EXDACLET: Un Herramienta Web para Limpieza y Transformación de Datos

V.2.0

MANUAL DE REFERENCIA

Universidad de Nariño
Facultad de Ingeniería
Departamento de Sistemas
San Juan de Pasto
2018

Tabla de contenido

TABLA DE FIGURAS

1. ARQUITECTURA DE LA HERRAMIENTA EXDACLET

En la figura 1 se muestra la arquitectura de la herramienta exdaclet

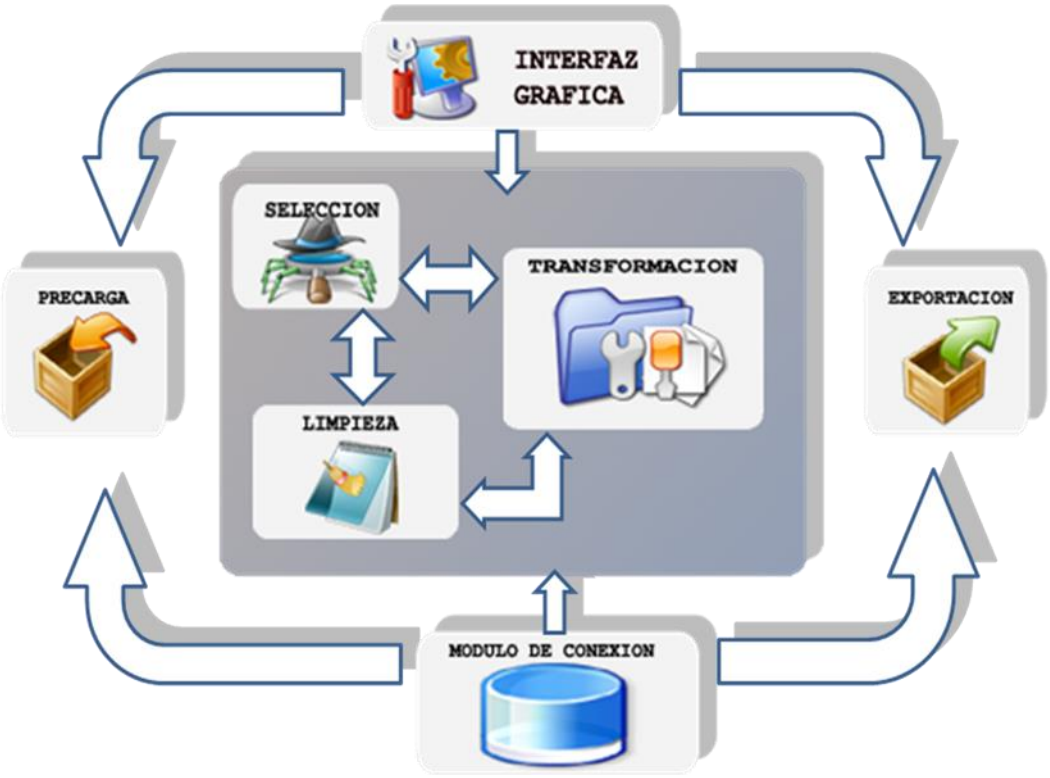


Figura 1. Arquitectura de EXDACLET

1.1. MODULOS

La Herramienta EXDACLET esta compuesta por los siguientes módulos:

1.1.1 Módulo de Conexión.

Este módulo es el encargado de mantener una comunicación constante entre los filtros de precarga, limpieza, selección y transformación que ejecute el usuario en el sistema, con el Sistema Gestor de Bases de Datos que almacena los resultados de dichos filtros en tablas de datos, además tiene como objetivo la identificación de usuarios en base a atributos que ellos contienen (login, password, etc.), con el fin de que solo personas autorizadas tengan acceso a la manipulación de la herramienta.

1.1.2 Módulo de importación.

Este módulo es el encargado de transferir a la máquina servidor todos los archivos planos a los cuales el usuario les aplicará uno a más filtros. Su objetivo se centra en verificar la consistencia física de cada uno de los archivos que se quieran enviar al servidor y cargar en la herramienta.

1.1.3 Módulo del Kernel de EXDACLET

En este módulo se encuentran los paquetes fundamentales para la aplicación de todo el proceso de limpieza de datos. En este se encuentran los módulos de Limpieza, Transformación y Selección.

El módulo de limpieza contiene todos los filtros necesarios para el tratamiento de datos nulos que se encuentren en las tablas de datos utilizadas por el usuario.

El módulo de transformación contiene todos los posibles procedimientos requeridos por el usuario para la transformación de los datos perdidos (missing) o fuera de rango (outliner) a datos reales y coherentes que puedan ayudar a posteriores procesos de análisis de datos, inteligencia de negocios o tareas de minería de datos.

El módulo de selección contiene filtros para realizar o aplicar búsquedas detalladas sobre conjuntos de datos con el fin de reducir la dimensionalidad de las tablas de datos que se utilicen y al igual que el módulo de transformación buscar que esos datos sean mas reales y coherentes.

1.1.4 Módulo de Exportación.

Este módulo es el encargado de convertir las tablas de datos que reposan en el sistema por medio del sistema gestor de bases de datos a todos los formatos de archivo plano contemplados por la herramienta, todo esto con el fin de mantener una copia física de los resultados aplicados durante todo el proceso de limpieza de datos.

1.1.5 Módulo de Interfaz Gráfica.

Este módulo da soporte visual a todos los demás módulos y se encarga de brindar al usuario una experiencia muy amigable durante la manipulación de la herramienta de modo tal que resulten sencillos todos los experimentos que el usuario desee realizar mediante la interacción con todos los componentes que abarca la herramienta.

1.2 ARQUITECTURA DE PAQUETES Y CLASES

A continuación se describe de manera general la estructura de paquetes de EXDACLET y las clases que pertenecen a cada paquete. Posteriormente, se amplia y se detalla la forma como fueron desarrolladas dichas clases.

Paquete Pre-load: carga e importación de archivos. Dentro de este paquete se encuentran las clases necesarias para generar en la herramienta las tablas de datos en base a los archivos planos soportados.

Paquete Cleaning: limpieza de datos. En este paquete se encuentran las clases que implementan los filtros de limpieza esenciales para una tabla de datos.

Paquete Transform: transformación de datos. En este paquete se encuentran las clases que implementan todos los procedimientos de transformación necesarios para garantizar la integridad y veracidad de los datos, incluidos en las tablas manejadas por los usuarios.

Paquete Selection: selección de datos. En este paquete se encuentran las clases que permiten la ejecución de algoritmos para el reconocimiento de patrones en palabras (Búsqueda de: Duplicados, Homónimos, etc.).

Paquete GUI: Contiene todas las clases que implementan la interfaz gráfica de la herramienta.

1.2.1 Paquete Pre-Load.

La clase que compone este paquete es la siguiente:

Clase Cargar: Esta clase permite establecer si los archivos que el usuario intenta cargar, cumplen con alguno de los formatos soportados por EXDACLET, una vez verificado e identificado dicho formato, si el archivo es XLS o CSV, entonces ejecuta le método **XLSresumen** o **CSVresumen** respectivamente para determinar la estructura que compone al archivo que se esta importando, una vez definida su estructura entonces ejecutan los métodos **Importar CSV** o **Importar XLS;** de otra forma si el archivo es ARFF o SