B.

El siguiente comando genera el analizador sintáctico en Java:

```
java -jar cup.jar sint_odl_oql.cup
```

Donde **cup.jar** es el archivo que contiene los archivos ejecutables de CUP y el archivo **sint_odl_oql.cup** contiene las reglas gramaticales, los símbolos terminales y no terminales, de los componente ODL y OQL.

La siguiente es la salida que genera CUP si el proceso se hizo correctamente:

```
------- CUP v0.11a beta 20060608 Parser Generation Summary --
  0 errors and 0 warnings
  72 terminals, 79 non-terminals, and 191 productions
declared,
  producing 371 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.java", and "sym.java".
----------------------------------------------------
(v0.11a beta 20060608)
```

Los archivos creados por CUP son: **sym.java** y **parser.java**. El primero contiene los símbolos utilizados en la gramática y el segundo contiene el analizador sintáctico con las reglas gramáticas, los símbolos terminales y no terminales así como también los métodos utilizados por la traducción dirigida por sintaxis para la generación de las estructuras de los archivos de jerarquía de clases la cual se explica más adelante.

Una de las ventajas de utilizar los generadores de analizadores léxico y sintácticos, JFlex / CUP, es la compatibilidad que entre estos existe, lo que facilitó de cierta manera el desarrollo de los analizadores léxico y sintáctico para el interprete del MODRO.

1.4. DEFINICION DE LA ESTRUCTURA DE DATOS

Para la correcta interacción de cada uno de los componentes del intérprete, era necesaria la creación de varios archivos de datos que permitieran almacenar información relevante acerca de las clases creadas. De igual manera, el intérprete genera un archivo temporal que es usado al momento de la creación de las entidades que componen cada una de las clases, el archivo **entidades.sql**.

La forma en la que el intérprete crea estos archivos, se conoce como traducción dirigida por sintaxis, que consiste en ir haciendo otras tareas al mismo tiempo en que se realiza el análisis sintáctico. Para esto se aprovecharon varias características muy importantes del generador de analizadores sintácticos CUP; la primera de ellas es que permite agregar métodos y variables adicionales a la clase "**parser.java**" generada, agregando sus correspondientes códigos Java entre "**parser code {:"** y **":}"** en el archivo de especificación de las reglas gramaticales, con extensión "**.cup".** Otra segunda propiedad de CUP es que permite agregar código Java dentro de expresiones gramaticales (debe estar encerrado entre "**{:**" y "**:}**", de tal manera que se pueda ejecutar una porción de código especifica cuando el texto que esta procesando el analizador coincida con una de las reglas gramaticales definidas.

La gramática definida para el analizador sintáctico está compuesta por reglas gramaticales que contienen terminales ("TOKENS" regresados por el analizador léxico) y no terminales. Los terminales pueden tener asociados un valor, como por ejemplo los terminales NAME, cuyo valor asociado puede ser el nombre de un atributo, clase o entidad; éstos son declarados en el archivo **Sint_ODL_OQL.cup** de tipo Object para indicar que su valor asociado puede ser de tipo Integer, String, etc. De forma similar si se quiere asociar un valor a un no terminal, se debe declararlos en la especificación de la gramática de tipo Integer si el valor asociado es un entero o de tipo Object para cualquier tipo, sea String, Integer o cualquier otro.

Ejemplo:

Declaración de terminales que no tienen ningún valor asociado:

```
terminal PUNTO, DOSPUNTO, MAS, MENOS
```

Declaración de terminales que tienen un valor asociado:

```
terminal Object NAME, TABLE, INTNUM, FECHA
```

Declaración de no terminales que tienen un valor asociado que puede ser de tipo Integer, String o cualquier otro.

```
non terminal Object herencia, definicion_clase, cuerpo_clase
```

NOTA: Para más información acerca de la declaración de terminales y no terminales, y de la especificación de las reglas gramaticales ver sección 1.1: DEFINICION DE UNA GRAMATICA LIBRE DE CONTEXTO PARA LOS COMPONENTES ODL Y OQL y ANEXO B: ARCHIVO CUP PARA GENERAR EL ANALIZADOR SINTACTICO DE LOS COMPONENTES ODL Y OQL.

Una vez hecha la declaración de los terminales y no terminales, es posible asignar y obtener los valores asociados a cada uno de ellos de la siguiente manera:

- **Asignación de valores a no terminales**: Esto se logra igualando la variable RESULT al valor que se quiera asignarle al no terminal de una expresión gramatical específica. RESULT es una etiqueta asignada automáticamente a todos los no terminales.

  Ejemplo: El siguiente bloque de código asigna al no terminal **tipo_dato** el valor String "**DECIMAL**" cuando el analizador léxico le envié al analizador sintáctico el "TOKEN" DECIMAL que no tiene ningún valor asociado.

  ```
  tipo_dato ::=  DECIMAL
                 {: RESULT = new String("DECIMAL"); :}
  ```

- **Obtener valores asociados a terminales y no terminales**: El procedimiento es el mismo cuando se quiere obtener el valor asociado de un terminal o un no terminal; se logra agregando la expresión "**:etiqueta**" después del terminal o no terminal al cual se le desea obtener el valor. Donde "**etiqueta**" es un nombre de variable valida en Java, del mismo tipo de dato que se declaro el terminal o no terminal.

  Ejemplo: El siguiente bloque de código asigna al no terminal tipo_dato el valor String "**VARCHAR(numero)**" donde "*numero"* es el valor asociado al terminal INTNUM; cuando el analizador sintáctico reciba del analizador léxico los "TOKENS": VARCHAR, IPAREN, INTNUM, DPAREN.

  ```
  tipo_dato ::=  VARCHAR IPAREN INTNUM:numero DPAREN
                 {: RESULT = new String("VARCHAR("+
                           numero.toString()+")"); :}
  ```

;

Esta característica de CUP de asignación y obtención de valores asociados a terminales y no terminales, es explotada ampliamente para la generación de los archivos de clase y mas aún para la creación del archivo que contiene las instrucciones SQL de definición de entidades, cuyo proceso se resume en la Figura No 8.

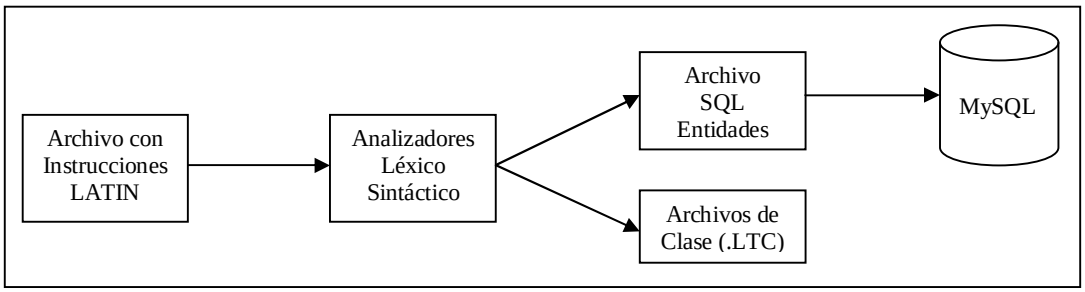


Figura No 8. Proceso de generación de archivos

1.4.1 Archivos de clases

Los archivos de clase cuya extensión es .LTC (Latin Class[9]), constituyen el medio de interacción entre el intérprete y el sistema de bases de datos MySQL (usado para la creación de las entidades) en el momento en que se desee implementar el método perteneciente a una clase específica.

Los archivos de clase se van generando al mismo tiempo que se hace el análisis sintáctico y se crea uno por cada clase de acuerdo a la estructura mostrada en la Figura No 9.

[9] Clases de LATIN (en español), nombre que hasta el momento se asignó al SGBDRO que se espera construir a futuro

```
[entity]
 nombres entidades propias de la clase
 nombres entidades heredadas del padre

[method]
 nombre método
 (parámetros heredados del método de la clase padre, parámetros del método de la
clase)
 {
   instrucciones SQL heredadas del método de la clase padre
   instrucciones SQL del método propio de la clase
 }
```

Figura No 9. Estructura de los archivos de clases

Los archivos de clase, son ficheros planos que contienen:

- La palabra reservada entity encerrada entre corchetes rectos ([ ]), para indicar que a continuación vienen palabras que corresponden a nombres de entidades.

- Los nombres de las entidades propias de la clase, que son declaradas en la interfaz del intérprete al momento de definir las entidades junto con sus atributos e interrelaciones y el método de cada una de las clases a crear.

- Los nombres de cada una de las entidades de la clase padre cuando la clase a la cual pertenece el archivo posee herencia.

- La palabra reservada method encerrada entre corchetes rectos ([ ]), para indicar que a continuación viene la especificación del método perteneciente a la clase por la cual se esta creando el archivo .LTC.

- El nombre del método que se especificó al momento de definir la estructura de la clase en la interfaz del intérprete, seguido de un paréntesis de apertura “(“ el cual indica que en la misma línea, a continuación de éste siguen los parámetros del método.

- Los nombres de los parámetros del metodo de la clase padre cuando se definió herencia, y los nombres de los parámetros propios de la clase a la cual pertenece el archivo LTC seguidos de un paréntesis de cierre “)” el cual indica que ha finalizado la definición de todos los parametros del metodo.

- Una llave de apertura ({), la cual indica que a continuación vienen las instrucciones SQL correspondientes al método.

- Las instrucciones SQL del método, tal como se escribieron al momento de definir la estructura de clase en la interfaz del interprete.

- Las instrucciones SQL del método de la clase padre, cuando la clase a la cual pertenece el archivo posee herencia.

- Una llave de cierre (}), para indicar que han finalizado las instrucciones SQL del método.

Aprovechando las características de CUP mencionadas anteriormente, se agregaron a la clase del analizador sintáctico "**parser.java**" los siguientes métodos y variables que son usados para la creación de los archivos de clase:

- La variable de tipo *String* **NombreClase***,* en la cual se almacena el nombre de la clase, cada vez que el analizador pasa por la siguiente expresión gramatical:

```
definicion_clase  ::=  CLASS NAME;
```

- La variable de tipo *String* **ClasePadre**, en la cual se almacena el nombre de la clase padre si se está definiendo herencia y se cumplen las siguientes expresiones gramaticales:

```
definicion_clase    ::=  CLASS NAME herencia;
herencia            ::=  EXTEND NAME;
```

- La variable de tipo *boolean* **TienePadre**, la cual tiene por defecto el valor **false,** hasta que se cumplan las expresiones gramaticales del ítem anterior cuando toma el valor **true** y volviendo a ser **false** cuando se declare una clase no derivada de ninguna otra, es decir se cumpla la expresión gramatical:

```
definicion_clase  ::=  CLASS NAME;
```

- El método **escribir_clase** que se encarga de escribir en los archivos de clase los nombres de las entidades y las instrucciones SQL del método (una entidad y una instrucción SQL por vez) pertenecientes a la clase que el analizador sintáctico se encuentra procesando en el momento. Éste método posee tres parámetros que son:

    - nom_archivo: De tipo *String*, corresponde al nombre de la clase que se está procesando.
    - cadena: De tipo *String*, corresponde al nombre de la entidad o a la instrucción SQL del método que se desea escribir en el archivo de clase.

- tipo: De tipo *char*, se usa para indicarle al método si la información que se desea escribir es el nombre de una entidad (en cuyo caso su valor debe ser ‘e’) o corresponde a una instrucción SQL perteneciente al método de la clase (en cuyo caso su valor deber ser ‘m’).

A continuación, un segmento de código del archivo **Sint_ODL_OQL.cup** donde se hace uso de éste método. Se escribe en el archivo de clase el nombre de una entidad perteneciente a la clase procesada en ese momento por el analizador sintáctico.

```
definicion_entidad ::= ENTITY NAME:nombre_entidad
                {: parser.escribir_clase(
                   parser.NombreClase,
                   nombre_entidad.toString()+"\n",'e');
                 :}
```

- El método **entidades_padre**, usado para copiar los nombres de las entidades de la clase padre (cuando se define herencia) al archivo de clase de la clase hija. Este método posee dos parámetros:

  - ClasePadre: De tipo *String*, corresponde al nombre de la clase de la cual es derivada la clase que el analizador sintáctico está procesando en el momento.
  - ClaseHijo: De tipo *String*, corresponde al nombre de la clase a la cual queremos agregar los nombres de las entidades del padre en su respectivo archivo clase.

  A continuación un segmento de código extraído del archivo de definición de reglas gramaticales CUP, que muestra el uso de éste método.

  Antes de llamar al método, es necesario verificar que a la clase efectivamente se le ha definido herencia, de ser así la variable **TienePadre** tendrá el valor **true** y en la variable **ClasePadre** estará almacenado el nombre de la clase padre.

```
cuerpo_clase ::= ILLAVE definicion_entidad
               {: if (parser.TienePadre)
                    parser.entidades_padre(
                    parser.ClasePadre,
                    parser.NombreClase);
               :}
               definicion_metodo
               DLLAVE
               ;
```

- El método de tipo *String* **parametros_padre**, el cual retorna todos los parámetros que posee el método definido para la clase de la cual deriva la actual clase procesada por el analizador sintáctico. Los parámetros son devueltos en una cadena de caracteres separados por una coma; y el método **escribir_herencia** que se encarga de copiar todas las instrucciones SQL del método perteneciente a la clase padre al archivo de clase perteneciente a la clase hija.

  El método **parametros_padre** posee un parámetro:

  - ClasePadre: de tipo *String*, hace referencia al nombre de la clase de la cual deseemos obtener sus parámetros.

  El método escribir_herencia posee dos parámetros:

  - Padre: de tipo *String*, hace referencia a la clase de la cual vamos a obtener las instrucciones SQL que se van a copiar en la clase hija.
  - Hijo: de tipo *String*, hace referencia al nombre de la clase que el analizador sintáctico está procesando en el momento y que deriva de otra clase.

  Debido a que en el MODRO una clase se plantea como un conjunto de entidades enlazadas e interactuantes que comparten el mismo método, al momento de declarar una clase como hija de otra previamente creada se debe declarar su propio método que complementa el de su padre, es decir las instrucciones SQL definidas en la clase padre como parte de su método son agregadas a las mismas instrucciones SQL pertenecientes a la clase hija. Esto es posible, gracias a los dos métodos descritos anteriormente.

  A continuación se muestran las expresiones regulares que corresponden a la definición de un método según la gramática desarrollada para el analizador sintáctico:

```
definicion_metodo   ::=  METHOD NAME IPAREN
                         paramet_list DPAREN
                         cuerpo_metodo
                         ;

paramet_lista  ::=  insert_paramet
                    |
                    paramet_lista COMA insert_paramet
```

```
                    ;

insert_paramet ::=  |
                    NAME
                    |
                    NULLX
                    ;

cuerpo_metodo  ::=  ILLAVE
                    definicion_funciones
                    DLLAVE
                    ;
```

A la anterior porción de la gramática se agregaron varias instrucciones Java que incluían entre otras cosas el llamado a los dos métodos anteriormente mencionados.

```
definicion_metodo   ::= METHOD NAME:nombre_metodo IPAREN
                    {:parser.escribir_clase(
                      parser.NombreClase,
                      nombre_metodo.toString()+
                      (",'m');

                    if (parser.TienePadre)
                         parser.escribir_clase(
                         parser.NombreClase,
                         parser.parametros_padre(
                         parser.ClasePadre),'m');
                    :}
                    paramet_lista:param_list DPAREN
                    {:parser.escribir_clase(
                      parser.NombreClase,
                      param_list.toString()+")",'m'); :}
                    cuerpo_metodo
                    ;


paramet_lista  ::=  insert_paramet:ins_param
                    {:RESULT = new String(
                    ins_param.toString());:}
                    |
                    paramet_lista:par_lis
                    COMA insert_paramet:ins_par
                    {:RESULT = new String(
                    param_list.toString()+",
                    "+ins_param.toString());:}
```

```
                    ;

insert_paramet ::=  {: RESULT = new String(); :}
                    |
                    NAME:nombre
                    {: RESULT = new String(
                    nombre.toString()); :}
                    |
                    NULLX
                    {: RESULT = new String("NULL"); :}
                    ;

cuerpo_metodo  ::=  ILLAVE
                    {: parser.escribir_clase(
                    parser.NombreClase,
                    " \n{\n",'m');
                    if (parser.TienePadre)
                        parser.escribir_herencia(
                              parser.ClasePadre,
                              parser.NombreClase);

                    :}
                    definicion_funciones:def_func
                    {:parser.escribir_clase(
                    parser.NombreClase,
                    def_func.toString(),'m'); :}
                    DLLAVE
                    {:parser.escribir_clase(
                    parser.NombreClase,
                    "}\n",'m'); :}
                    ;
```

- El método **existe_entidad**, de tipo *boolean*; es usado para determinar si una entidad declarada en una clase derivada de otra (herencia), ya se encuentra definida en la clase padre (en cuyo caso el método retorna el valor *true*) o si por el contrario dicha entidad no existe en la clase padre y se está tratando de crear una nueva entidad (en cuyo caso el método retorna false). Este método posee dos parámetros:

    - ClasePadre: De tipo *String*, corresponde al nombre de la clase de la cual deriva la actual clase procesada por el analizador sintáctico.
    - NombreEntidad: De tipo *String*, corresponde al nombre de la entidad declarada en la clase hija y cuya existencia en la clase padre se desea verificar.

- El método **actualizar_metodo**, el cual es usado para actualizar el número de campos usados en las instrucciones INSERT del método de la clase padre, cuando en la clase hija se agrega un nuevo campo a una entidad ya existente en la clase padre. Este método es de gran importancia, ya que si no se actualizan estas instrucciones al momento de tratar de implementar el método de la clase padre, ocurrirá un error al no coincidir el número de campos que se tratan de insertar con el número de campos de la entidad.

  El método **actualizar_metodo** posee dos parámetros:

  - NombreClase: De tipo *String*, corresponde al nombre de la clase padre que posee el método a modificar.
  - NombreEntidad: De tipo *String*, corresponde al nombre de la entidad a la cual se han agregados nuevos campos en la clase hija y cuyas instrucciones de inserción (si las hay) deben ser modificadas en el cuerpo del método existente en la clase padre.

1.4.2 Archivo SQL de definición de entidades

Debido a que inicialmente el Lenguaje LATIN hace uso del motor de bases de datos MySQL para el guardado de los datos, es necesario generar un archivo intermedio que convierta las instrucciones de definición de entidades usadas en el nuevo lenguaje a instrucciones SQL validas que puedan ser enviadas a MySQL.

El proceso de creación de dicho archivo fue muy sencillo, considerando que para la definición de entidades en el nuevo lenguaje LATIN se hace uso de la palabra "**entity"** en lugar del tradicional CREATE TABLE usado en SQL, seguido de dos llaves y entre ellas la definición de campos cuya sintaxis es idéntica a la usada en el lenguaje SQL, por lo tanto para generar el archivo **entidades.sql** solo era necesario reemplazar la palabra "**entity"** por la expresión CREATE TABLE, de igual forma reemplazar las llaves por paréntesis. La definición de los campos se copia exactamente igual desde la interfaz al archivo SQL generado.

Sin embargo, debido a que en una clase se pueden definir campos de una entidad existente en la clase padre (si se ha definido herencia), es necesario modificar esa entidad ya creada con anterioridad agregándole los nuevos campos definidos en la correspondiente clase hija, haciendo uso ya no de las instrucciones CREATE TABLE para la creación de tablas, sino de las correspondientes instrucciones ALTER TABLE para la modificación de tablas. Para determinar cual de las dos instrucciones copiar en el archivo generado, el analizador sintáctico hace uso del método **existe_entidad** descrito en el apartado 1.4.1: Archivos de clases.

En ambos casos fue necesario el uso del método **escribir_sql**, que permite copiar en el archivo **entidades.sql** cualquier cadena de texto que se desee. Dicho método posee un parámetro:

- cadena: De tipo *String*, corresponde a la cadena de texto que se desea copiar en el archivo entidades.sql con las instrucciones SQL necesarias para la creación de entidades.

1.5. CONEXIÓN DEL INTÉRPRETE CON MySQL

Una vez creados los archivos de jerarquía de clases y el archivo encargado del enlace con el motor de bases de datos es necesario realizar la conexión con este último. Para esto se requiere de un gestor de bases de datos capaz de operar junto con Java.

Aprovechando las ventajas del software libre y el licenciamiento GNU/GPL (tanto JFlex como CUP están bajo este tipo de licencia), se utilizo como gestor de bases de datos a MySQL 5.0 y como motor de almacenamiento InnoDB.

MySQL proporciona un servidor de bases de datos SQL (Structured Query Language) muy rápido, multi-threaded, multi usuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido[10].

Las características anteriores convierten a MySQL en una excelente solución para la implementación del intérprete desarrollado con los componentes ODL y OQL.

Por su parte entre las principales características de InnoDB están:

> ”InnoDB dota a MySQL de un motor de almacenamiento transaccional (conforme a ACID) con capacidades de commit (confirmación), rollback (reversión) y recuperación de fallas. InnoDB realiza bloqueos a nivel de fila y también proporciona funciones de lectura consistente sin bloqueo al estilo Oracle en sentencias SELECT. Estas características incrementan el rendimiento y la capacidad de gestionar múltiples usuarios simultáneos. No se necesita un bloqueo escalado en InnoDB porque los bloqueos a nivel de fila ocupan muy poco espacio. InnoDB también soporta restricciones FOREIGN KEY. En consultas SQL, aún dentro de la misma consulta, pueden incluirse libremente tablas del tipo InnoDB con tablas de otros tipos.

[10] Manual de Referencia MySQL 5.0. MySQL AB, 2006.

> InnoDB se diseñó para obtener el máximo rendimiento al procesar grandes volúmenes de datos. Su eficiencia en el uso de CPU probablemente no es igualada por ningún otro motor de bases de datos relaciones en disco.
>
> A pesar de estar totalmente integrado con el servidor MySQL, el motor de almacenamiento InnoDB mantiene su propio pool de almacenamiento intermedio para tener un caché de datos e índices en la memoria principal. InnoDB almacena sus tablas e índices en un espacio de tablas, el cual puede consistir de varios ficheros (o particiones disco). Esto difiere de, por ejemplo, el motor MyISAM, donde cada tabla se almacena empleando ficheros separados."[11]

Una vez definido el gestor de bases de datos y el motor de almacenamiento a utilizar se procedió a hacer la conexión entre MySQL y Java. Esto se logra gracias a MySQL Connector/J.

MySQL Connector/J es la solución implementada por MySQL para proporcionar conectividad a las aplicaciones clientes desarrolladas en Java vía el JDBC[12] driver. JDBC es una interfase de acceso a bases de datos estándar SQL que brinda un acceso uniforme a una gran variedad de bases de datos relaciónales. Para usar el JDBC con un sistema gestor de bases de datos en particular, es necesario disponer del driver JDBC apropiado que haga de intermediario entre este y el JDBC.

En síntesis el JDBC permite:

- Crear una conexión con la base de datos.
- Enviar sentencias SQL.
- Procesar los resultados.

La clase DriverManager es la encargada de establecer la conexión entre MySQL / Java. El DriverManager necesita saber que drivers JDBC debe usar para hacer la conexión. Una vez que el driver se ha registrado con el DriverManager, se puede obtener una instancia de Connection que está conectada a una base de datos particular llamando a DriverManager.getConnection(). A continuación se muestra un ejemplo de cómo hacer esta conexión:

---

11 Ibíd. Panorámica de InnoDB.

12 JDBC = Java DataBase Connectivity.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

try{
    Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost/prueba_sql","root",
    "root");
   // Utilizar la conexión establecida.
}
catch (SQLException ex) {
  System.out.println("SQLException: "+
                     ex.getMessage());

  System.out.println("SQLState: " +
                     ex.getSQLState());

  System.out.println("VendorError: " +
                     ex.getErrorCode());
}
```

Para más información sobre la conexión con MySQL ver el Anexo C (Clase Conexion.java).

1.5.1. Envío del archivo de definición de entidades a MySQL

Una vez establecido el procedimiento de conexión con MySQL el siguiente paso después de haber creado los archivos de jerarquía de clases y el archivo **entidades.sql** es enviar este último a MySQL para su ejecución.

Al momento de definir las clases se crean los archivos de clases (.LTC) y el archivo que contiene las entidades de dichas clases. Para este enlace se utilizó el siguiente comando:

```
mysql prueba_sql --user=root --password=root --execute
"source /ruta/entidades.sql"
```

Donde prueba_sql es una base de datos previamente creada. También se requiere para la conexión nombre de usuario y contraseña, seguida por la ruta del archivo entidades.sql.
Para más información sobre este procedimiento consultar el Anexo C (Clase Interfaz.java).

1.5.2. Implementación del comando “method”.

Realizada la definición de clases, creados los archivos LTC y hecha la conexión con MySQL por medio del archivo de **entidades.sql,** es decir, la implementación del ODL completa, es hora de utilizar el OQL para manipulación y consulta de las clases creadas.

Como se comentó en la definición de la gramática el OQL permite realizar consultas SQL (de una por vez) de tipo: **select, delete, update, insert, drop.** Estas consultas son recibidas por el intérprete, para luego realizar la conexión por medio del driver JDBC a MySQL.

Pero también el OQL permite el uso de los métodos propios de las clases definidas. Esto se hace por medio del comando **method**.

El procedimiento que realiza el intérprete para usar este comando se describe a continuación:

1. Analizar la entrada digitada por el usuario. (Análisis Léxico y Sintáctico).

2. Verificar que la clase a la cual se espera acceder, exista.

3. Contar los parámetros digitados por el usuario desde la entrada del intérprete.

4. Contar los parámetros usados por el método de la clase consultada.

5. Una vez verificado que el número de parámetros son iguales. Se procede a reemplazar los parámetros digitados por el usuario por los parámetros dentro del método de la clase consultada.

6. Hecho el reemplazo se crea la conexión con MySQL y se ejecutan los comandos SQL dentro del método de la clase consultada.

Para más información sobre la implementación del comando **method** ver el Anexo C (Clase Metodo.java).

1.6. IMPLEMENTACION DE LA INTERFAZ DE USUARIO

Siguiendo un estilo y un diseño tradicional, se ha creado una interfaz gráfica de usuario muy sencilla e intuitiva y sobre todo muy fácil de usar, bastante parecidas a los editores de textos más populares, WordPad y Bloc de notas.

La interfaz del intérprete de los componentes ODL y OQL, es básicamente un editor de texto que cuenta con las características necesarias para hacer definiciones y consultas sobre bases de datos bajo el nuevo paradigma manejado, “Reorientado a Objetos”. Permitiendo visualizar de manera sencilla y práctica el propósito del presente proyecto.

La interfaz sintetiza toda la parte del proyecto que debe ser tangible para el usuario final. El propósito de la creación de ésta, es el de proveer una interacción mas amigable entre la persona y el ordenador, mejorar la facilidad de uso del sistema generador de bases de datos y por supuesto, superar la típica interfaz de línea de comandos.

En la construcción de la interfaz del intérprete de los componentes ODL y OQL, no se utilizó ningún entorno de desarrollo visual, la realización de ésta se hizo en un editor de texto, dado que el código generado por algunas plataformas java, no era modificable. Explotando la característica de la Programación Orientada a Objetos (POO), la cual es inherente en java, se creó y estructuró el prototipo final de la aplicación.

Durante toda la fase de creación de la interfaz gráfica, el diseño se sesgo a ser totalmente consistente y coherente, usando y basándonos en el paradigma WIMP (ventanas, iconos, menús y dispositivos de interfaz humano), que está actualmente vigente en el grueso del mercado computacional. Por tanto, la aplicación permite el reconocimiento intuitivo a través de signos, normalmente familiares al usuario, facilita la manipulación de los símbolos de modo que su representación da orientación sobre qué tipo de objeto es y qué tipo de acciones podemos realizar con él, posibilita establecer relaciones lógicas entre datos que de otra forma serían complicadas de expresar, comprender y ejecutar.

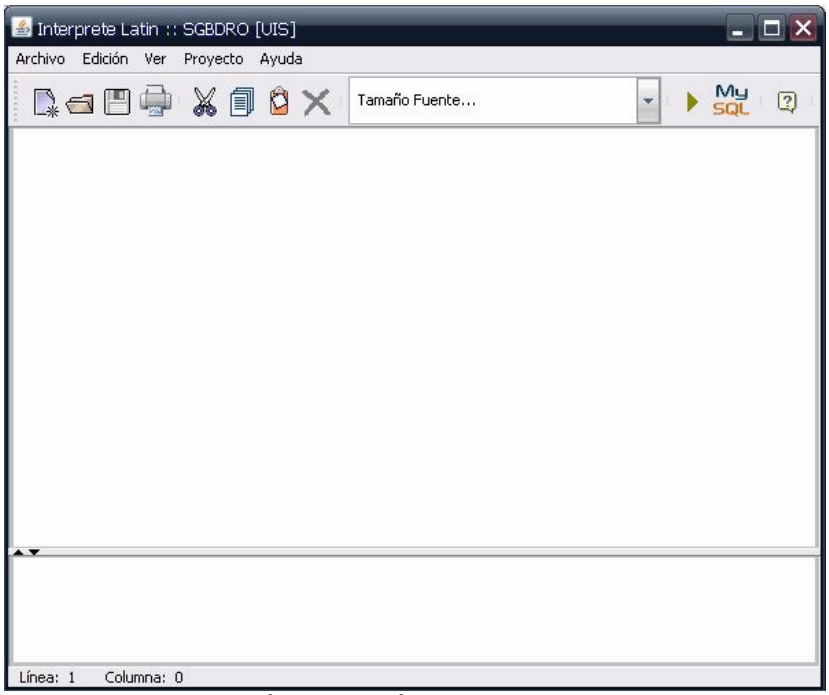


Figura No 10. Interfaz gráfica del intérprete de los componentes ODL y OQL

Se presenta una imagen en la que se ve el diseño global de la interfaz y resta expresar que solo se hizo un enfoque muy general sobre los atributos y operaciones de cada clase implementada en la aplicación.

1.6.1 Elementos Interactivos de la Interfaz gráfica

Los elementos de salida, tienen que ver con elementos que se han ido configurando para dar información de estado del sistema al usuario en un momento dado.

- **Ventana principal**

  El elemento interactivo vital de la aplicación es la ventana presentada en la Figura (Interfaz Gráfica del intérprete de los componentes ODL y OQL). Este marco contiene un conjunto de elementos funcionales que son el sustento de la aplicación. A parte de la ventana principal, la interfaz cuenta con otra serie de ventanas modales y de confirmación fundamentales para el funcionamiento correcto de la aplicación, que serán explicadas posteriormente.

- **Barra de menú**

  La lista de menús contextuales de esta aplicación contiene cinco agrupaciones de ítems como lo muestra la figura siguiente.

Figura No 11. Barra de menú de la interfaz grafica del intérprete

  - **Menú Archivo**
    Menú contextual que permite crear, abrir, guardar, imprimir un documento, así como salir de la interfaz.

    Formas de acceder al comando:
    Teclado: ATL + A

Figura No 12. Menú Archivo.

  - *Nuevo*: Éste ítem del menú Archivo permite crear un documento nuevo.

    Formas de acceder al comando:

    Barra de herramientas: 
    Teclado: CTRL + N
    A través del menú Archivo: ATL + A + N

    Al acceder a este comando, se muestra su cuadro de diálogo, donde advierte acerca de la pérdida de los cambios no guardados.

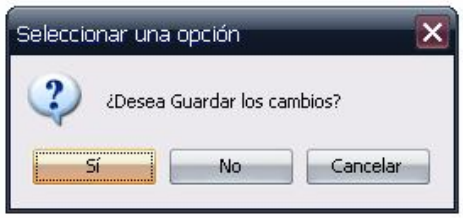


Figura No 13. Cuadro de dialogo del ítem Nuevo.

o *Abrir*: Este comando permite abrir un programa (conjunto de instrucciones) anteriormente creado y listo para editar.

Formas de acceder al comando:

Barra de herramientas: 
Teclado: CTRL + O
A través del menú Archivo: ATL + A + A

Al acceder a este comando, se muestra su cuadro de diálogo, para indicar la ruta donde se encuentra el archivo que se desea abrir.

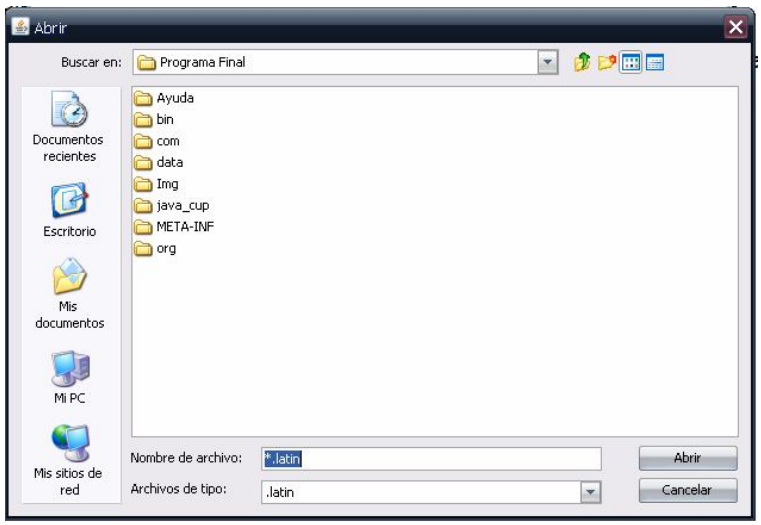


Figura No 14. Cuadro de diálogo del ítem Abrir.

o *Guardar*: Este comando permite guardar el archivo que se este editando.

Formas de acceder al comando:

Barra de herramientas: 
Teclado: CTRL + G
A través del menú Archivo: ATL + A + G

- *Guardar Como*: Este comando permite guardar el archivo que se este editando pero con otro nombre.

  Formas de acceder al comando:
  Teclado: CTRL + G
  A través del menú Archivo: ATL + A + O

  Al acceder a este comando, se muestra un cuadro de diálogo, donde se debe indicar el nombre y la ruta donde se desea guardar el archivo. Es importante tener en cuenta que el nombre del archivo se guarda por defecto con una extensión *.latin.

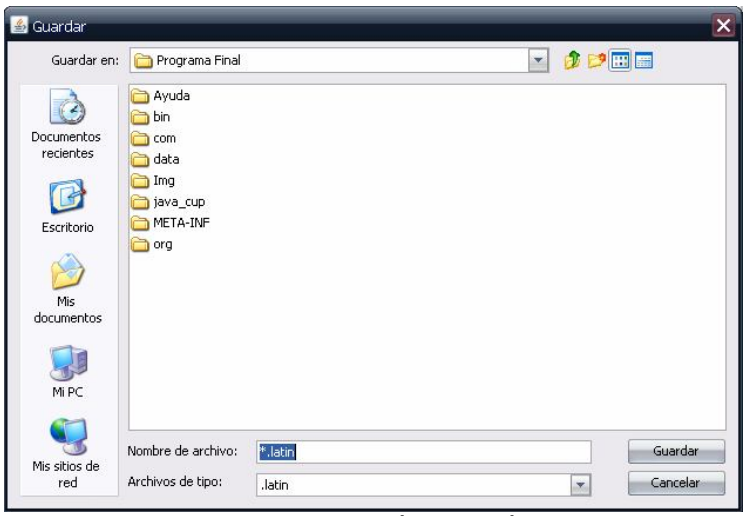


Figura No 15. Cuadro de diálogo del ítem Guardar.

- *Imprimir*: Este comando permite Imprimir el archivo que se esté editando, mediante el despliegue del cuadro de diálogo de impresión del sistema.

  Formas de acceder al comando:

  Barra de herramientas: 
  Teclado: CTRL + P
  A través del menú Archivo: ATL + A + I

  Al acceder a este comando, se muestra una ventana emergente, que nos permite configurar detalles básicos de la impresión como lo son: el intervalo de impresión, el número de copias y las propiedades de la impresora.

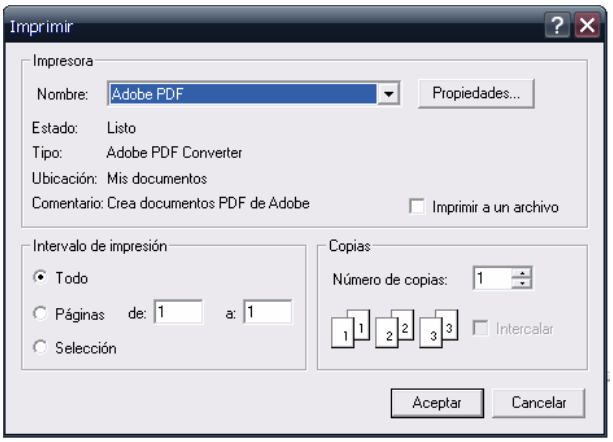


Figura No 16. Cuadro de diálogo del ítem Imprimir.

o *Salir:* Este comando permite abandonar la aplicación.

Formas de acceder al comando:

Teclado: ATL + F4
Mediante el icono de cerrar de la ventana activa.
A través del menú Archivo: ATL + A + L

Al acceder a este comando, se muestra un cuadro de diálogo, donde advierte acerca de la pérdida de los cambios no guardados.

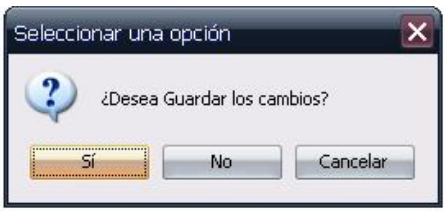


Figura No 17. Cuadro de dialogo del ítem Salir.

o **Menú Edición**
Menú contextual que permite cortar, copiar, pegar, seleccionar y hacer diferentes modificaciones sobre el documento de texto actual.

Formas de acceder al comando:
Teclado: ATL + E

Figura No 18. Menú Edición.

- *Cortar*: Este comando se usa para quitar el texto seleccionado del documento actual, y se lleva al portapapeles. Al utilizar este ítem, éste remplaza el contenido del portapapeles.

  Formas de acceder al comando:

  Barra de herramientas: 
  Teclado: CTRL + X
  Menú contextual del área de texto: Clic derecho + Cortar
  A través del menú Edición: ATL + E + C

- *Copiar*: Este comando permite obtener una copia de un segmento de texto seleccionado y pasarlo al portapapeles.

  Formas de acceder al comando:

  Barra de herramientas: 
  Teclado: CTRL + C
  Menú contextual del área de texto: Clic derecho + Copiar
  A través del menú Edición: ATL + E + O

- *Pegar*: Este comando permite insertar el segmento de texto que se encuentra en el portapapeles, en la posición seleccionada. El portapapeles contiene aún el segmento de texto que fue pegado, para poderlo pegar las veces que se desee.

  Formas de acceder al comando:

  Barra de herramientas: 
  Teclado: CTRL + V

Menú contextual del área de texto: Clic derecho + Pegar
A través del menú Edición: ATL + E + P

- *Eliminar*: Este comando se usa para quitar un segmento de texto seleccionado del área de texto de la interfaz.

  Formas de acceder al comando:

  Barra de herramientas: 
  Teclado: Suprimir
  Menú contextual del área de texto: Clic derecho + Eliminar
  A través del menú Edición: ATL + E + E

- *Seleccionar todo*: Este comando se usa para seleccionar, rápidamente, todo el texto del área de texto de la interfaz.

  Formas de acceder al comando:
  Teclado: CTRL + E
  Menú contextual del área de texto: Clic derecho + Seleccionar todo
  A través del menú Edición: ATL + E + T

- *Buscar*: Este comando se usa para hallar una determinada secuencia de caracteres que se encuentre dentro del área de texto.

  Formas de acceder al comando:

  Teclado: CTRL + B
  Menú contextual del área de texto: Clic derecho + Buscar
  A través del menú Edición: ATL + E + B

  Al acceder a este comando, se muestra un cuadro de diálogo, donde se introduce el texto que se desea buscar, si este es encontrado se selecciona sobre el área de texto y si no es hallado se presenta un dialogo de confirmación de que la secuencia de caracteres no fue localizada.

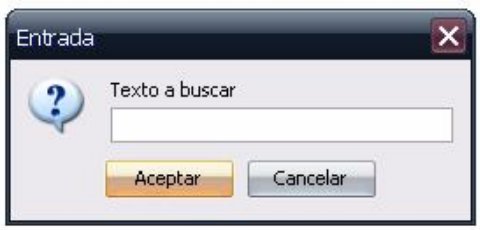


Figura No 19. Cuadro de dialogo del ítem Buscar.

Figura No 20. Cuadro de dialogo del ítem Buscar (palabra no encontrada).

- *Tamaño Fuente*: Este comando se usa para cambiar el tamaño de la fuente del área de texto al gusto del usuario.

  Formas de acceder al comando:

  Barra de herramientas: 

  Teclado: CTRL + F
  A través del menú Edición: ATL + E + F

  Al acceder a este comando, se muestra un cuadro de diálogo, donde se introduce el número del tamaño de la fuente que se desea, si se introduce por error un carácter alfabético el tamaño de la fuente no es modificado.

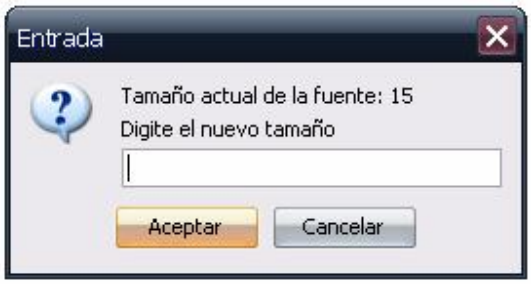


Figura No 21. Cuadro de dialogo del ítem Tamaño fuente.

- **Menú Ver**
  Menú contextual que permite modificar la ventana principal de la aplicación.

  Formas de acceder al comando:
  Teclado: ATL + V

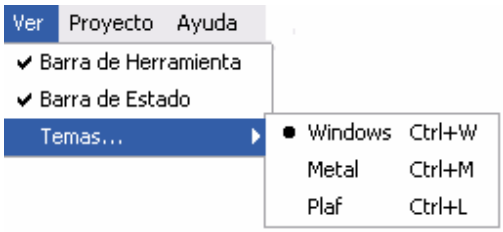


Figura No 22. Menú Ver.

- *Barra de Herramienta*: Este comando se usa para hacer visible o no la barra de herramientas de la ventana principal.

  Formas de acceder al comando:
  A través del menú Ver: ATL + V + H

- *Barra de Estado*: Este comando se usa para hacer visible o no la barra de estado de la ventana principal.

  Formas de acceder al comando:
  A través del menú Ver: ATL + V + H

- *Temas*: Este es un submenú del menú Ver que permite elegir uno de tres temas posibles para la interfaz grafica del SGBDRO. Cada tema elegido cambia tanto el aspecto de la ventana principal como el de las ventanas emergentes.

  Formas de acceder al comando:
  A través del menú Ver: ATL + V + T

  Al acceder a este comando, se muestra un submenú con tres opciones:

  - ✓ Windows: cambia el aspecto de la ventana al formato de Windows. es el tema asignado por defecto a la interfaz grafica.

    Formas de acceder al comando:
    Teclado: CTRL + W
    A través del menú Edición: ATL + V + T + W

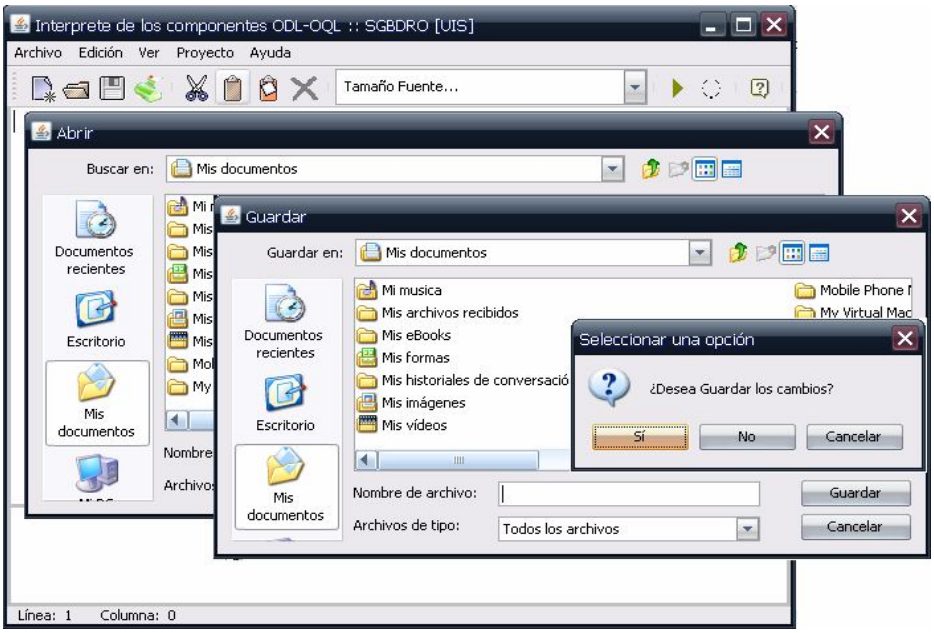


Figura No 23. Menú Ver (Tema: Windows).

- ✓ Metal: Cambia el aspecto de la ventana a formato metalizado

Formas de acceder al comando:
Teclado: CTRL + M
A través del menú Edición: ATL + V + T + M

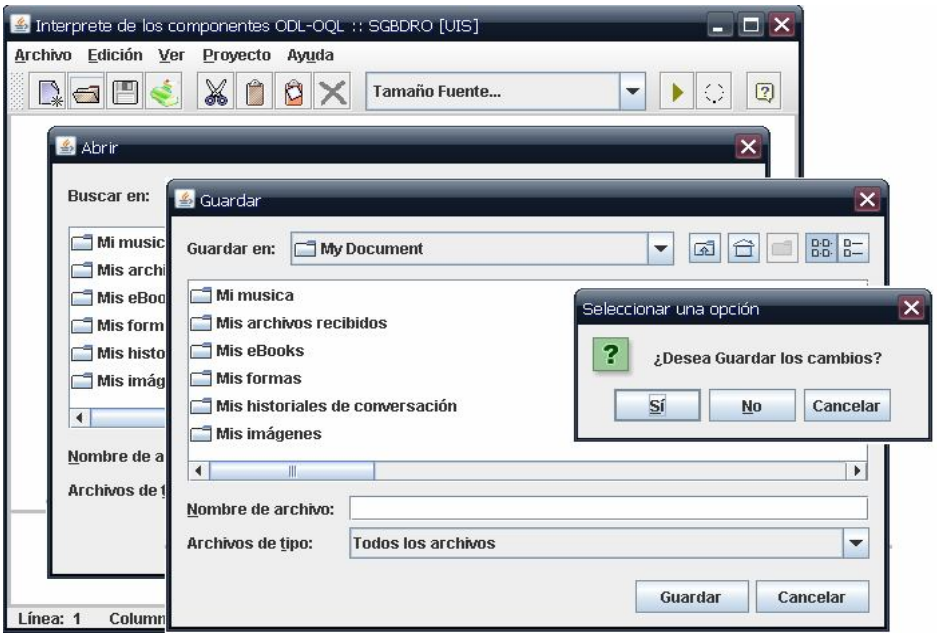


Figura No 24. Menú Ver (Tema: Metal).

✓ Plaf: cambia el aspecto de la ventana al formato de grises

Formas de acceder al comando:
Teclado: CTRL + L
A través del menú Edición: ATL + V + T + L

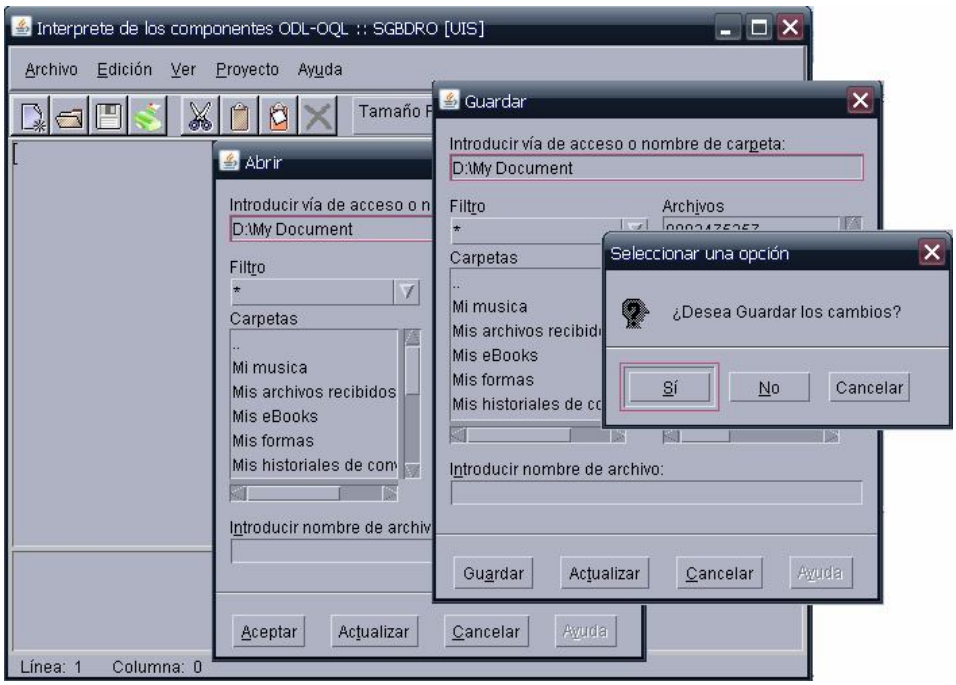


Figura No 25. Menú Ver (Tema: Plaf).

o **Menú Proyecto**

Menú contextual que permite ejecutar, graficar y cambiar variables de conexión en un conjunto de instrucciones que se encuentran en el área de texto de la interfaz. Una vez construidos el programa, se puede probar, si las sentencias implementadas responden a las necesidades del problema mediante la ejecución de la lógica del SGBDRO.

Formas de acceder al comando:
Teclado: ATL + P

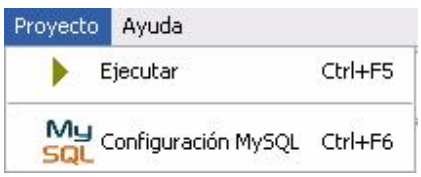


Figura No 26. Menú Proyecto.

- *Ejecutar*: Este comando se usa para ejecutar secuencialmente la lógica del programa escrito en el área de texto de la interfaz, la ejecución se realiza de manera completa.

  Formas de acceder al comando:

  Barra de herramientas:
  Teclado: CTRL + F5
  A través del menú Proyecto: ATL + P + E

  Al acceder a este comando, se muestra una ventana emergente de confirmación de éxito en la ejecución de todas las sentencias o mensajes de error en el área de texto inferior.

- *Conexión MySQL*: Este comando se usa para cambiar las variables de conexión con el servidor de bases de datos MySQL.

  Formas de acceder al comando:

  Barra de herramientas:
  Teclado: CTRL + F6
  A través del menú Proyecto: ATL + P + C

  Al acceder a este comando se muestra una ventana que permite cambiar las variables de conexión con MySQL como la base de datos a acceder, nombre de usuario y contraseña.

Figura No 27. Cuadro de dialogo del ítem: Configuración MySQL.

- **Menú Ayuda**
  Menú contextual que nos brinda información sobre el uso del uso del programa

  Formas de acceder al comando:
  Teclado: ATL + U

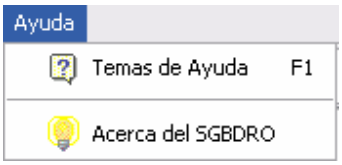


Figura No 28. Menú Ayuda.

- *Temas de Ayuda*: Este comando es usado para acceder a la ayuda del software e información de cómo definir variables, entidades, clases, etc.

  Formas de acceder al comando:

  Barra de herramientas:
  A través del menú Edición: ATL + U + T

- *Acerca del SGBDRO*: Este comando se usa para mostrar una ventana con el nombre del proyecto y el se su director, los nombres de los desarrolladores y el de la universidad.

  Formas de acceder al comando:
  A través del menú Edición: ATL + U + A

- **Barra Herramienta**

  Consiste en una barra dispuesta de forma horizontal en la parte superior de la ventana principal, debajo de la barra de menús. Ofrece un conjunto de iconos que activan operaciones sobre la aplicación.

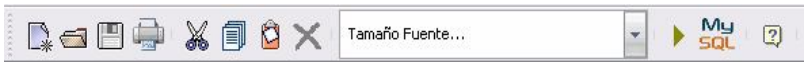


Figura No 29. Barra de Herramientas de la Interfaz Grafica.

- **Área de Texto**

  Es la región más grande de la interfaz gráfica. Delimita un área en blanco, redimensionadle por la ventana principal y por las barras de desplazamiento emergentes cuando son necesarias. La mayoría de las operaciones que se realizan con los menús, están relacionadas con éste área de texto.

  El Área de texto esta ligada con un menú contextual emergente, que ofrece algunas de los ítems del menú ver. Para tener acceso a este menú se debe hacer clic derecho sobre cualquier parte del área de texto.

Figura No 30. Menú ligado al área de texto.

- **Elementos de Información de Salida**

  Los elementos de salida, tienen que ver con elementos que se han ido configurando para dar información de estado del sistema al usuario en un momento dado.

- **Tip Box**

  Es un recurso que surge en ciertos elementos de la interfaz para indicar información adicional sobre algún elemento u acción del usuario sobre el sistema.
  La forma de acceder a este recurso es posicionar el puntero sobre el elemento de la interfaz sobre el que se desee más información.

- **Área de errores**

  En la parte inferior de la ventana principal, se encuentra un área de texto en donde se muestran los errores detectados luego de hacer la ejecución de las instrucciones ingresadas en el área de texto principal.

- **Barra de estado**

  La barra de estado ofrece información al usuario sobre la posición del cursor en el área de texto. Posicionada en la parte inferior de la ventana de aplicación.

Figura No 31. Barra de Estado de la Interfaz Grafica.

2. PRUEBAS DE LOS COMPONENTES

Durante el desarrollo de cada uno de los componentes del intérprete para el MODRO se aplicaron las pruebas respectivas según las necesidades del mismo. El procedimiento seguido para realizar este tipo de pruebas fue:

- Establecer previamente su ejecución.
- Realizar evaluación de un componente.
- Observar y registrar los resultados.

Considerando lo anterior se realizaron las pruebas de acuerdo al proceso mostrado en la Figura 32.

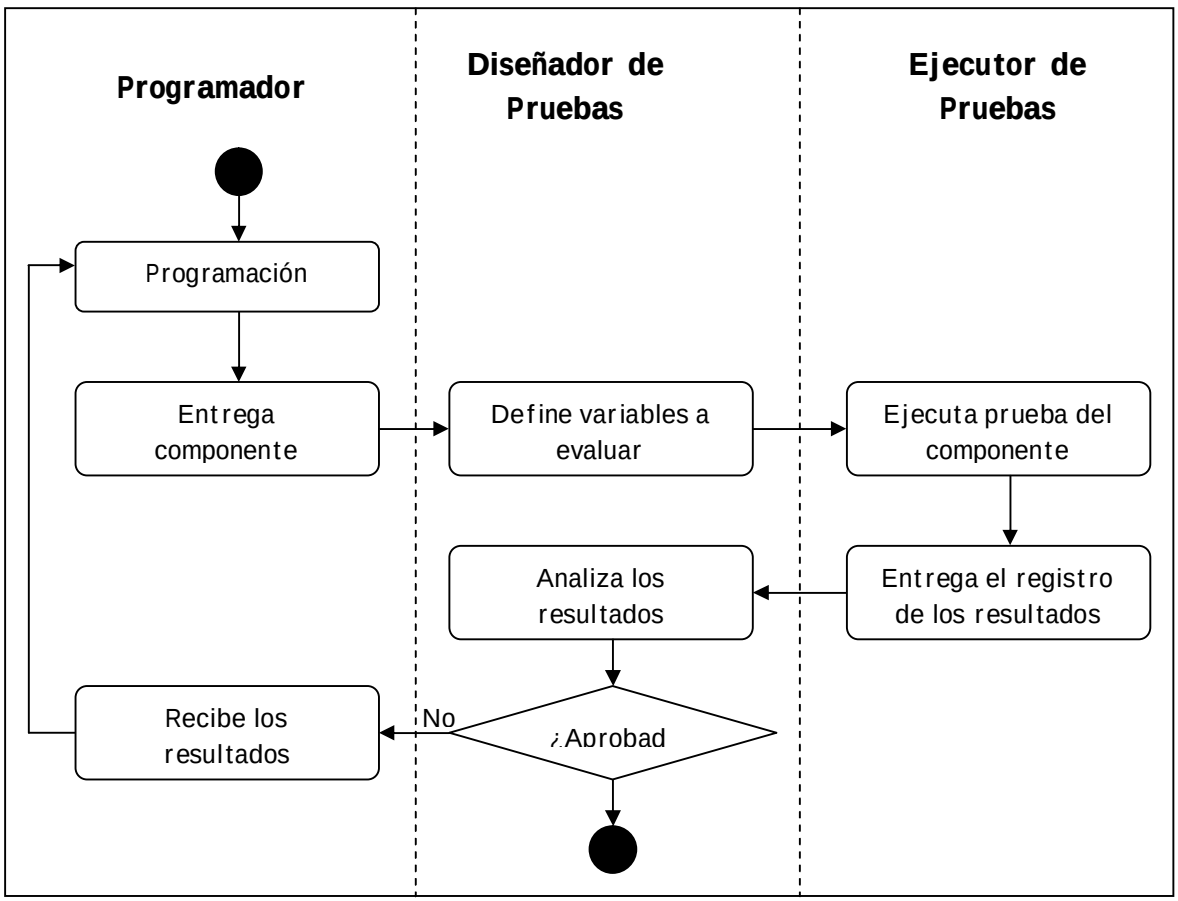


Figura No 32. Proceso de Pruebas.

2.1. IMPLEMENTACION Y RESULTADO DE LAS PRUEBAS

A continuación se describen los componentes del intérprete y las variables tenidas en cuenta para la ejecución de las pruebas.

2.1.1. Prueba Analizador Léxico

Una vez el generador de analizadores léxico JFlex crea la clase **lexer.java** se procede a compilar y ejecutar el analizador.

Se realizaron tres tipos de pruebas para el analizador léxico. La primera consistió en ingresar una serie de cadenas de caracteres válidos por la gramática definida para los componentes ODL y OQL. Para la segunda prueba se ingresaron cadenas de caracteres no permitidos dentro de la gramática. Y por último se ingresaron cadenas de caracteres válidos e inválidos.

Los resultados obtenidos en las pruebas mostraron un gran desempeño en tiempo y validación del analizador léxico para las cadenas ingresadas.

2.1.2. Prueba Analizador Sintáctico

Una vez el generador de analizadores sintácticos CUP crea las clases **sym.java parser.java** se procede a compilar y ejecutar el analizador. Para ello se hizo necesario crear la siguiente clase:

```
public class Main {
  static public void main(String argv[]) {
    /* Inicia el analizador */
    try {
    parser p = new parser
     (new Lexer
     (new FileReader(argv[0])));
      Object result = p.parse().value;
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

La cual recibe como parámetro un archivo de texto plano con las cadenas digitadas por el usuario y realiza la conexión entre el analizador léxico y sintáctico.

Para el diseño de estas pruebas se consideró la sintaxis de los componentes ODL y OQL. Se realizaron un conjunto de entradas para comprobar el ODL, definiendo clases, subclases, entidades y métodos. De igual manera se realizaron entradas para comprobar el OQL definiendo sentencias SQL **select, update, insert, drop** así como también el uso de los métodos de las clases con el comando **method.**

Las pruebas realizadas a cada uno de los componentes del intérprete del MODRO permitieron verificar la efectividad de los analizadores léxico y sintáctico.

2.1.3. Prueba de creación de archivos de jerarquía de clases.

La estructura de los archivos de jerarquía de clases permite comprobar que éstos han sido creados de manera satisfactoria por el intérprete de los componentes ODL y OQL.

La prueba aplicada para verificar que la creación de éstos archivos es correcta fue definir un conjunto de clases y subclases para luego constatar en los archivos de clases creadas la estructura de los mismos. Esta prueba permitió confirmar la correcta creación de la estructura de los archivos de jerarquía de clases del intérprete.

2.1.4. Prueba de conexión con MySQL.

La conexión con el servidor de bases de datos MySQL es realizada por el intérprete durante la definición de clases, así como también al momento de realizar las consultas SQL y en la implementación de los métodos.

Para la realización de las pruebas de conexión con MySQL se utilizó la herramienta PHPMyAdmin Database Manager v2.9.2, la cual brinda un entorno gráfico para la administración de las bases de datos de MySQL.

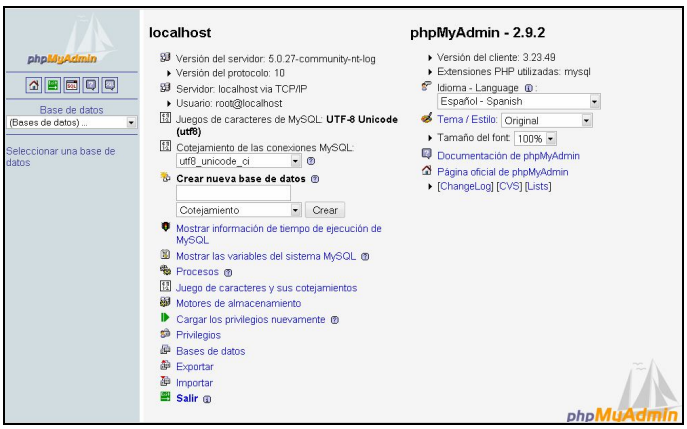


Figura No 33. Interfaz Grafica de PHPMyAdmin 2.9.2.

PHPMyAdmin permitió verificar que la creación de las clases, las consultas utilizadas y la implementación de los métodos se realizara de manera satisfactoria.

Se aplicaron pruebas de conexión para la definición de clases, consultas SQL e implementación de los métodos. Para más información sobre uso del interprete ver el capítulo 3.

Las pruebas permitieron verificar que la conexión con el servidor de bases de datos MySQL se realiza de manera correcta y efectiva.

2.1.5. Prueba de la interfaz de usuario.

Las pruebas realizadas a la interfaz de usuario se desarrollaron conjunto a las pruebas realizadas anteriormente, esto debido a la constante interacción con la misma al momento de realizar las pruebas. Permitiendo así un chequeo constante de los elementos desarrollados para la interfaz.

Se aplicaron pruebas de ejecución con cada uno de los botones de la barra de herramientas. Así como también de captura de información desde el área de texto principal.

En cada prueba desarrollada sobre la interfaz se lograba mejorar la experiencia del usuario al momento de interactuar con el interprete mostrando en cada mejora mayor intuición en su presentación.

2.1.6. Prueba general del intérprete de los componentes ODL y OQL.

Para la implementación de las pruebas generales del intérprete se utilizaron los ejemplos desarrollados por el profesor Albarracín en su tesis doctoral. Más información sobre estos en el capítulo 3.

Las pruebas generales se realizaron bajo los siguientes sistemas operativos:

- Windows XP SP2. 32-Bits.
- Windows Vista Ultimate. 32-Bits.
- Linux Ubuntu 7.04

En cada uno de los sistemas operativos anteriores se logro ejecutar de manera satisfactoria el intérprete.

Estas pruebas permitieron observar el desempeño del intérprete de los componentes ODL – OQL y mostrar de manera satisfactoria la implementación de un MODRO sencillo en un sistema computacional.

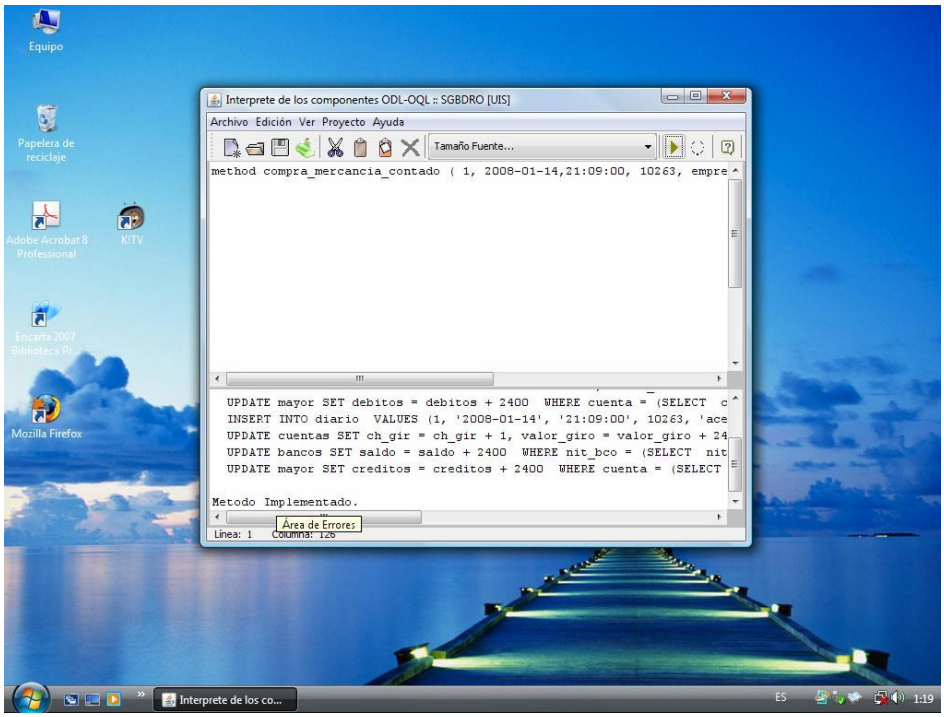


Figura No 34. Implementación del Intérprete en Windows Vista Ultimate.

3. USO DE LOS COMPONENTES ODL Y OQL EN LA IMPLEMENTACION DE UN MODRO

3.1 REQUISITOS DEL SISTEMA

3.1.1. Requisitos Software

Para la ejecución del intérprete de los componentes ODL y OQL se necesita lo siguiente:

- Servidor de Bases de Datos MySQL 5.0. (Con el motor de almacenamiento InnoDB como predeterminado).
- Java SE Runtime Environment (JRE) Versión 6.
- Sistema Operativo: Windows de 32 bits (XP, Vista).

3.1.2. Requisitos Mínimos de Hardware

- Procesador Intel Pentium III (o equivalente) de 933MHz o superior.
- 512 MB de memoria RAM o superior.
- 1 GB de espacio libre en disco o superior.

3.2 EJEMPLO No 1[13]

Supóngase una instancia de la transacción "*compra al contado, nacional, de mercancía para la venta*" la cual podría ser la transacción de código 1, que evolucionaría a través de tres áreas funcionales: Por el Almacén (actualizando el auxiliar de mercancías), por Tesorería (actualizando el auxiliar de bancos) y por Contabilidad (actualizando los libros diario y mayor), suponiendo que el pago se hace al momento de recibir la mercancía y que no hay IVA ni otros conceptos, para simplificar. Se subraya que las áreas (entidades del MODRO) que participan, así como la forma en que evolucionan las transacciones, están determinadas por las reglas del negocio particulares de cada

[13] Los ejemplos 1 y 2 se encuentran en la tesis doctoral del profesor Albarracin. Pág. 65.

Organización. Además se supone que la transacción de este ejemplo ha sido compactada mediante Reingeniería (ha sido rediseñada) y ha sido también computarizada, por lo que su evolución ocurre dentro del hardware y corresponde a un solo programa de computador y solo uno. No puede corresponder a varios programas porque esto equivaldría a mantener fragmentada la transacción, contradiciendo el espíritu de la Reingeniería.

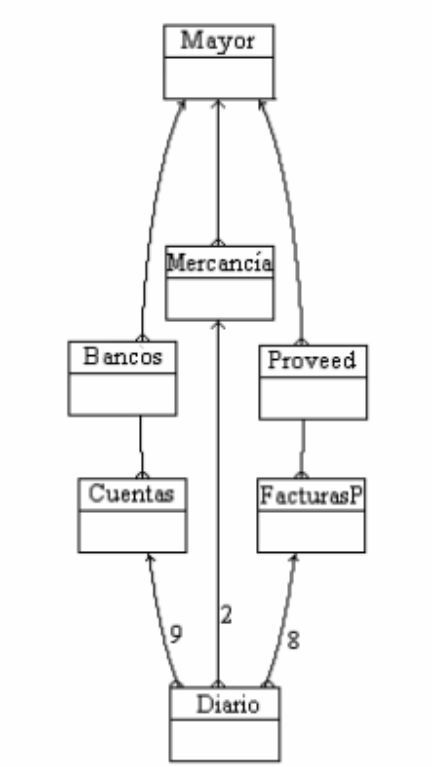


Figura No 35. Tipo compras.

Por otro lado, una Organización no procesa abstracciones sino transacciones concretas, por lo que también hay que suponer que la transacción en cuestión ocurrió por la compra de 200 cuchillos por $2400 (a $12 c/u), en la ferretería “Aceros”, a quien se le pagó con un cheque del banco Nacional. Sin embargo, hay que advertir que mientras para los señores de Contabilidad, dicha transacción se reduce a los asientos contables mostrados adelante, para este ejemplo significa la actualización de la Base de datos, tabla por tabla (área por área), en términos de cpu, como se muestra detalladamente en la figura No 37. Aunque en términos de usuario de carne y hueso, la transacción sucede por completo y al instante, gracias primero a la Reingeniería y segundo gracias a la informática.

El MODRO está en realidad compuesto de conjuntos referenciales, cada uno de los cuales constituyen un tipo y representa a un área de la empresa. Así, el área de compras está pues representada por el tipo compras (véase figura No 35), el cual se

implementa mediante la "*clase compras*" y las subclases "*compra nacional de mercancía al contado*" y las otras modalidades de compra que la empresa ejecute, entre las cuales podría estar la "*compra nacional de mercancía a plazos*".

Pero para simplificar el tipo y la definición de la clase y de las subclases, se ignoran deliberadamente otras variedades de compras posibles, como se puede intuir de la figura No 36, en donde existe la entidad "Mercancía" pero no existen entidades del MODRO tales como "Seguros" y "Vehículos", los cuales también son susceptibles de ser comprados y deberían estar presentes abriendo posibilidades de otras modalidades de compras. Excepto que la compra de seguros y/o de vehículos esté a cargo de un área diferente, según lo señalen las reglas del negocio.

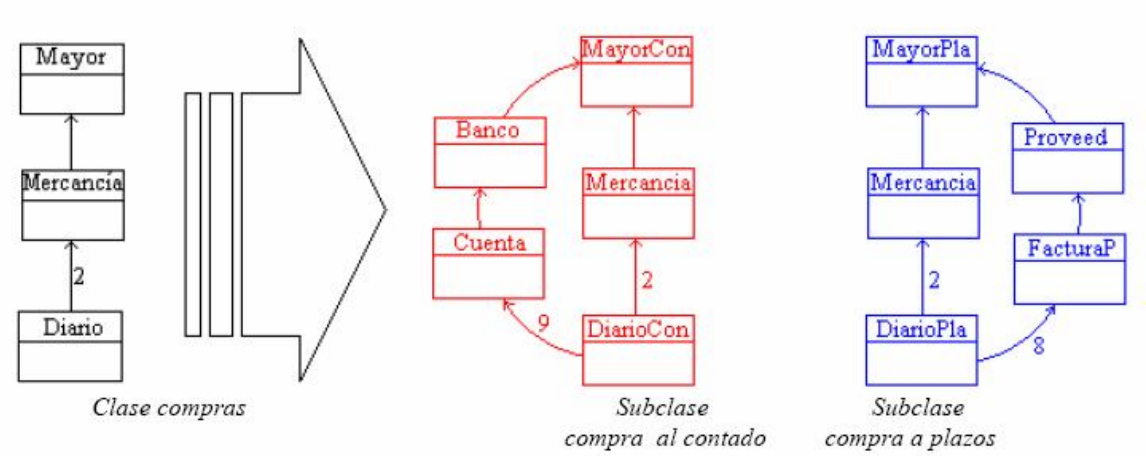
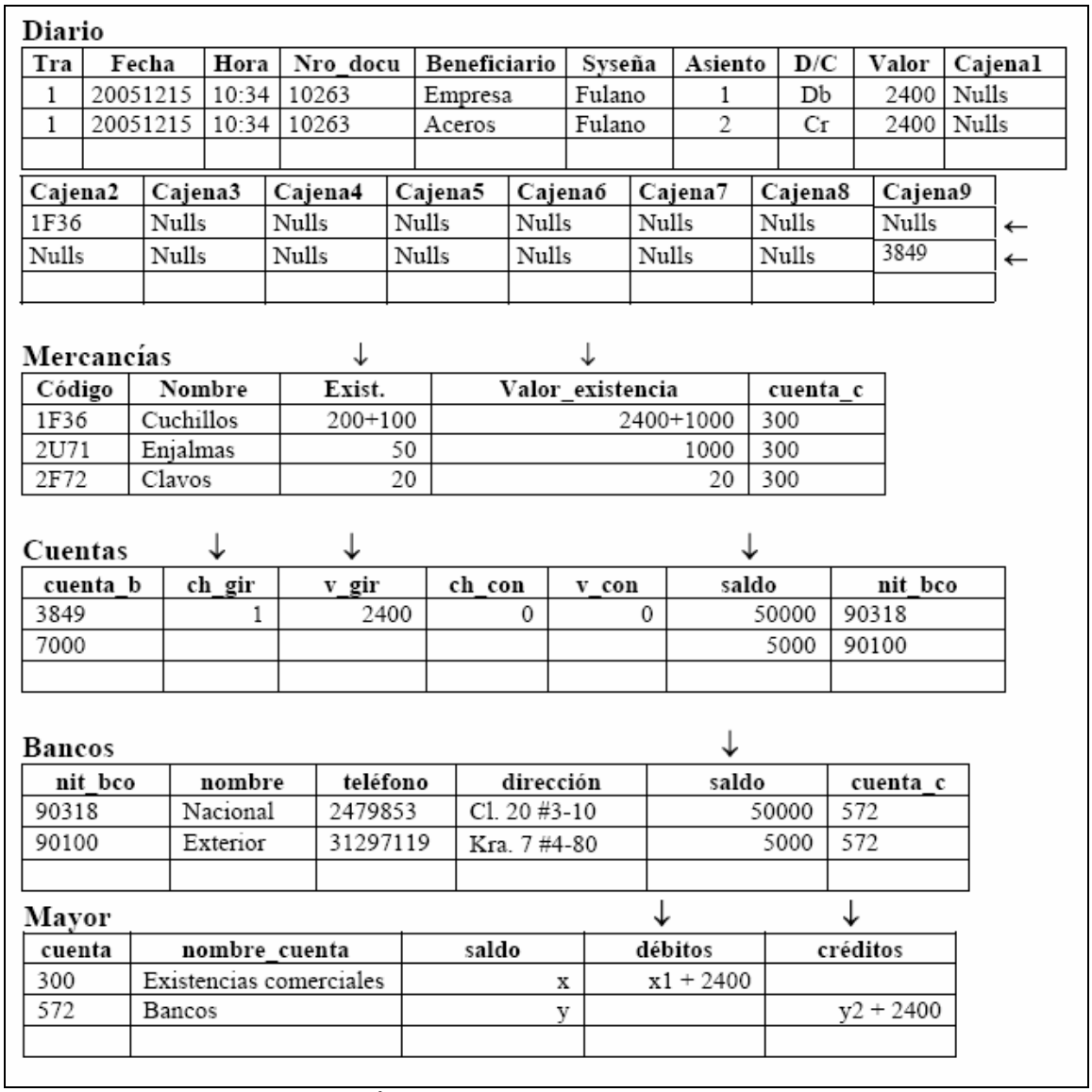


Figura No 36. Implementación del tipo compras.

Volviendo a la transacción del caso, lo que frecuentemente hacen las empresas (aún mediante computadores) es aplazar el registro de los asientos contables en la sección de Contabilidad después del registro de los demás datos en las otras secciones, según la naturaleza de la transacción ejecutada. Aquí por el contrario, un único programa de computador asumirá en la misma ejecución el registro íntegro de toda la transacción invocándolo como un método, sea "*comprar_al_contado_mercancía_nacional* ()" el nombre de tal método, respetando así la naturaleza atómica de la transacción. Para esto comienza a navegar por el MODRO, por las rutas 2 y 9, ejecutando operaciones más elementales y primitivas con las cuales se compone el método invocado. Primero realiza un INSERT en la Bitácora o libro Diario, para cada uno de los siguientes asientos contables, generando una dupla por cada uno.

Código Cuenta	**Nombre Cuenta**	**Débito**	**Crédito**
300	Existencias comerciales	2400	
572	Bancos		2400

Se ve así que el “Diario” (figura No 37) contiene dos registros insertados (apuntados por las dos flechas horizontales de la extrema derecha), cuya fecha es el 15 de diciembre de 2005 y cuya hora es 10:34: Uno por cada asiento y cada uno por $2400. El primer registro es el asiento débito sobre la cuenta mayor 300, “Existencias comerciales” y el segundo es el asiento crédito sobre la cuenta mayor 572, “Bancos”. Nótese, sin embargo, que estas cuentas no aparecen en el “Diario”. ¿Cómo sabe entonces el programa que estas son las cuentas a debitar y a acreditar y no otras?



Diario

Tra	Fecha	Hora	Nro_docu	Beneficiario	Syseña	Asiento	D/C	Valor	Cajena1
1	20051215	10:34	10263	Empresa	Fulano	1	Db	2400	Nulls
1	20051215	10:34	10263	Aceros	Fulano	2	Cr	2400	Nulls

Cajena2	Cajena3	Cajena4	Cajena5	Cajena6	Cajena7	Cajena8	Cajena9	
1F36	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	3849	←

Mercancías

Código	Nombre	Exist.	Valor_existencia	cuenta_c
1F36	Cuchillos	200+100	2400+1000	300
2U71	Enjalmas	50	1000	300
2F72	Clavos	20	20	300

Cuentas

cuenta_b	ch_gir	v_gir	ch_con	v_con	saldo	nit_bco
3849	1	2400	0	0	50000	90318
7000					5000	90100

Bancos

nit_bco	nombre	teléfono	dirección	saldo	cuenta_c
90318	Nacional	2479853	Cl. 20 #3-10	50000	572
90100	Exterior	31297119	Kra. 7 #4-80	5000	572

Mayor

cuenta	nombre_cuenta	saldo	débitos	créditos
300	Existencias comerciales	x	x1 + 2400	
572	Bancos	y		y2 + 2400

Figura No 37. Actualización de tablas de la base de datos en la compra al contado.

Precisamente la propia estructura del MODRO tiene la respuesta. Porque para poder acceder al registro del archivo “Mayor” cuya clave es 300, “Existencias comerciales”, primero se debe acceder al registro cuya clave es F36, cuchillos, en el archivo “Mercancías”. Por eso en el primer registro del “Diario” aparece como clave ajena 2 (“Cajena2”) el código de mercancía F36 y no el código de cuenta mayor 300. Porque de acuerdo al MODRO, la ruta referencial número 2, está formada por las tablas “Diario”→”Mercancías”→”Mayor”, lo que significa que para llegar hasta el “Mayor” debo pasar primero por “Diario” y por “Mercancías”. El archivo de “Mercancías” es auxiliar del archivo “Mayor” ya que explica y detalla los datos generales contenidos en él, en la cuenta 300. Si por ejemplo, desde el período contable anterior ya existían los 100 cuchillos, las 50 enjalmas y las 20 cajas de clavos, el valor del saldo “x” en el “Mayor” sería 2020, al de ocurrir la presente transacción.

En general, el MODRO obliga a que los programas se muevan de lo detallado a lo general por cada ruta referencial, desde el “Diario”, el receptáculo más extendido en detalles, hasta el “Mayor”, el receptáculo más resumido y conciso, igual a como sucede en la realidad. Así cada asiento sigue su propia ruta referencial, metiéndose primero en el “Diario”, pasando luego por el auxiliar respectivo y finalmente llegando al “Mayor”, siempre de sur a norte, de polo a polo. De este modo, si una transacción se contabiliza con dos asientos, se moverá por dos rutas referenciales, si con tres se moverá por tres y así sucesivamente.

El funcionamiento para el otro asiento, el de la cuenta 572 es similar. Aquí también antes de acceder al registro del “Mayor” cuya clave es 572, “Bancos”, debo acceder en la tabla “Cuentas” al registro con clave 3849, que es la cuenta bancaria de donde se ha girado el cheque 10263. Por eso en el “Diario” aparece como clave ajena 9 (“Cajena9”) el número de la cuenta bancaria 3849, y no el código de cuenta mayor 572. En el MODRO la ruta referencial número 9 está formada por las tablas “Diario”→”Cuentas”→”Bancos”→”Mayor”, lo que significa que para llegar hasta el “Mayor” debo pasar primero por el “Diario”, por “Cuentas” y por “Bancos”. El “Diario” contiene todos los cheques girados de cada cuenta bancaria. “Cuentas” contiene los saldos de cada cuenta bancaria y “Bancos” el saldo total de todas las cuentas tenidas en cada banco. Finalmente “Bancos” es el auxiliar del “Mayor” ya que explica y detalla los saldos globales contenidos en la cuenta mayor 572. Por ejemplo, si desde el período contable anterior esta compra es la primera compra, el saldo “y” en el “Mayor” sería 55000 pesos.

La estructura esencial del “Diario” es la misma de un libro diario como el usado en la Contabilidad general de cualquier empresa, excepto que no contiene directamente las cuentas de Contabilidad sino las claves ajenas a través de las cuales se llega a ellas en el “Mayor”. Esta estrategia de claves ajenas a través de las cuales se conocen las cuentas mayores de Contabilidad, funciona lo mismo para los auxiliares como “Mercancías”, “Cuentas” y “Bancos”, y le da cohesión al MODRO. Pero también la estructura esencial del “Mayor” es la misma del libro mayor usado en la Contabilidad general de cualquier empresa.

El MODRO en todo momento está expresando lo relacionado con las transacciones sucedidas en la Organización, desde los más finos detalles, hasta la generalidad requerida por los altos niveles administrativos. Su lectura general, de muchos a uno, al menos por las dos rutas referenciales usadas en este ejemplo, es como sigue. Por la ruta dos: En el “Diario” hay varios asientos que corresponden a un artículo negociado en una transacción, y a la forma de negociación. Además en la tabla de “Mercancías” muchos artículos se totalizan en una sola cuenta mayor, la de inventarios en el “Mayor”. Por la ruta nueve: En el “Diario” hay varios asientos que están relacionados con una sola cuenta bancaria, expresando sus movimientos detallados. Muchas cuentas bancarias en la tabla de “Cuentas” pueden pertenecer a un mismo banco y sus saldos sumados son el saldo del banco respectivo. El saldo de cada banco es a su vez totalizado en una sola cuenta mayor, la de “Bancos”, en el “Mayor”.

Sobre cada tabla mostrada en la figura No 37 se ven unas flechitas apuntando verticalmente hacia abajo (excepto en el “Diario”) y otras apuntando horizontalmente hacia la izquierda en el “Diario”, mostrando las primeras los campos de cada tabla que han sido actualizados (UPDATE) por la transacción y las segundas los registros insertados (INSERT) por la transacción.

Si la compra considerada en esta transacción involucrara a más artículos además de los cuchillos, por cada artículo se generaría un asiento en el “Diario” y por cada asiento de estos el programa navegaría repetidamente por la ruta referencial número 2, tantas veces como artículos se hubiesen comprado. De este modo, la funcionalidad del MODRO conlleva a mayor exactitud que en la contabilidad convencional que se suele llevar en las empresas.

A continuación se muestra el ODL para las clases “compra mercancía” y “compra mercancía de contado”.

```
/*
Implementación del MODRO. Ejemplo 1 extraído del capitulo 5 de la
tesis doctoral del profesor Jaime Albarracín.
*/
class universal
{
  entity mayor
  {
   cuenta                varchar(5) not null primary key,
   nombre_cuenta         varchar(30) not null,
   saldo                 decimal(10,2)not null,
   debitos               decimal(8,2)not null,
   creditos              decimal(8,2)not null
  }
```

```
  entity diario
  {
   tra                  varchar(3) not null,
   fecha                date not null,
   hora                 time not null,
   nro_docum            varchar(8) not null,
   beneficiario         varchar(30),
   sysena               varchar(10),
   asiento              varchar(2) not null,
   d_c                  varchar(1),
   valor                decimal(8,2) not null
  }
  method universal()
  {
  }
};

// Definición Clase Compra_Mercancia
class compra_mercancia extend universal
{
/*Inicia la definicion de Entidades pertenecientes a
la Clase Compra_Mercancia */
  entity mercancia
  {
   codigo               varchar(5) not null primary key,
   nombre               varchar(30) not null,
   existencia           integer not null,
   valor_existencia     decimal(10,2) not null,
   cuenta_c             varchar(5) not null,
   foreign key(cuenta_c) references mayor(cuenta)
  }
  entity diario
  {
   cajena2              varchar(5),
   foreign key(cajena2)  references mercancia(codigo)
  }
  /* Finaliza la definicion de Entidades pertenecientes a
  la Clase Compra_Mercancia */
  method comprar
  // Parametros Entidad Diario
  (cod_trans__,fecha__,hora__,nro_docum__,beneficiario__,
   sysenia__,asiento__,deb_cred__,valor__,cod_mercancia__,
  // Parametros Entidad Mercancia
  cantidad__)
```

```
 {
   insert into diario values
  (&cod_trans__,'fecha__','hora__',&nro_docum__,
  'beneficiario__','sysenia__',&asiento__,
  'deb_cred__',&valor__,'cod_mercancia__');

   update mercancia set
   existencia = existencia + cantidad__,
   valor_existencia = valor_existencia + valor__
   where codigo = 'cod_mercancia__';

   update mayor set
   debitos = debitos + valor__  where cuenta = (
   select cuenta_c from mercancia
   where codigo = 'cod_mercancia__' ) ;
 }
};

// Definicion Clase Compra_Mercancia_Contado
class compra_mercancia_contado extend compra_mercancia
{
 entity bancos
 {
  nit_bco               varchar(14) not null primary key,
  nom_banco             varchar(30) not null,
  telefono              varchar(15),
  direccion             varchar(30),
  saldo                 decimal(10,2),
  cuenta_c              varchar(5) not null,
  foreign key(cuenta_c) references mayor(cuenta)
 }
 entity cuentas
 {
  cuenta_ban            varchar(15) not null primary key,
  cheque_gir            smallint not null,
  valor_giro            decimal(10,2) not null,
  cheque_con            smallint,
  valor_con             decimal(10,2),
  saldo                 decimal(10,2) not null,
  nit_bco               varchar(14) not null,
  foreign key(nit_bco)  references bancos(nit_bco)
 }
```

```
 entity diario
 {
  cajena9                varchar(15),
  foreign key(cajena9)   references cuentas(cuenta_ban)
 }
 method comprar
 // Parametros Entidad Cuentas
 (ch_gir__,beneficiario2__,asiento2__,deb_cred2__,cuenta_ban__ )
 {
   insert into diario values
   &cod_trans__,'fecha__','hora__',&nro_docum__,'
   beneficiario2__','sysenia__',&asiento2__,
   'deb_cred2__',&valor__,null,&cuenta_ban__);

   update cuentas set
   ch_gir = ch_gir + ch_gir__,
   valor_giro = valor_giro + valor__,
   saldo = saldo - valor__
   where cuenta_ban = cuenta_ban__;

   update bancos set
   saldo = saldo - valor__ where nit_bco = (
   select nit_bco from cuentas where cuenta_ban = cuenta_ban__) ;

   update mayor set
   creditos = creditos + valor__ where cuenta = (
   select cuenta_c from bancos where nit_bco = (
   select nit_bco from cuentas where cuenta_ban = cuenta_ban__));
 }
};
```

El comando a utilizar para implementar el método de la clase compra de contado:

```
method compra_mercancia_contado ( 1, 2008-01-14,21:09:00, 10263,
empresa, fulano, 1, d, 2400, f36, 200,1, aceros,2,c,3849);
```

Pero antes debe estar ingresada la siguiente información en la base de datos:

```
insert into mercancia values ('f36','cuchillos', 100, 1000, 300);
insert into mercancia values ('U71','enjalmas', 50, 1000, 300);
insert into mercancia values ('F72','clavos', 20, 20, 300);
insert into cuentas values (3849, 1, 2400, 0, 0, 50000, 90318 );
insert into cuentas values (7000, 0, 0, 0, 0, 5000, 90100);
```

```
insert    into    bancos    values    (    90318,    'Nacional',
2479853,'Calle20No3_10', 50000, 572  );
insert into bancos values ( 90100, 'Exterior', 31297119,
 'Carrera20No4_80', 5000, 572  );
insert into mayor values (300,'Existencias_Comerciales',100,0,0);
insert into mayor values ( 572,'Bancos',100,0,0 );
```

3.3 EJEMPLO No 2

Otra circunstancia que ayudará a esclarecer más el funcionamiento del MODRO, se presenta cuando una supuesta transacción se interrumpe para diferir en el tiempo algunas partes. En este caso realmente no se trata de una, sino de dos o más transacciones. Por ejemplo, si en la compra que se acaba de ver, su pago no se hubiera realizado al momento de recibir la mercancía sino en una fecha posterior, se originaría otra transacción complementaria, que podría ser el "*Pago de facturas a proveedores nacionales*". Entonces, al momento de recibir la mercancía no se afectarían los bancos sino las cuentas por pagar a los proveedores, siendo los asientos los siguientes:

Código Cuenta	**Nombre Cuenta**	**Débito**	**Crédito**
300	Existencias comerciales	2400	
400	Proveedores		2400

Esto quiere decir que ahora la transacción, por decir algo la transacción 0, sería la "*Compra a crédito nacional de mercancía*", que es otra subclase de compra y se moverá por las rutas referenciales 2 y 8 en lugar de hacerlo por las rutas 2 y 9 como antes. Por lo tanto, ahora el "Diario" tiene insertados también dos registros, uno para cada asiento. El de los cuchillos (F36) es igual, correspondiendo a la misma clave ajena número 2 de antes, pero el otro corresponde ahora a las cuentas por pagar a los proveedores, y por eso el valor que está en la clave ajena número 8, 78, es el número de la factura remitida por el proveedor. De aquí en adelante el programa navegará por el MODRO con un asiento por la ruta 2 y con el otro por la ruta 8. Las tablas que participan ahora son el "Diario", "Mercancías", "FacturasP" (facturas de proveedor), "Proveedores" y "Mayor". (Ver figura No 38).

Pero además, esta instancia de la transacción 0, crea la necesidad de ocurrir otra instancia de otra transacción, sea la 8, en algún momento posterior. Cuando llegue el día de pagar, por ejemplo el 15 de enero de 2006, se ejecutará entonces la transacción 8 complementaria, instanciada con los siguientes asientos:

Código Cuenta	**Nombre Cuenta**	**Débito**	**Crédito**
400	Proveedores	2400	
572	Bancos		2400

El “Diario” es una tabla del MODRO sobre la que los programas sólo pueden ejecutar inserciones, no pueden modificar registros o eliminar registros porque esto sería como alterar la verdadera historia de la Organización. Por lo tanto, se puede observar en el “Diario” de la figura No 40 la acumulación de transacciones. Entonces coexisten cuatro registros en esta entidad, dos de la transacción 0 y dos de la transacción 8. Los dos primeros son semejantes a los de la transacción 1. Los dos siguientes son los asientos 3 y 4, uno débito y otro crédito, cada uno por $2400 que es el valor a pagar.

Diario

Tra	Fecha	Hora	Nro_doc	Benefi	Syseña	Asiento	D/C	Valor	Cajena1
0	20051215	10:34	10000	Empresa	Fulano	1	Db	2400	Nulls
0	20051215	10:34	10000	Empresa	Fulano	2	Cr	2400	Nulls

Cajena2	Cajena3	Cajena4	Cajena5	Cajena6	Cajena7	Cajena8	Cajena9	
1F36	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	78	Nulls	←

Mercancías

Código	Nombre	Exist.	Valor_existencia	cuenta_c
1F36	Cuchillos	200+100	2400+1000	300

FacturasP

nro_fac	valor_fac	fecha_fac	nit_prov	
78	2400	20051215	91300	←

Proveedores

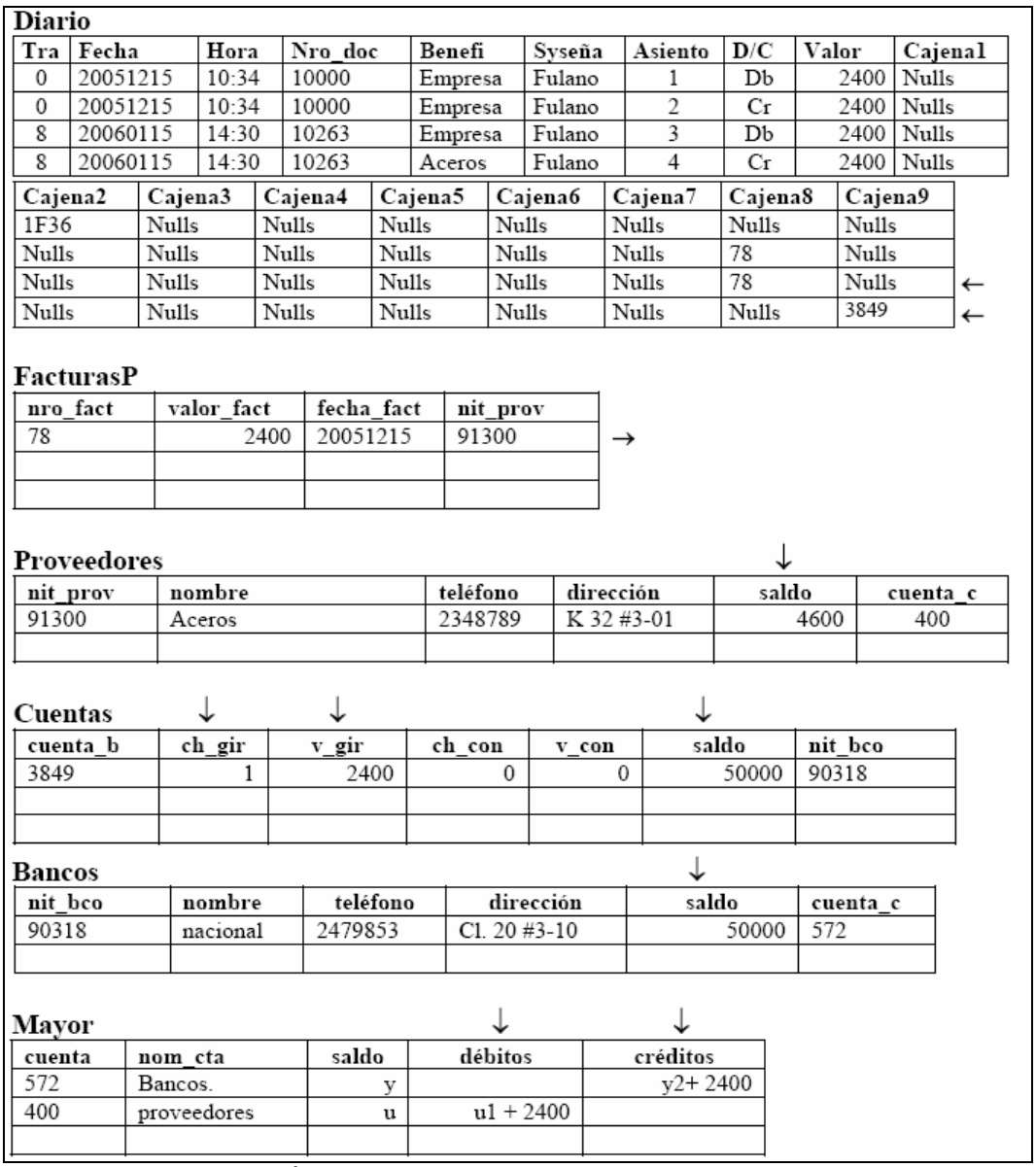
nit_prov	nombre	teléfono	dirección	saldo	cuenta_c
91300	Aceros	2348789	K 32 #3-01	7000	400

Mayor

cuenta	nombre_cuenta	saldo	débitos	créditos
300	Existencias comerciales	x	x1 + 2400	
400	proveedores	u		u2 + 2400

Figura No 38. Actualización de tablas de la base de datos en la compra a plazos.

La transacción 8, como se deduce del “Diario” en la figura No 39, se moverá por las rutas 8 y 9 ya que existe un valor no nulo (78 que es número de la factura) para la clave ajena 8 (Cajena8) en el asiento 3 y un valor no nulo (3849 que es número de la cuenta del cheque girado), para la clave ajena 9 (Cajena9) en el asiento 4. Además, el programa correspondiente a la transacción 8 accederá a las tablas “Facturas”, “Proveedores”, “Cuentas”, “Bancos” y “Mayor” del MODRO, modificándolas como se ve en la figura No 41. Los valores x1, x2 (para “Existencias comerciales”), y1, y2 (para “Bancos”) y u1 y u2 (para “proveedores”) en el archivo “Mayor”, se refieren a los créditos o a los débitos acumulados del mes, con los cuales se calculará al terminar el período contable (el mes), los nuevos saldos x, y, u de cada cuenta.



Diario

Tra	Fecha	Hora	Nro_doc	Benefi	Syseña	Asiento	D/C	Valor	Cajena1
0	20051215	10:34	10000	Empresa	Fulano	1	Db	2400	Nulls
0	20051215	10:34	10000	Empresa	Fulano	2	Cr	2400	Nulls
8	20060115	14:30	10263	Empresa	Fulano	3	Db	2400	Nulls
8	20060115	14:30	10263	Aceros	Fulano	4	Cr	2400	Nulls

Cajena2	Cajena3	Cajena4	Cajena5	Cajena6	Cajena7	Cajena8	Cajena9	
1F36	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	78	Nulls	
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	78	Nulls	←
Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	Nulls	3849	←

FacturasP

nro_fact	valor_fact	fecha_fact	nit_prov	
78	2400	20051215	91300	→

Proveedores

nit_prov	nombre	teléfono	dirección	saldo	cuenta_c
91300	Aceros	2348789	K 32 #3-01	4600	400

Cuentas

cuenta_b	ch_gir	v_gir	ch_con	v_con	saldo	nit_bco
3849	1	2400	0	0	50000	90318

Bancos

nit_bco	nombre	teléfono	dirección	saldo	cuenta_c
90318	nacional	2479853	Cl. 20 #3-10	50000	572

Mayor

cuenta	nom_cta	saldo	débitos	créditos
572	Bancos.	y		y2+ 2400
400	proveedores	u	u1 + 2400	

Figura No 39. Actualización de tablas de la base de datos en el pago de la compra a plazos.

A continuación se muestra el ODL para la clase “compra mercancía a plazos”, “clase mercancía” y la clase “pago factura proveedores”.

```
/*
Implementacion del MODRO. Ejemplo 2 extraido del capitulo 5 de la
tesis doctoral del profesor Jaime Albarracin.
*/

class universal
{

  entity mayor
  {
   cuenta                varchar(5) not null primary key,
   nombre_cuenta         varchar(30) not null,
   saldo                 decimal(10,2)not null,
   debitos               decimal(8,2)not null,
   creditos              decimal(8,2)not null
  }

  entity diario
  {
   tra                   varchar(3) not null,
   fecha                 date not null,
   hora                  time not null,
   nro_docum             varchar(8) not null,
   beneficiario          varchar(30),
   sysena                varchar(10),
   asiento               varchar(2) not null,
   d_c                   varchar(1),
   valor                 decimal(8,2) not null
  }

  method universal()
  {
  }

};
```

```
// Definicion Clase Compra_Mercancia
class compra_mercancia extend universal
{
  /*Inicia la definicion de Entidades pertenecientes a
  la Clase Compra_Mercancia */
  entity mercancia
  {
   codigo                varchar(5) not null primary key,
   nombre                varchar(30) not null,
   existencia            integer not null,
   valor_existencia      decimal(10,2) not null,
   cuenta_c              varchar(5) not null,
   foreign key(cuenta_c) references mayor(cuenta)
  }
  entity diario
  {
   cajena2               varchar(5),
   foreign key(cajena2)  references mercancia(codigo)
  }
  /*Finaliza la definicion de Entidades pertenecientes a
  la Clase Compra_Mercancia */

  method comprar
  // Parametros Entidad Diario
  (cod_trans__,fecha__,hora__,nro_docum__,beneficiario__,
   sysenia__,asiento__,deb_cred__,valor__,cod_mercancia__,
  // Parametros Entidad Mercancia
  cantidad__)
  {
   insert into diario values
   (&cod_trans__,'fecha__','hora__',&nro_docum__,'
    beneficiario__','sysenia__',&asiento__,
   'deb_cred__',&valor__,'cod_mercancia__',null);

   update mercancia set
   existencia = existencia + cantidad__,
   valor_existencia = valor_existencia + valor__
   where codigo = 'cod_mercancia__';

   update mayor set
   debitos = debitos + valor__ where cuenta = (
   select cuenta_c from mercancia
   where codigo = 'cod_mercancia__' ) ;
  }
};
```

```
// Definicion Clase Compra_Mercancia_Plazos
class compra_mercancia_plazos extend compra_mercancia
{

 entity proveedores
 {
  nit_prov               numeric not null primary key,
  nombre                 varchar(30) not null,
  telefono               numeric,
  direccion              varchar(30),
  saldo                  numeric not null,
  cuenta_c               numeric(4) not null,
  foreign key(cuenta_c)  references mayor(cuenta)
 }

 entity facturasp
 {
  nro_fact               varchar(5) not null primary key,
  valor_fact             numeric not null,
  fecha_fact             date not null,
  nit_prov               numeric not null,
  foreign key(nit_prov)  references proveedores(nit_prov)
 }

 entity diario
 {
  cajena8                varchar(5),
  foreign key(cajena8)   references facturasp(nro_fact)
 }

 method comprar
  // Parametros Entidad facturasp
  (beneficiario2__,asiento2__,deb_cred2__,nro_fact__,nit_prov__ )
 {
   insert into diario values
   (&cod_trans__,'fecha__','hora__',&nro_docum__,
   'beneficiario2__','sysenia__',&asiento2__,'deb_cred2__',
   &valor__,null,null,&nro_fact__);

   insert into facturasp values
   (&nro_fact__,&valor__,'fecha__',&nit_prov__);

   update proveedores set
   saldo = saldo + valor__
   where nit_prov = nit_prov__ ;
```

```
   update mayor set
   creditos = creditos + valor__
   where cuenta = ( select cuenta_c from proveedores
   where nit_prov = nit_prov__ );
 }
};


// Definicion Clase Pagos
class pagos extend universal
{

 entity bancos
 {
  nit_bco               varchar(14) not null primary key,
  nom_banco             varchar(30) not null,
  telefono              varchar(15),
  direccion             varchar(30),
  saldo                 decimal(10,2),
  cuenta_c              varchar(5) not null,
  foreign key(cuenta_c) references mayor(cuenta)
 }

 entity cuentas
 {
  cuenta_ban            varchar(15) not null primary key,
  cheque_gir            smallint not null,
  valor_giro            decimal(10,2) not null,
  cheque_con            smallint,
  valor_con             decimal(10,2),
  saldo                 decimal(10,2) not null,
  nit_bco               varchar(14) not null,
  foreign key(nit_bco)  references bancos(nit_bco)
 }

 entity diario
 {
  cajena9               varchar(15),
  foreign key(cajena9)  references cuentas(cuenta_ban)
 }
```

```
 method pagar(
  beneficiario__,sysenia__,asiento__,deb_cred__,valor__,
  cod_mercancia__,nit_prov__,cuenta_b__,ch_gir__ )
  {
   insert into diario values
   (&cod_trans__,'fecha__','hora__',&nro_docum__,
   'beneficiario__','sysenia__',&asiento__,'deb_cred__',
   &valor__,null,null,&nro_fact__);

   update cuentas set
   ch_gir = ch_gir + ch_gir__,
   valor_giro = valor_giro + valor__,
   saldo = saldo - valor__
   where cuenta_ban = cuenta_b__;

   update bancos set
   saldo = saldo + valor__
   where nit_bco = ( select nit_bco from cuentas
   where cuenta_ban = cuenta_b__) ;

   update mayor set
   creditos = creditos + valor__
   where cuenta = ( select cuenta_c from bancos where nit_bco = (
     select nit_bco from cuentas
  where cuenta_ban = cuenta_b__));
 }
};


//Definicion clase pagos_facturas
class pagos_facturas extend pagos
{

 entity proveedores
 {
 }

 entity facturasp
 {
 }
```

```
 method pagar(
  cod_trans__,fecha__,hora__,nro_docum__,
  beneficiario2__,asiento2__,deb_cred2__,nro_fact__ )
 {
   insert into diario values
   (&cod_trans__,'fecha__','hora__',&nro_docum__,
   'beneficiario2__','sysenia__',&asiento2__,'deb_cred2__',
   &valor__,null,&cuenta_b__,null);

   delete from facturasp
   where nro_fact = nro_fact__ ;

   update proveedores set
   saldo = saldo - valor__
   where nit_prov = nit_prov__ ;

   update mayor set
   debitos = debitos + valor__
   where cuenta = ( select cuenta_c from proveedores
   where nit_prov = nit_prov__ );

 }
};
```

El comando a utilizar para implementar el método de la clase compra a plazos:

```
method compra_mercancia_plazos (0, 2008-01-14,21:09:00,
10000,empresa, fulano, 1, d, 2400, f36, 200,empresa,
2,c,78,91300);
```

El comando a utilizar para implementar el método de la pago de facturas a proveedores:

```
method pagos_facturas (8, 2008-02-24,21:09:00, 10263,empresa,
fulano, 1, d, 2400, f36, aceros, 2,c,78,91300,3849,1);
```

CONCLUSIONES

El desarrollo del intérprete LATIN permitió implementar computacionalmente un MODRO (sencillo) según lo planteado en la tesis doctoral del profesor Albarracín. Con este logro se finaliza la primera fase del desarrollo del sistema de gestión de bases de datos reorientado a objetos (SGBDRO) obteniendo excelentes resultados que permitirán a los siguientes proyectos continuar con el desarrollo de este sistema.

La integración de los componentes ODL y OQL en la conformación del lenguaje LATIN que permite la descripción del MODRO, convierte al LATIN en un lenguaje de integración de datos y a futuro en un lenguaje de programación con gestión de datos. Lo anterior con base al nuevo modelo de datos planteado por el profesor Albarracín que busca resolver la desintegración de la organización en general y ofrecer simultáneamente una alternativa para las bases de datos orientada a objetos, las cuales continúan generando muchas expectativas, insatisfechas todavía.

Finalmente se destaca el seguimiento por parte del profesor Albarracín durante el análisis, diseño e implementación de cada uno de los componentes del intérprete LATIN, que permitió corroborar los conceptos expuestos en su tesis doctoral con la estructuca y funcionalidad del intérprete.

RECOMENDACIONES

- El desarrollo de los tipos (*tipo compras, tipo ventas, etc.*) para el lenguaje LATIN, que sirvan como guía en la implementación del MODRO, es fundamental para facilitar el uso del SGBDRO.

- El desarrollo de un entorno gráfico para la consulta a las clases del MODRO. Éste entorno debe permitir visualizar las clases así como también las entidades pertenecientes a ellas, los métodos y todas las características necesarias que le permitan al usuario del SGBDRO consultar y manipular de manera visual las clases del MODRO.

- El análisis, diseño e implementación de un motor de almacenamiento propio (para este trabajo de grado se utilizó el InnoDB con MySQL) que permita aumentar el nível estructural y funcional del modelo entidad – relación, a un modelo más acorde al planteado en la tesis del profesor Albarracín, que entre sus caracteristicas contemple la bidireccionalidad entre las entidades padre – hijo.

- El análisis, diseño e implementación de todas las estructuras y funciones propias de un sistema de gestion de bases de datos. Estas estructuras y funciones deben contemplar los componentes desarrollados para el SGBDRO y lograr la integración de los mismos.

BIBLIOGRAFIA

AHO, Alfred, SETHI, Ravi ULLMAN, Jeffrey, LAM Monica. Compilers: Principles, Techniques, and Tools. Segunda Edición. Addison Wesley 2006.

LOUDEN, Keneth C, Construcción de Compiladores: Principios y práctica. International Thomson Editores. 2004

ALBARRACÍN, Jaime. Tesis Doctoral "Integración de datos y procesos en las organizaciones mediante un Modelo de Datos Reorientado a Objetos".Universidad de Oviedo (España). 2006.

LEVINE, John R, MASON Tony, BROWN Doug. Lex & Yacc. Segunda Edición 1992. O'Reilly & Associates.

Bases de Datos Orientadas a Objetos Consultadas

http://source.db4o.com/browse/db4o/trunk/objectmanager/src
http://www.db4o.com/
http://www.odbms.org/
http://www.odmg.org/

ANEXOS

ANEXO A: ARCHIVO JFLEX PARA GENERAR EL ANALIZADOR LEXICO DE LOS COMPONENTES ODL Y OQL

```
/*
  Este documento contiene las reglas del analizador lexico implementado
  para los componentes ODL y OQL del SGBDRO.

  Implementado por:
                        - Javier Cala Uribe
                        - Francisco Contreras
                        - Arturo Garcia Payares
                        Grupo de Investigacion FOCO Organizacional
                        Universidad Industrial de Santander

  Nombre del Archivo: Lex_ODL_OQL.flex
*/

import java_cup.runtime.*;
import javax.swing.*;
import javax.swing.text.Caret;
import java.awt.event.*;
import java.awt.*;
import java.io.*;


%%

/* -----------------Opciones y Declaraciones----------------- */

/*
   El nombre de la Clase que JFlex creara sera Lexer
   El codigo se escribira en el archivo Lexer.java
*/

%class Lexer

/*
 El numero actual de lineas puede ser accedido por la varaibale yyline
 y el numero actual de columnas con la variable yycolumn
*/

%line
%column

/*Compatibilidad con el generador de analizadores sintactico CUP*/

%cup
```

```
/*Declaraciones*/
%{
    /* Para crear un nuevo java_cup.runtime.Symbol con informacion sobre
       el actual token, los token no tendrian valor en este caso. */

       private Symbol symbol(int type) {
            return new Symbol(type, yyline, yycolumn);
    }

    /* Tambien se crea un nuevo java_cup.runtime.Symbol con informacion
       sobre el actual token, pero este objecto tiene un valor. */

       private Symbol symbol(int type, Object value) {
            return new Symbol(type, yyline, yycolumn, value);
    }
%}


/*
  Declaraciones Macro

  Estas declaraciones son expresiones regulares que seran usadas
  despues en las Reglas Lexicas
*/



/* Comentarios */
Comentario = {ComentarioTradicional} | {FinLineaComentario} |
             {DocumentacionComentario}
InputCharacter = [^\r\n]
ComentarioTradicional   = "/*" [^*] ~"*/" | "/*" "*"+ "/"
FinLineaComentario     = "//" {InputCharacter}* {FinLinea}
DocumentacionComentario = "/**" {ContenidoComentario} "*"+ "/"
ContenidoComentario       = ( [^*] | \*+ [^/*] )*

/* Final de una linea \r , \n , o  \r\n. */
FinLinea = \r|\n|\r\n

/* Espacio en Blanco. */
EspacioBlanco     = {FinLinea} | [ \t\f]

/* Numeros  */
dec_numero = 0
            | [1-9][0-9]*
            | [0-9]+"."[0-9]*
            | "."[0-9]*

/* Campo Fecha */
//DATE
/*Una fecha. El rango soportado es de '1000-01-01' a '9999-12-31'. MySQL
muestra valores DATE en formato 'YYYY-MM-DD', pero permite asignar valores a
columnas DATE usando cadenas de carácteres o números.  */
campo_fecha = [1-9][0-9][0-9][0-9]"-"[0-1][0-9]"-"[0-3][0-9]
```

```
/* Campo Hora */
//TIME
/*Una hora. El rango es de '-838:59:59' a '838:59:59'. MySQL muestra los
valores TIME en formato 'HH:MM:SS', pero permite asingar valores a columnas
TIME usando números o cadenas de carácteres. */
campo_hora = [0-2][0-9]":"[0-5][0-9]":"[0-5][0-9]


/* Variable */
dec_variable = [A-Za-z_][A-Za-z_0-9]*

/* Comparacion */
comparar =  "="
            | "<>"
            | "<"
            | ">"
            | "<="
            | ">="

%%
/* ------------------------Reglas Lexicas---------------------- */

<YYINITIAL> {

    ";"                 { return symbol(sym.PUNTOYCOMA); }
    "+"                 { return symbol(sym.MAS); }
    "-"                 { return symbol(sym.MENOS); }
    "*"                 { return symbol(sym.POR); }
    "/"                 { return symbol(sym.DIVISION); }
    "("                 { return symbol(sym.IPAREN); }
    ")"                 { return symbol(sym.DPAREN); }
    "{"                 { return symbol(sym.ILLAVE); }
    "}"                 { return symbol(sym.DLLAVE); }
    ","                 { return symbol(sym.COMA); }
    "."                 { return symbol(sym.PUNTO); }
    ":"                 { return symbol(sym.DOSPUNTO); }
    "'"                 { return symbol(sym.COMILLA); }
    "&"                 { return symbol(sym.AMPERSAND); }

    /* Palabras Reservervadas */
    "all"               { return symbol(sym.ALL); }
    "avg"               { return symbol(sym.AMMSC, new String(yytext())); }
    "min"               { return symbol(sym.AMMSC, new String(yytext())); }
    "max"               { return symbol(sym.AMMSC, new String(yytext())); }
    "sum"               { return symbol(sym.AMMSC, new String(yytext())); }
    "count"             { return symbol(sym.AMMSC, new String(yytext())); }
    "and"               { return symbol(sym.AND); }
    "any"               { return symbol(sym.ANY); }
    "auto_increment"    { return symbol(sym.AUTOINCREMENT); }
    "between"           { return symbol(sym.BETWEEN); }
    "by"                { return symbol(sym.BY); }
    "varchar"           { return symbol(sym.VARCHAR); }
    "cascade"           { return symbol(sym.CASCADE); }
    "check"             { return symbol(sym.CHECK); }
```

```
"class"            { return symbol(sym.CLASS); }
"date"             { return symbol(sym.DATE); }
"decimal"          { return symbol(sym.DECIMAL); }
"default"          { return symbol(sym.DEFAULT); }
"delete"           { return symbol(sym.DELETE); }
"distinct"         { return symbol(sym.DISTINCT); }
"double"           { return symbol(sym.DOUBLE); }
"drop"             { return symbol(sym.DROP); }
"entity"           { return symbol(sym.ENTITY); }
"escape"           { return symbol(sym.ESCAPE); }
"exists"           { return symbol(sym.EXISTS); }
"extend"           { return symbol(sym.EXTEND); }
"float"            { return symbol(sym.FLOAT); }
"foreign"          { return symbol(sym.FOREIGN); }
"from"             { return symbol(sym.FROM); }
"group"            { return symbol(sym.GROUP); }
"having"           { return symbol(sym.HAVING); }
"in"               { return symbol(sym.IN); }
"indicator"        { return symbol(sym.INDICATOR); }
"insert"           { return symbol(sym.INSERT); }
"integer"          { return symbol(sym.INTEGER); }
"into"             { return symbol(sym.INTO); }
"is"               { return symbol(sym.IS); }
"key"              { return symbol(sym.KEY); }
"like"             { return symbol(sym.LIKE); }
"method"           { return symbol(sym.METHOD); }
"not"              { return symbol(sym.NOT); }
"null"             { return symbol(sym.NULLX); }
"numeric"          { return symbol(sym.NUMERIC); }
"on"               { return symbol(sym.ON); }
"or"               { return symbol(sym.OR); }
"precision"        { return symbol(sym.PRECISION); }
"primary"          { return symbol(sym.PRIMARY); }
"real"             { return symbol(sym.REAL); }
"references"       { return symbol(sym.REFERENCES); }
"select"           { return symbol(sym.SELECT); }
"set"              { return symbol(sym.SET); }
"smallint"         { return symbol(sym.SMALLINT); }
"some"             { return symbol(sym.SOME); }
"show"             { return symbol(sym.SHOW); }
"table"            { return symbol(sym.TABLE); }
"time"             { return symbol(sym.TIME); }
"unique"           { return symbol(sym.UNIQUE); }
"update"           { return symbol(sym.UPDATE); }
"values"           { return symbol(sym.VALUES); }
"where"            { return symbol(sym.WHERE); }

/* Si encuentra un numero, retorna el token INTNUM
   y el valor en el string yytext */
{dec_numero}       { return symbol(sym.INTNUM, new String(yytext())); }

/* Si encuentra un identificador retorna el token NAME */
{dec_variable}     { return symbol(sym.NAME, new String(yytext()));}
```

```
      /* Si encuentra un identificador retorna el token FECHA      */
    {campo_fecha}    { return symbol(sym.FECHA, new String(yytext()));}

      /* Si encuentra un identificador retorna el token HORA      */
    {campo_hora}     { return symbol(sym.HORA, new String(yytext()));}

      /*Si encuentra un signo de comparacion retorna el token COMPARACION */
    {comparar}       { return symbol(sym.COMPARACION, new String(yytext())); }

    /* No hace nada si encuetra espacio en blanco */
    {EspacioBlanco}  { /* una vez encontrado, no hace nada */ }

    /*Comentarios */
    {Comentario}     { /* una vez encontrado, no hace nada */ }

}


/* Si encuentra un caracter extraño regresa un error. */
[^]              {
                      Interfaz.correcto = false;
                      Interfaz.areaTexto2.setText(
                      Interfaz.areaTexto2.getText()+
                      "Error : Error Lexico \n");

                      Interfaz.areaTexto2.setText(
                      Interfaz.areaTexto2.getText()+
                      "Error : Caracter no reconocido < "+yytext()+" >");
                       throw new Error("Caracter no reconocido < "+
                      yytext()+" >");
                 }
```

ANEXO B: ARCHIVO CUP PARA GENERAR EL ANALIZADOR SINTACTICO DE LOS COMPONENTES ODL Y OQL

```
//--------------------------------------------------------------------------
//  Este archivo contiene las reglas del analizador sintactico implementado
//  para los componentes ODL y OQL del LATIN.
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
//  Implementado por:
//                      - Javier Cala Uribe
//                      - Francisco Contreras
//                      - Arturo Garcia Payares
//                      Grupo de Investigacion FOCO Organizacional
//                      Universidad Industrial de Santander
//
//  Nombre del Archivo:      Sint_ODL_OQL.cup
// --------------------------------------------------------------------------


/* ----------------------Declaraciones Preliminares--------------------*/

/* Importar la java_cup.runtime.* y las clases necesarias
   para manipulación de archivos  */

import java_cup.runtime.*;
import java.lang.*;
import java.io.*;
import javax.swing.*;
import javax.swing.text.*;
import java.awt.Color;

/* En el siguiente segmento de codigo encerrado en "parser code {:  :}"
se cambian los metodos de reportes de error y se agregan nuevos metodos para
ser usados en la generacion de archivos*/

parser code {:

  /*Variable en la cual se almacena el nombre de la Clase,
  necesaria para la creacion de archivos de clases*/

  public String NombreClase = new String();

  /*Variable en la cual se almacena el nombre de la clase padre
  cuando una de ellas posee herencia*/

  public String ClasePadre = new String();

  /*Variable booleana que me indica si una clase posee herencia*/

  public boolean TienePadre = false;
```

```
/*Variable en la cual se almacena el nombre de la entidad*/

public String NombreEntidad = new String();

/* Cambia el metodo report_error que muestra la linea y la columna
donde ocurrio el error en la entrada como tambien la razon por medio
del String 'message'. */

public void report_error(String message, Object info) {
/* Crea un StringBuffer llamado 'm' con la cadena 'Error'. */
  StringBuffer m = new StringBuffer("Error");

  /* Verifica si la informacion pasada es del tipo
  java_cup.runtime.Symbol. */

    if (info instanceof java_cup.runtime.Symbol) {
    /* Declara un objeto java_cup.runtime.Symbol 's'
    con la informacion en la info del objeto. */

      java_cup.runtime.Symbol s = ((java_cup.runtime.Symbol) info);

      /* Verifica si el numero de lineas en la entrada es >= cero */
      if (s.left >= 0) {

      /* Agrega al final del StringBuffer un mensaje de error
      con el numero de la linea en la entrada. */
        m.append(" en linea "+(s.left+1));
        /* Verifica si el numero de columna en la entrada es >= cero. */
        if (s.right >= 0)
        /* Agrega al final del StringBuffer un mensaje de error
        con el numero de la columna en la entrada. */
          m.append(", columna "+(s.right+1));
      }
    }

    /* Agrega al final del StringBuffer un mensaje de error
    creado en este metodo. */
    m.append(" : "+message+"\n");

    /* Imprimie el contenido del StringBuffer 'm',
    que contiene un mensaje de error.  En AreaTexto2*/

    Interfaz.correcto = false;

    try {
      StyleConstants.setForeground(Interfaz.colorTexto, Color.RED);
      Interfaz.areaTexto2.getStyledDocument().insertString(
        Interfaz.areaTexto2.getStyledDocument().getLength(),
        m.toString()+"\n", Interfaz.colorTexto);
    }
    catch (Exception ex_) {}

}
```

```
/* Cambia el metodo report_fatal que reporta un error fatal mostrando el
numero de la linea y la columna donde el error ocurrio en la entrada como la
razon por la que el error fatal ocurrio dentro del metodo.*/

public void report_fatal_error(String message, Object info) {
  report_error(message, info);
}

public void syntax_error(Symbol cur_token) {
/* Este metodo es llamado por el parser tan pronto como un error
de sintaxis es detectado*/
  report_error("Error de Sintaxis", cur_token);
}

public void unrecovered_syntax_error(Symbol cur_token) {
/* Este metodo es llamado por el parser si no es posible
recuperarse de un error de sintaxis */
  report_fatal_error("No se puede continuar el analisis", null);
}

/*Metodo usado para la creación y escritura del archivo SQL
generado por LATIN que contiene las consultas SQL equivalentes para
cada una de las entidades*/
public void escribir_sql(String cadena) {
  char comilla = 34;
  try {
    DataOutputStream archivo_sql = null;
    archivo_sql = new DataOutputStream(
      new FileOutputStream(
      "data/"+Interfaz.nombre_basedatos+"/entidades.sql",true));
    archivo_sql.writeBytes(cadena);
    archivo_sql.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
    report_error("No existe el archivo: "+
    comilla+"entidades.sql"+comilla, null);
  }
  catch (IOException ioe) {/* Error al escribir */
    report_error("No es posible escribir en el archivo:
    "+comilla+"entidades.sql"+comilla, null);
  }
}

/*Metodo usado para la creacion de los archivos de clase, que contienen
los nombres de las  entidades y los metodos de cada una de las clases
declaradas por el usuario
nom_archivo: Nombre del archivo de clase donde se va a escribir
cadena: Es la cadena de texto que va a escribirse en el archivo
tipo: Indica si se va a escribir el nombre de una entidad (e)
o se va a escribir un metodo (m) */
public void escribir_clase(String nom_archivo, String cadena, char tipo) {
  String linea = new String();
  boolean escribirTitulo = true; char comilla = 34;
```

```
try {
  BufferedReader archivo_clase = null;
  archivo_clase = new BufferedReader(
  new InputStreamReader(new FileInputStream(
  "data/"+Interfaz.nombre_basedatos+"/"+nom_archivo+".ltc")));
  if (tipo == 'e') {
    linea = archivo_clase.readLine();
    while (linea != null) {
      if (linea.equals("[entity]")){
        escribirTitulo = false;
        break;
      }
      linea = archivo_clase.readLine();
    }
  }
  if (tipo == 'm') {
    linea = archivo_clase.readLine();
    while (linea != null) {
      if (linea.equals("[method]")){
        escribirTitulo = false;
        break;
      }
      linea = archivo_clase.readLine();
    }
  }
  archivo_clase.close();
}
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
  report_error("No existe el archivo: "+
  comilla+nom_archivo+".ltc"+comilla, null);
}
catch(IOException ioe) {/* Error al leer*/
report_error("No es posible leer el archivo: "+
comilla+nom_archivo+".ltc"+comilla, null);
}

try {
  DataOutputStream archivo_clase = null;
  archivo_clase = new DataOutputStream(new FileOutputStream(
  "data/"+Interfaz.nombre_basedatos+"/"+nom_archivo+".ltc", true));

  if (escribirTitulo && tipo == 'e')
    archivo_clase.writeBytes("[entity]\n");

  if (escribirTitulo && tipo == 'm')
    archivo_clase.writeBytes("\n[method]\n");

  archivo_clase.writeBytes(cadena);
  archivo_clase.close();
}
catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
  report_error("No existe el archivo: "+
  comilla+nom_archivo+".ltc"+comilla, null);
}
```

```
  catch(IOException ioe) {/* Error al escribir */
    report_error("No es posible escribir en el archivo: "+
    comilla+nom_archivo+".ltc"+comilla, null);
  }
}

/*Metodo usado para copiar las instrucciones SQL del metodo de la clase
padre al metodo de la clase Hijo en su respectivo archivo de clase*/
public void escribir_herencia(String Padre, String Hijo) {
  String linea = new String(); char comilla = 34;

  try {
    BufferedReader archivo_padre = null;
    archivo_padre = new BufferedReader(
    new InputStreamReader(new FileInputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+Padre+".ltc")));
    try {
      DataOutputStream archivo_hijo = null;
      archivo_hijo = new DataOutputStream(new FileOutputStream(
      "data/"+Interfaz.nombre_basedatos+"/"+Hijo+".ltc", true));

      linea = archivo_padre.readLine();
      while(linea != null) {
        if (linea.equals("{")) {
           linea = archivo_padre.readLine();
          while(linea.equals("}")==false) {
            archivo_hijo.writeBytes(linea+"\n");
            linea = archivo_padre.readLine();
          }
        }
        linea = archivo_padre.readLine();
      }
      archivo_hijo.close();
    }
    catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
      report_error("No existe el archivo: "+
      comilla+Hijo+".ltc"+comilla, null);
    }
    catch(IOException ioe) {/* Error al escribir */
      report_error("No es posible escribir en el archivo: "+
      comilla+Hijo+".ltc"+comilla, null);
    }

  archivo_padre.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
  report_error("No existe el archivo: "+
  comilla+Padre+".ltc"+comilla, null);
  }
  catch(IOException ioe) {/* Error al leer */
  report_error("No es posible leer el archivo: "+
  comilla+Padre+".ltc"+comilla, null);
  }
}
```

```
/*Retorna un string que contiene los parametros del metodo de la clase
padre, para poder ser escritos en el metodo de la clase hijo*/
public String parametros_padre(String ClasePadre) {
  String linea = new String(); char comilla = 34;
  try {
    BufferedReader archivo_padre = null;
    archivo_padre = new BufferedReader(
    new InputStreamReader(new FileInputStream("
    data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));

    linea = archivo_padre.readLine();
    while(linea != null){
      if(linea.equals("[method]")) {
        linea = archivo_padre.readLine();
        archivo_padre.close();
        int us_comas = 0;
        for ( int i=0; i<linea.length()-1; i++) {
          if ( linea.substring(i,i+1).compareTo(",") == 0)
            us_comas++;
        }
        if(us_comas != 0 )
          return linea.substring(linea.indexOf('(')+1,
                                 linea.indexOf(')'))+", ";
        else
          return linea.substring(linea.indexOf('(')+1,
                                 linea.indexOf(')'));
      }
      linea=archivo_padre.readLine();
    }
    archivo_padre.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
    report_error("No existe el archivo: "+
    comilla+ClasePadre+".ltc"+comilla, null);
  }
  catch(IOException ioe) {/* Error al leer */
    report_error("No es posible leer el archivo: "+
    comilla+ClasePadre+".ltc"+comilla, null);
  }
  return "";
}

/*Este metodo copia el nombre de todas las entidades de la clase padre
al archivo de la clase hijo*/
public String entidades_padre(String ClasePadre, String ClaseHijo) {
  String linea = new String(); char comilla = 34;
  try {
    BufferedReader archivo_padre = null;
    archivo_padre = new BufferedReader(
    new InputStreamReader(new FileInputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));
    DataOutputStream archivo_hijo = null;
    archivo_hijo = new DataOutputStream( new FileOutputStream("
```

```
    data/"+Interfaz.nombre_basedatos+"/"+ClaseHijo+".ltc", true));

    linea = archivo_padre.readLine();
    while(linea.equals("[method]") == false) {
      if(linea.equals("[entity]") == false)
        archivo_hijo.writeBytes(linea+"\n");
      linea = archivo_padre.readLine();
    }

    archivo_padre.close();
    archivo_hijo.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
    report_error("No es posible abrir uno de los siguientes archivos: "+
    comilla+ClasePadre+".ltc"+comilla+" ó "+
    comilla+ClaseHijo+".ltc"+comilla, null);
  }
  catch(IOException ioe) {/*Error al leer*/
    report_error("No es posible leer y/o escribir en uno de los"+
    "siguientes archivos: "+comilla+ClasePadre+".ltc"+comilla+
    " ó "+comilla+ClaseHijo+".ltc"+comilla, null);
  }

  return "";
}

/*Comprueba que la entidad: "NombreEntidad"  esté creada en la clase:
"ClasePadre", en cuyo caso retorna true en caso contrario retorna false */
public boolean existe_entidad(String ClasePadre, String NombreEntidad) {
  String linea = new String(); char comilla = 34;
  try{
    BufferedReader archivo_padre = new BufferedReader(
    new InputStreamReader(new FileInputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+ClasePadre+".ltc")));
    linea = archivo_padre.readLine();
    while(linea != null){
      if(linea.equals(NombreEntidad)){
        archivo_padre.close();
        return true;
      }
      linea = archivo_padre.readLine();
    }
    archivo_padre.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
    report_error("No existe el archivo: "+
    comilla+ClasePadre+".ltc"+comilla, null);
  }
  catch(IOException ioe) {/*Error al leer*/
    report_error("No es posible leer el archivo: "+
    comilla+ClasePadre+".ltc"+comilla, null);
  }
  return false;
}
```

```
/*Modifica los metodos de  una clase padre cuando en una entidad hija
se agregan nuevos campos a sus entidades*/
public void actualizar_metodo(String NombreClase, String NombreEntidad) {
  String linea = new String(); char comilla = 34;
  //Paso todo el contenido del archivo de clase, a un archivo temporal
  try{
    BufferedReader archivo_clase = new BufferedReader(
    new InputStreamReader(new FileInputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".ltc")));
    DataOutputStream archivo_temporal = new DataOutputStream(
    new FileOutputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp"));
    linea = archivo_clase.readLine();
    while(linea != null){
      archivo_temporal.writeBytes(linea+"\n");
      linea = archivo_clase.readLine();
    }
    archivo_temporal.close();
    archivo_clase.close();
  }
  catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
    report_error("No es posible abrir uno de los siguientes archivos: "+
    comilla+NombreClase+".ltc"+comilla+" ó "+
    comilla+NombreClase+".tmp"+comilla, null);
  }
  catch(IOException ioe) {/*Error al leer*/
    report_error("No es posible leer y/o escribir en uno de los "+
    "siguientes archivos: "+comilla+NombreClase+".ltc"+comilla+" ó "+
    comilla+NombreClase+".tmp"+comilla, null);
  }

  /*Ahora reescribo todo el archivo de clase, haciendo las modificaciones
  al metodo si encuentra una sentencia INSERT INTO NombreEntidad*/
  try{
    BufferedReader arch_temporal = new BufferedReader(
    new InputStreamReader(new FileInputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp")));
    DataOutputStream arch_clase = new DataOutputStream(
    new FileOutputStream(
    "data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".ltc"));
    linea = arch_temporal.readLine();
    while(linea != null){
      if(linea.startsWith("  INSERT INTO "+NombreEntidad)){
        linea = linea.substring(0, linea.lastIndexOf(')'))+", NULL);";
        arch_clase.writeBytes(linea+"\n");
      }
      else
        arch_clase.writeBytes(linea+"\n");
      linea = arch_temporal.readLine();
    }
    arch_clase.close();
    arch_temporal.close();
  }
```

```
    catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
      report_error("No es posible abrir uno de los siguientes archivos: "+
      comilla+NombreClase+".ltc"+comilla+" ó "+
      comilla+NombreClase+".tmp"+comilla, null);
    }
    catch(IOException ioe) {/*Error al leer*/
      report_error("No es posible leer y/o escribir en uno de los "+
      "siguientes archivos: "+comilla+NombreClase+".ltc"+comilla+" ó "+
      comilla+NombreClase+".tmp"+comilla, null);
    }

    //Por ultimo borro el archivo temporal
    File temporal = new    File(
    "data/"+Interfaz.nombre_basedatos+"/"+NombreClase+".tmp");
    temporal.delete();
  }
:};

/* El bloque de codigo encerrado en "init with {: :}" se ejecuta justo antes
de que el parser empiece a hacer el analisis*/

init with{:
  /*Verifico que la carpeta "data/base_datos" donde se guardan los archivos
  de clase esté creada, si no es así, entonces se crea antes de empezar el
  analizis */

  File carpeta = new File("data");
  if(carpeta.isDirectory() == false)
    carpeta.mkdir();

  carpeta = new File("data/"+Interfaz.nombre_basedatos);
  if(carpeta.isDirectory() == false)
    carpeta.mkdir();

  /*La siguiente porción de código es usada para crear un nuevo archivo
  entidades.sql justo antes de empezar el analisis y escribir un encabezado.
  Si ya existe un archivo llamado entidades.sql se borra todo su contenido*/
  char comilla = 34;
  try {
    DataOutputStream archivo_sql = null;
    archivo_sql = new DataOutputStream( new  FileOutputStream(
    "data/"+Interfaz.nombre_basedatos+"/entidades.sql"));
    archivo_sql.writeBytes("/*"+
    "Archivo SQL generado automaticamente por LATIN v0.1\n"+
    "Este archivo contiene las consultas SQL equivalentes a "+
    "cada una de las entidades\n"+
    "Programado Por:\n"+
    "      -Javier Cala Uribe\n"+
    "      -Francisco Contreras Herazo\n"+
    "      -Arturo Garcia Payares\n"+
    "      Grupo de Investigación: Foco Organizacional\n"+
    "      Universidad Industrial de Santander\n"+
    "Nombre del archivo: entidades.sql  "+
    "*/\n");
```

```
      archivo_sql.close();
    }
    catch(FileNotFoundException fnfe) {/*Archivo no encontrado*/
      report_error("No es posible abrir el archivo: "+
      comilla+"entidades.sql"+comilla, null);
    }
    catch (IOException ioe) { /* Error al escribir */
    report_error("No es posible escribir en el archivo: "+
    comilla+"entidades.sql"+comilla, null);
  }

:};

/* ------------Declaracion de Terminales y No Terminales ----------- */

/*
Terminales (tokens regresados por el scanner).
Los terminales que no tienen valor son llamados primeros
y los que si tienen son llamados posteriormente.
*/

terminal    AMPERSAND, MAS, MENOS, POR, DIVISION, IPAREN,
            DPAREN, ILLAVE, DLLAVE, COMA;
terminal    PUNTO, DOSPUNTO, ALL, AND, ANY, BETWEEN, BY,
            VARCHAR, CASCADE, CHECK, CLASS;
terminal    COMPARACION, COMILLA, DATE, DECIMAL, DEFAULT,
            DELETE, DISTINCT, DOUBLE;
terminal    ENTITY, ESCAPE, EXISTS, EXTEND, FLOAT, FOREIGN,
            FROM, GROUP, HAVING;
terminal    IN, INDICATOR, INSERT, INTEGER, INTO, IS, KEY,
            LIKE, METHOD, NOT, NULLX;
terminal    NUMERIC, ON, OR, PRECISION, PRIMARY, REAL,
            REFERENCES, SELECT, SET, SMALLINT;
terminal    SOME, TIME, UNIQUE, UPDATE, VALUES, WHERE,
            PUNTOYCOMA, SHOW, AUTOINCREMENT;
terminal    Object AMMSC, INTNUM, NAME, HORA, FECHA, DROP, TABLE;

/*
No terminales usados en la gramatica.
Los Terminales que no tienen un valor entero están en la lista.
Un tipo objeto significa que puede ser de cualquier tipo,es decir,
no pertenece a un conjunto específico.
Por lo tanto, podría ser un Integer o un String o lo que sea.
*/

non terminal Object     select_statement, consulta_entidades,
                        update_statement_searched;
non terminal Object     assignment, assignment_commalist, insert_atom;
non terminal Object     insert_atom_commalist, query_spec,
                        values_or_query_spec, opt_column_commalist;
non terminal Object     insert_statement, opt_where_clause,
                        delete_statement_searched;
non terminal Object     definicion_funciones, cuerpo_metodo, insert_paramet,
                        paramet_lista, definicion_metodo;
```

```
non terminal Object     parameter, identificador, columna, tipo_dato,
                        column_ref, table;
non terminal Object     function_ref, parameter_ref, atom, escalar_exp,
                        opt_having_clause, column_ref_commalist;
non terminal Object     opt_group_by_clause, where_clause, table_ref,
                        table_ref_commalist, from_clause, table_exp;
non terminal Object     escalar_exp_commalist, selection, opt_all_distinct,
                        subquery, prueba_existencia, any_all_some;
non terminal Object     todo_o_algun_predicado, atom_commalist, en_predicado,
                        prueba_para_null, opt_escape;
non terminal Object     like_predicado, entre_predicado,
                        comparacion_predicado, predicado, condicion_busqueda;
non terminal Object     listado_columna, def_interrelacion,
                        listado_opc_columna, opc_def_columna, def_columna;
non terminal Object     elemento_tabla, listado_atributo, definicion_entidad,
                        cuerpo_clase, definicion_clase;
non terminal Object     usar_metodo, inicial, herencia, identificador_copy,
                        seguir;
non terminal Object     insertar_parametro, parametro_lista,
                        aux_definicion_funciones, drop_statement;


/* -------------Precedencia y asociacion ----------- */
/*
La línea Inferior siempre tendrá mayor precedencia que la línea anterior,
es decir, que POR y DIVISION debería tener una mayor precedencia que MAS,
MENOS
*/
precedence left MAS, MENOS;
precedence left POR, DIVISION;
precedence left AND, OR;


/* ----------------------------  Gramatica  -------------------- */

/* Producciones para gramatica :: ODL */

inicial ::= definicion_clase PUNTOYCOMA seguir
            {: Interfaz.def_clase = true; :}
            |
            consulta_entidades PUNTOYCOMA
            {: Interfaz.consulta_sql = true; :}
            |
            METHOD usar_metodo PUNTOYCOMA
            {: Interfaz.usar_metodo = true; :}
            |
            SHOW NAME PUNTOYCOMA
            ;

seguir ::=  |
            inicial
            ;
```

```
usar_metodo ::=   NAME IPAREN parametro_lista DPAREN
                  ;

parametro_lista   ::=   insertar_parametro
                        |
                        parametro_lista COMA insertar_parametro
                        ;

insertar_parametro      ::=   NAME
                              |
                              INTNUM
                              |
                              NULLX
                              |
                              HORA
                              |
                              FECHA
                              ;

/* Definicion de una Clase */

definicion_clase  ::=   CLASS NAME:nombre_clase
                        {:
                          parser.NombreClase = nombre_clase.toString();
                          /*A continuación se borran los archivos de clase
                           con el mismo nombre */
                          File archivo_clase;
                          archivo_clase = new File(
                           "data/"+Interfaz.nombre_basedatos+"/"+
                           parser.NombreClase+".ltc");
                          if(archivo_clase.exists())
                             if(archivo_clase.delete()){
                               char comilla = 34;
                               Interfaz.areaTexto2.setText(
                                Interfaz.areaTexto2.getText()+
                                "Advertencia: Ya existia una clase llamada
                                "+comilla+parser.NombreClase+comilla+
                                " y ha sido reescrita\n");
                             }
                          archivo_clase.createNewFile();
                        :}
                        herencia:padre
                        {: parser.ClasePadre = padre.toString(); :}

                        cuerpo_clase
                        ;

herencia    ::=   {: RESULT = new String(); parser.TienePadre = false; :}
                  |
                  EXTEND identificador:padre
                  {: RESULT = new String(padre.toString());
                  parser.TienePadre = true; :}
                  ;
```

```
cuerpo_clase      ::=   ILLAVE
                        definicion_entidad
                        {:
                         if (parser.TienePadre)
                           parser.entidades_padre(parser.ClasePadre,
                           parser.NombreClase);
                        :}
                        definicion_metodo
                        DLLAVE
                        ;

definicion_entidad      ::=   |
                              ENTITY NAME:nombre_entidad
                              {: parser.NombreEntidad =
                                  nombre_entidad.toString();
                                 parser.escribir_sql("\n\n/*Entidad "+
                                  parser.NombreEntidad+
                                  " perteneciente a la clase "+
                                  parser.NombreClase+" */\n");

                                  if (parser.TienePadre &&
                                  parser.existe_entidad(
                                  parser.ClasePadre, parser.NombreEntidad))
                                      parser.escribir_sql("ALTER TABLE "+
                                       parser.NombreEntidad + " \n");
                                 else {
                                     parser.escribir_sql("CREATE TABLE "+
                                     parser.NombreEntidad + "(\n" );

                                     parser.escribir_clase(parser.NombreClase,
                                     parser.NombreEntidad +"\n",'e');
                                 }
                              :}
                              ILLAVE listado_atributo:list_atrib DLLAVE
                              {: if (parser.TienePadre &&
                                  parser.existe_entidad(
                                  parser.ClasePadre, parser.NombreEntidad))

                                       parser.escribir_sql(
                                       list_atrib.toString()+"\n;" );
                                 else
                                       parser.escribir_sql(
                                       list_atrib.toString()+"\n);" );
                              :}
                              definicion_entidad
                              ;
```

```
listado_atributo  ::=   elemento_tabla:elem_tabla
                        {:if (parser.TienePadre && parser.existe_entidad(
                          parser.ClasePadre, parser.NombreEntidad))
                              RESULT = new String(
                               " ADD "+elem_tabla.toString());
                          else
                              RESULT = new String(" "+elem_tabla.toString());
                        :}
                        |
                        listado_atributo:list_atrib COMA
                        elemento_tabla:elem_tabla
                        {: if (parser.TienePadre && parser.existe_entidad(
                          parser.ClasePadre, parser.NombreEntidad))
                              RESULT = new String(list_atrib.toString()+
                               ",\n ADD "+elem_tabla.toString());
                          else
                              RESULT = new String(list_atrib.toString()+
                               ",\n "+elem_tabla.toString());

                        :}
                        ;

elemento_tabla    ::=   def_columna:def_col
                        {: RESULT = new String(def_col.toString()); :}
                        |
                        def_interrelacion:def_inter
                        {:
                              RESULT = new String(def_inter.toString());
                              if(parser.TienePadre && parser.existe_entidad(
                               parser.ClasePadre, parser.NombreEntidad))
                                    parser.actualizar_metodo(
                                   parser.ClasePadre,parser.NombreEntidad);

                        :}
                        ;

def_columna       ::=   columna:col tipo_dato:tip_dat
                        opc_def_columna:opc_def_col
                        {: RESULT = new String(col.toString()+
                        " "+tip_dat.toString()+opc_def_col.toString()); :}
                        ;

opc_def_columna   ::=   {: RESULT = new String(); :}
                        |
                        opc_def_columna:opc_def_col
                        listado_opc_columna:list_opc_col
                        {: RESULT = new String(opc_def_col.toString()+
                        " "+list_opc_col.toString()); :}
                        ;

listado_opc_columna ::= NOT NULLX
                        {: RESULT = new String("NOT NULL"); :}
                        |
                        NOT NULLX UNIQUE
```

```
                        {: RESULT = new String("NOT NULL UNIQUE"); :}
                        |
                        NOT NULLX PRIMARY KEY
                        {: RESULT = new String("NOT NULL PRIMARY KEY"); :}
                        |
                        NOT NULLX AUTOINCREMENT PRIMARY KEY
                        {: RESULT = new String(
                        "NOT NULL AUTO_INCREMENT PRIMARY KEY"); :}
                        |
                        DEFAULT INTNUM:numero
                        {: RESULT = new String("DEFAULT "+
                        numero.toString()); :}
                        |
                        DEFAULT NULLX
                        {: RESULT = new String("DEFAULT NULL"); :}
                        |
                        DEFAULT NAME:nombre
                        {: RESULT = new String("DEFAULT "+
                        nombre.toString()); :}
                        |
                        DEFAULT COMILLA NAME:nombre COMILLA
                        {: RESULT = new String("DEFAULT '"+
                        nombre.toString()+"'"); :}
                        |
                        CHECK IPAREN condicion_busqueda:cond_busq DPAREN
                        {: RESULT = new String("CHECK("+
                        cond_busq.toString()+")"); :}
                        ;

def_interrelacion ::=   FOREIGN KEY IPAREN listado_columna:list_col
                        DPAREN REFERENCES table:nombre
                        {: RESULT = new String("FOREIGN KEY("+
                        list_col.toString()+") REFERENCES "+
                        nombre.toString()); :}
                        |
                        FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN
                        REFERENCES table:nombre IPAREN
                        listado_columna:list_col2 DPAREN
                        {: RESULT = new String("FOREIGN KEY("+
                        list_col1.toString()+") REFERENCES "+
                        nombre.toString()+"("+list_col2.toString()+")"); :}
                        |
                        FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN
                        REFERENCES table:nombre IPAREN
                        listado_columna:list_col2 DPAREN ON DELETE CASCADE
                        {: RESULT = new String("FOREIGN KEY("+
                        list_col1.toString()+") REFERENCES "+
                        nombre.toString()+"("+list_col2.toString()+
                        ") ON DELETE CASCADE"); :}
                        |
                        FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN
                        REFERENCES table:nombre IPAREN
                        listado_columna:list_col2 DPAREN ON UPDATE CASCADE
                        {: RESULT = new String("FOREIGN KEY("+
```

```
                        list_col1.toString()+") REFERENCES "+
                        nombre.toString()+"("+list_col2.toString()+
                        ") ON UPDATE CASCADE"); :}
                        |
                        FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN
                        REFERENCES table:nombre IPAREN
                        listado_columna:list_col2 DPAREN
                        ON DELETE CASCADE ON UPDATE CASCADE
                        {: RESULT = new String("FOREIGN KEY("+
                        list_col1.toString()+") REFERENCES "+
                        nombre.toString()+"("+list_col2.toString()+") ON
                        DELETE CASCADE ON UPDATE CASCADE"); :}
                        |
                        FOREIGN KEY IPAREN listado_columna:list_col1 DPAREN
                        REFERENCES table:nombre IPAREN
                        listado_columna:list_col2 DPAREN
                        ON UPDATE CASCADE ON DELETE CASCADE
                        {: RESULT = new String("FOREIGN KEY("+
                        list_col1.toString()+") REFERENCES "+
                        nombre.toString()+"("+list_col2.toString()+
                        ") ON UPDATE CASCADE ON DELETE CASCADE"); :}
                        ;

listado_columna   ::=   columna:col
                        {: RESULT = new String(col.toString()); :}
                        |
                        listado_columna:list_col COMA columna:col
                        {: RESULT = new String(list_col.toString()+
                        ", "+col.toString()); :}
                        ;

condicion_busqueda ::=  condicion_busqueda:cond_busq1 OR
                        condicion_busqueda:cond_busq2
                        {: RESULT = new String(cond_busq1.toString()+
                        " OR "+cond_busq2.toString()); :}
                        |
                        condicion_busqueda:cond_busq1 AND
                        condicion_busqueda:cond_busq2
                        {: RESULT = new String(cond_busq1.toString()+
                        " AND "+cond_busq2.toString()); :}
                        |
                        NOT condicion_busqueda:cond_busq
                        {: RESULT = new String(" NOT "+
                        cond_busq.toString()); :}
                        |
                        IPAREN condicion_busqueda:cond_busq DPAREN
                        {: RESULT = new String(" ("+
                        cond_busq.toString()+")"); :}
                        |
                        predicado:pred
                        {: RESULT = new String(pred.toString()); :}
                        ;
```

```
predicado   ::=         comparacion_predicado:comp_pred
                        {: RESULT = new String(comp_pred.toString()); :}
                        |
                        entre_predicado:ent_pred
                        {: RESULT = new String(ent_pred.toString()); :}
                        |
                        like_predicado:like_pred
                        {: RESULT = new String(like_pred.toString()); :}
                        |
                        prueba_para_null:prueb_null
                        {: RESULT = new String(prueb_null.toString()); :}
                        |
                        en_predicado:en_pred
                        {: RESULT = new String(en_pred.toString()); :}
                        |
                        todo_o_algun_predicado:toa_pred
                        {: RESULT = new String(toa_pred.toString()); :}
                        |
                        prueba_existencia:exist
                        {: RESULT = new String(exist.toString()); :}
                        ;

comparacion_predicado   ::=   escalar_exp:esc_exp1 COMPARACION:comp
                              escalar_exp:esc_exp2
                              {: RESULT = new String(esc_exp1.toString()+
                              " "+comp.toString()+" "+esc_exp2.toString()); :}
                              |
                              escalar_exp:esc_exp COMPARACION:comp
                              subquery:subcons
                              {: RESULT = new String(esc_exp.toString()+
                              " "+comp.toString()+" "+subcons.toString()); :}
                              ;

entre_predicado   ::=   escalar_exp:esc_exp1 NOT BETWEEN
                        escalar_exp:esc_exp2 AND escalar_exp:esc_exp3
                        {: RESULT = new String(esc_exp1.toString()+
                        " NOT BETWEEN "+esc_exp2.toString()+
                        " AND "+esc_exp3.toString()); :}
                        |
                        escalar_exp:esc_exp1 BETWEEN escalar_exp:esc_exp2
                        AND escalar_exp:esc_exp3
                        {: RESULT = new String(esc_exp1.toString()+
                        " BETWEEN "+esc_exp2.toString()+" AND "+
                        esc_exp3.toString()); :}
                        ;

like_predicado    ::=   escalar_exp:esc_exp NOT LIKE atom:atm
                        opt_escape:opt_esc
                        {: RESULT = new String(esc_exp.toString()+
                        " NOT LIKE "+atm.toString()+opt_esc.toString()); :}
                        |
                        escalar_exp:esc_exp LIKE atom:atm opt_escape:opt_esc
                        {: RESULT = new String(esc_exp.toString()+
                        " LIKE "+atm.toString()+opt_esc.toString()); :} ;
```

```
opt_escape  ::=         {: RESULT = new String(); :}
                        |
                        ESCAPE atom:atm
                        {: RESULT = new String(" ESCAPE "+atm.toString()); :}
                        ;


prueba_para_null  ::=   column_ref:col_ref IS NOT NULLX
                        {: RESULT = new String(col_ref.toString()+
                         "IS NOT NULL "); :}
                        |
                        column_ref:col_ref IS NULLX
                        {: RESULT = new String(col_ref.toString()+
                         "IS NULL "); :}
                        ;


en_predicado      ::=   escalar_exp:esc_exp NOT IN
                        IPAREN subquery:sub_cons DPAREN
                        {: RESULT = new String(esc_exp.toString()+
                        " NOT IN( "+sub_cons.toString()+")"); :}
                        |
                        escalar_exp:esc_exp IN IPAREN subquery:sub_cons DPAREN
                        {: RESULT = new String(esc_exp.toString()+
                        " IN( "+sub_cons.toString()+")"); :}
                        |
                        escalar_exp:esc_exp NOT IN IPAREN
                        atom_commalist:atm_com DPAREN
                        {: RESULT = new String(esc_exp.toString()+
                        " NOT IN( "+atm_com.toString()+")"); :}
                        |
                        escalar_exp:esc_exp IN IPAREN
                        atom_commalist:atm_com DPAREN
                        {: RESULT = new String(esc_exp.toString()+
                        " IN( "+atm_com.toString()+")"); :}
                        ;


atom_commalist    ::=   atom:atm
                        {: RESULT = new String(atm.toString()); :}
                        |
                        atom_commalist:atm_com COMA atom:atm
                        {: RESULT = new String(atm_com.toString()+
                        ", "+atm.toString()); :}
                        ;


todo_o_algun_predicado  ::=   escalar_exp:esc_exp COMPARACION:comp
                              any_all_some:any_some subquery:sub_cons
                              {: RESULT = new String(esc_exp.toString()+
                              " "+comp.toString()+" "+any_some.toString()+
                              " "+sub_cons.toString()); :}
                              ;
```

```
any_all_some      ::=   ANY
                        {: RESULT = new String("ANY"); :}
                        |
                        ALL
                        {: RESULT = new String("ALL"); :}
                        |
                        SOME
                        {: RESULT = new String("SOME"); :}
                        ;

prueba_existencia ::=   EXISTS subquery:sub_cons
                        {: RESULT = new String(" EXISTS "+
                        sub_cons.toString()); :}
                        ;

subquery    ::=   IPAREN SELECT opt_all_distinct:opt_all
                  selection:sel table_exp:tab_exp DPAREN
                  {: RESULT = new String("(SELECT "+opt_all.toString()+
                  " "+sel.toString()+" "+tab_exp.toString()+")"); :}
                  ;

opt_all_distinct  ::=   {: RESULT = new String(); :}
                        |
                        ALL
                        {: RESULT = new String("ALL"); :}
                        |
                        DISTINCT {: RESULT = new String("DISTINCT"); :}
                        ;

selection   ::=   escalar_exp_commalist:esc_exp_list
                  {: RESULT = new String(esc_exp_list.toString()); :}
                  |
                  POR
                  {: RESULT = new String("*"); :}
                  ;

escalar_exp_commalist ::=     escalar_exp:esc_exp
                              {: RESULT = new String(esc_exp.toString()); :}
                              |
                              escalar_exp_commalist:esc_exp_list
                               COMA escalar_exp:esc_exp
                              {: RESULT = new String(esc_exp_list.toString()+
                              ", "+esc_exp.toString()); :}
                              ;

table_exp   ::=   from_clause:from opt_where_clause:where
                  opt_group_by_clause:group opt_having_clause:having
                  {: RESULT = new String(from.toString()+
                  " "+where.toString()+" "+group.toString()+
                  " "+having.toString()); :}
                  ;
```

```
from_clause       ::=   FROM table_ref_commalist:tab_ref
                        {: RESULT = new String(" FROM "+
                        tab_ref.toString()); :}
                        ;


table_ref_commalist ::= table_ref:tab_ref
                        {: RESULT = new String(tab_ref.toString()); :}
                        |
                        table_ref_commalist:tab_ref_list
                        COMA table_ref:tab_ref
                        {: RESULT = new String(tab_ref_list.toString()+
                        ", "+tab_ref.toString()); :}
                        ;


table_ref   ::=   table:nombre
                  {: RESULT = new String(nombre.toString()); :}
                  ;


where_clause      ::=   WHERE condicion_busqueda:cond_busq
                        {: RESULT = new String(" WHERE "+
                        cond_busq.toString()); :}
                        ;


opt_group_by_clause ::= {: RESULT = new String(); :}
                        |
                        GROUP BY column_ref_commalist:col_ref_list
                        {: RESULT = new String(" GROUP BY "+
                        col_ref_list.toString()); :}
                        ;


column_ref_commalist ::=column_ref:col_ref
                        {: RESULT = new String(col_ref.toString()); :}
                        |
                        column_ref_commalist:col_ref_list
                        COMA column_ref:col_ref
                        {: RESULT = new String(col_ref_list.toString()+
                        ", "+col_ref.toString()); :}
                        ;


opt_having_clause ::=   {: RESULT = new String(); :}
                        |
                        HAVING condicion_busqueda:cond_busq
                        {: RESULT = new String(" HAVING "+
                        cond_busq.toString()); :}
                        ;
```

```
escalar_exp ::=   escalar_exp:esc_exp1 MAS escalar_exp:esc_exp2
                  {: RESULT = new String(esc_exp1.toString()+
                  " + "+esc_exp2.toString()); :}
                  |
                  escalar_exp:esc_exp1 MENOS escalar_exp:esc_exp2
                  {: RESULT = new String(esc_exp1.toString()+
                  " - "+esc_exp2.toString()); :}
                  |
                  escalar_exp:esc_exp1 POR escalar_exp:esc_exp2
                  {: RESULT = new String(esc_exp1.toString()+
                  " * "+esc_exp2.toString()); :}
                  |
                  escalar_exp:esc_exp1 DIVISION escalar_exp:esc_exp2
                  {: RESULT = new String(esc_exp1.toString()+
                  " / "+esc_exp2.toString()); :}
                  |
                  MAS escalar_exp:esc_exp
                  {: RESULT = new String(" + "+esc_exp.toString()); :}
                  |
                  MENOS escalar_exp:esc_exp
                  {: RESULT = new String(" - "+esc_exp.toString()); :}
                  |
                  atom:atm
                  {: RESULT = new String(atm.toString()); :}
                  |
                  column_ref:col_ref
                  {: RESULT = new String(col_ref.toString()); :}
                  |
                  function_ref:fun_ref
                  {: RESULT = new String(fun_ref.toString()); :}
                  |
                  IPAREN escalar_exp:esc_exp DPAREN
                  {: RESULT = new String("("+esc_exp.toString()+")"); :}
                  ;

atom  ::=   parameter_ref:param_ref
            {: RESULT = new String(param_ref.toString()); :}
            |
            INTNUM:numero
            {: RESULT = new String(numero.toString()); :}
            |
            AMPERSAND NAME:nombre
            {: RESULT = new String("&"+nombre.toString()); :}
            |
            COMILLA NAME:nombre COMILLA
            {: RESULT = new String("'"+nombre.toString()+"'"); :}
            |
            COMILLA HORA:ora COMILLA
            {: RESULT = new String("'"+ora.toString()+"'"); :}
            |
            COMILLA FECHA:fech COMILLA
            {: RESULT = new String("'"+fech.toString()+"'"); :}
            ;
```

```
parameter_ref     ::=   parameter:param
                        {: RESULT = new String(param.toString()); :}
                        |
                        parameter:param1 parameter:param2
                        {: RESULT = new String(param1.toString()+
                        " "+param2.toString()); :}
                        |
                        parameter:param1 INDICATOR parameter:param2
                        {: RESULT = new String(param1.toString()+
                        " INDICATOR "+param2.toString()); :}
                        ;

function_ref      ::=   AMMSC:func IPAREN POR DPAREN
                        {: RESULT = new String(func.toString()+"(*)"); :}
                        |
                        AMMSC:func IPAREN DISTINCT column_ref:col_ref DPAREN
                        {: RESULT = new String(func.toString()+
                        "(DISTINCT "+col_ref.toString()+" )"); :}
                        |
                        AMMSC:func IPAREN ALL escalar_exp:esc_exp DPAREN
                        {: RESULT = new String(func.toString()+
                        "(ALL "+esc_exp.toString()+")"); :}
                        |
                        AMMSC:func IPAREN escalar_exp:esc_exp DPAREN
                        {: RESULT = new String(func.toString()+
                        "("+esc_exp.toString()+")"); :}
                        ;

table ::=   NAME:nombre
            {:RESULT = new String(nombre.toString()); :}
            |
            NAME:nombre1 PUNTO NAME:nombre2
            {:RESULT = new String(nombre1.toString()+
            "."+nombre2.toString()); :}
            ;

column_ref  ::=   table:nombre
                  {: RESULT = new String(nombre.toString()); :}
                  |
                  NAME:nombre1 PUNTO NAME:nombre2 PUNTO NAME:nombre3
                  {: RESULT = new String(nombre1.toString()+"."+
                  nombre2.toString()+"."+nombre3.toString()); :}
                  ;


tipo_dato   ::=   VARCHAR IPAREN INTNUM:numero DPAREN
                  {: RESULT = new String("VARCHAR("+numero.toString()+")"); :}
                  |
                  NUMERIC
                  {: RESULT = new String("NUMERIC"); :}
                  |
                  NUMERIC IPAREN INTNUM:numero DPAREN
                  {: RESULT = new String("NUMERIC("+numero.toString()+")"); :}
                  |
```

```
                    NUMERIC IPAREN INTNUM:numero1 COMA INTNUM:numero2 DPAREN
                    {: RESULT = new String("NUMERIC("+numero1.toString()+
                    ","+numero2.toString()+")"); :}
                    |
                    DECIMAL
                    {: RESULT = new String("DECIMAL"); :}
                    |
                    DECIMAL IPAREN INTNUM:numero DPAREN
                    {: RESULT = new String("DECIMAL("+numero.toString()+")"); :}
                    |
                    DECIMAL IPAREN INTNUM:numero1 COMA INTNUM:numero2 DPAREN
                    {: RESULT = new String("DECIMAL("+numero1.toString()+
                    ","+numero2.toString()+")"); :}
                    |
                    INTEGER
                    {: RESULT = new String("INTEGER"); :}
                    |
                    SMALLINT
                    {: RESULT = new String("SMALLINT"); :}
                    |
                    FLOAT
                    {: RESULT = new String("FLOAT"); :}
                    |
                    FLOAT IPAREN INTNUM:numero DPAREN
                    {: RESULT = new String("FLOAT("+numero.toString()+")"); :}
                    |
                    REAL
                    {: RESULT = new String("REAL"); :}
                    |
                    DOUBLE PRECISION
                    {: RESULT = new String("DOUBLE PRECISION"); :}
                    |
                    DATE
                    {: RESULT = new String("DATE"); :}
                    |
                    TIME
                    {: RESULT = new String("TIME"); :}
                    ;

columna     ::=   NAME:nombre
                  {: RESULT = new String(nombre.toString()); :}
                  ;

identificador     ::=   NAME:nombre identificador_copy:id
                        {: RESULT = new String(nombre.toString()+
                        ""+id.toString()); :}
                        ;

identificador_copy      ::=   {: RESULT = new String(); :}
                              |
                              PUNTO NAME:nombre identificador_copy:id
                              {: RESULT = new String("."+
                              nombre.toString()+""+id.toString()); :}
                              ;
```

```
parameter   ::=   DOSPUNTO NAME:nombre
                  {: RESULT = new String(":"+nombre.toString() ); :}
                  ;

/* Fin Definicion de Atributos : Entidades - Inicio definicion de metodos  */

definicion_metodo ::=   METHOD NAME:nombre_metodo IPAREN
                        {: parser.escribir_clase(parser.NombreClase,
                           nombre_metodo.toString()+"(",'m');
                           if (parser.TienePadre)
                              parser.escribir_clase(parser.NombreClase,
                               parser.parametros_padre(parser.ClasePadre),'m');
                        :}
                        paramet_lista:param_list DPAREN
                        {: parser.escribir_clase(parser.NombreClase,
                           param_list.toString()+")",'m'); :}
                        cuerpo_metodo
                        ;

paramet_lista     ::=   insert_paramet:ins_param
                        {: RESULT = new String(ins_param.toString()); :}
                        |
                        paramet_lista:param_list COMA insert_paramet:ins_param
                        {: RESULT = new String(param_list.toString()+
                        ", "+ins_param.toString());
                        :}
                        ;

insert_paramet    ::=   {: RESULT = new String(); :}
                        |
                        NAME:nombre
                        {: RESULT = new String(nombre.toString()); :}
                        |
                        NULLX
                        {: RESULT = new String("NULL"); :}
                        ;

cuerpo_metodo     ::=   ILLAVE
                        {: parser.escribir_clase(parser.NombreClase,
                        " \n{\n",'m'); :}
                        {: if (parser.TienePadre)
                           parser.escribir_herencia(parser.ClasePadre,
                           parser.NombreClase);
                        :}
                        definicion_funciones:def_func
                        {: parser.escribir_clase(parser.NombreClase,
                           def_func.toString(),'m');
                        :}
                        DLLAVE
                        {: parser.escribir_clase(
                           parser.NombreClase, "}\n",'m');
                        :}
                        ;
```

```
aux_definicion_funciones ::= definicion_funciones:def_func
                             {:RESULT = new String(def_func.toString()); :}
                             ;

definicion_funciones ::=     {: RESULT = new String(); :}
                             |
                             delete_statement_searched:delete PUNTOYCOMA
                             aux_definicion_funciones:aux_func
                             {: RESULT = new String("  "+delete.toString()+
                                ";\n"+aux_func.toString());
                             :}
                             |
                             insert_statement:insert PUNTOYCOMA
                             aux_definicion_funciones:aux_func
                             {: RESULT = new String("  "+
                                insert.toString()+";\n"+aux_func.toString());
                             :}
                             |
                             update_statement_searched:update PUNTOYCOMA
                             aux_definicion_funciones:aux_func
                             {: RESULT = new String("  "+update.toString()+
                                ";\n"+aux_func.toString());
                             :}
                             ;

delete_statement_searched ::=DELETE FROM table:nombre opt_where_clause:where
                             {: RESULT = new String("DELETE FROM "+
                                nombre.toString()+" "+where.toString());
                             :}
                             ;

opt_where_clause  ::=   {: RESULT = new String(); :}
                        |
                        where_clause:where
                        {: RESULT = new String(where.toString()); :}
                        ;

insert_statement  ::=   INSERT INTO table:nombre opt_column_commalist:opc_col
                        values_or_query_spec:values
                        {: RESULT = new String("INSERT INTO "+
                           nombre.toString()+" "+opc_col.toString()+
                           " "+values.toString());
                        :}
                        ;

opt_column_commalist ::={: RESULT = new String(); :}
                        |
                        IPAREN listado_columna:list_col DPAREN
                        {: RESULT = new String("("+
                           list_col.toString()+")");
                        :}
                        ;
```

```
values_or_query_spec ::=VALUES IPAREN insert_atom_commalist:ins_atm DPAREN
                        {: RESULT = new String("VALUES ("+
                           ins_atm.toString()+")");
                        :}
                        |
                        query_spec:cons_spec
                        {: RESULT = new String(cons_spec.toString()); :}
                        ;


query_spec  ::=   SELECT opt_all_distinct:opc_all
                  selection:sel table_exp:tab_exp
                  {: RESULT = new String("SELECT "+opc_all.toString()+
                     " "+sel.toString()+" "+tab_exp.toString());
                  :}
                  ;


insert_atom_commalist ::=     insert_atom:ins_atm
                              {: RESULT = new String(ins_atm.toString()); :}
                              |
                              insert_atom_commalist:ins_atm_list
                              COMA insert_atom:ins_atm
                              {: RESULT = new String(ins_atm_list.toString()+
                                 ", "+ins_atm.toString());
                              :}
                              ;


insert_atom ::=   atom:atm
                  {: RESULT = new String(atm.toString()); :}
                  |
                  NULLX
                  {: RESULT = new String("NULL"); :}
                  ;


assignment_commalist ::=assignment:asignacion
                        {: RESULT = new String(asignacion.toString()); :}
                        |
                        assignment_commalist:asign_com
                        COMA assignment:asignacion
                        {: RESULT = new String(asign_com.toString()+
                           ", "+asignacion.toString());
                        :}
                        ;
```

```
assignment  ::=   columna:col COMPARACION:comp escalar_exp:esc_exp
                  {: RESULT = new String(col.toString()+" "+
                  comp.toString()+" "+esc_exp.toString()); :}
                  |
                  columna:col COMPARACION:comp NULLX
                  {: RESULT = new String(col.toString()+
                  " "+comp.toString()+" NULL"); :}
                  |
                  columna:col COMPARACION:comp
                  subquery:subcons escalar_exp:esc_exp
                  {: RESULT = new String(col.toString()+
                  " "+comp.toString()+" "+subcons.toString()+
                  " "+esc_exp.toString()); :}
                  ;

update_statement_searched ::= UPDATE table:nombre SET
                              assignment_commalist:asignacion
                              opt_where_clause:opc_where
                              {: RESULT = new String("UPDATE "+
                              nombre.toString()+" SET "+asignacion.toString()+
                              " "+opc_where.toString()); :}
                              ;

/* Finaliza Definición Metodos  -- OQL Definición de Consulta de Entidades  */

consulta_entidades ::=  select_statement
                        {: Conexion.select = true;  :}
                        |
                        delete_statement_searched
                        |
                        insert_statement
                        |
                        update_statement_searched
                        |
                        drop_statement
                                    ;

select_statement  ::=   SELECT opt_all_distinct selection table_exp
                        ;

drop_statement    ::=   DROP TABLE paramet_lista
                        ;
```

ANEXO C: CORREOS ENVIADOS POR LOS DESARROLLADORES DEL GENERADOR DE ANALIZADORES SINTÁCTICOS CUP

Scott Hudson <scott.hudson@cs.cmu.edu> ocultar detalles 8/10/07
responder a scott.hudson@cs.cmu.edu
para Javier Cala Uribe <javier.cala@gmail.com>
fecha 08-oct-2007 17:41
asunto RE: Question CUP
enviado por cs.cmu.edu

I actually had a look at canonical-LR when I built CUP. I thought that perhaps the kind of size issues that were the reason that canonical-LR was considered impractical might be less of an issue on modern machines (which have several orders of magnitude more memory than the ones available when LR parsing was invented). However, I quickly realized that the state space growth was in fact exponential not only in the worst case, but for lots of common/practical cases -- canonical-LR significantly exceeded available memory even for some small and seemingly tame examples. Since LALR(1) is only a slightly smaller class of grammars -- in practice you just don't seem to come across many if any grammars that are LR(1) but not LALR(1) -- I think the common practice of using LALR instead of canonical-LR is definitely the right one.

Scott
--
Prof. Scott Hudson http://www.cs.cmu.edu/~hudson
HCI Institute, School of Computer Science, Carnegie Mellon University

---

Andrea Flexeder <flexeder@in.tum.de> ocultar detalles 10/10/07
para Javier Cala Uribe <javier.cala@gmail.com>
fecha 10-oct-2007 0:42
asunto Re: Question CUP

Hello Javier,

tanks for your email.
Since we thought about developing a LR(1)-Parser ourselves, I am
delighted to hear that it is your plan to do so.
I think there is no drawback in using LR instead of LALR.
It is only that the states of the finite automaton will become quite
large. However, you can perform

an optimization to reduce the number of states.

The only thing you have to consider when developing an LR(1)-Parser in
Java, is the limitation of methods,
because they are not allowed to be bigger than 65k.
Idem is a problem in CUP.
That should be the only difficulty you have to deal with.
The LR-Parser is powerful enough

Is it your plan to do this project open source?
Since we would be interested in comparing it to CUP.
If you have any further questions, they are welcome.

Best regards,
Andrea

--

._____. ._____.
Andrea Flexeder|_. ._. | | ._. ._. |
Technische Universitaet München | | | | | | | | | |
Institut für Informatik | | | |_| | | | | |
Lehrstuhl für Sprachen und Beschreibungsstrukturen |_| |_____| |_| |_|
Lehrstuhl I/2 Boltzmannstrasse 3 85748 Garching http://www2.in.tum.de
Telefon: 089 / 28918178 Fax: 089 / 28918161 Email: flexeder@in.tum.de

---

Help with CUP (LR-Parser)!!! From Colombia

Hi Andrea, My name is Arturo and I'm developing with Javier Cala a
LR-Parser with CUP Tool. But now, we have a problem, we need to know the
String value of a terminal token.
We defined the terminal like an Object, but we don't know how to obtain
the value of the token.

For example,

terminal CLASS
terminal Object CLASSNAME

Now, we need to get the value of CLASSNAME like a String in CUP.

How can we do that?

When we type:

```
expr ::= CLASS CLASSNAME:NameClass
        {: [..Java Code..] NameClass.intValue() :}
        ;
```

We obtain a 1 (one) like a result.
But if we typed for example "CLASS Class1" we want to obtain "Class1"
like a string for continue with our develope.
We try with NameClass.stringValue() but obviously we obtained an error.

Thanks for your help.

/Arturo II Garcia Payares
Student. Systems Engineering and Computer Science
Universidad Industrial de Santander - Colombia/

Hi,

in your example.
I do not know if the terminal CLASSNAME should be of type object.
If not I would make it type String, otherwise:
if you want to obtain the String of CLASS and CLASSNAME, I would write:

```
terminal Object CLASS
terminal Object CLASSNAME
expr ::= CLASS:c CLASSNAME:n
      {: String s = c.toString() + n.toString();  :}
      ;
```

That should work ;-) (see
http://www2/projekte/cuptum/manual.html#production_list)
I hope this will help you with your problem. If not, let me know.

Best regards,
Andrea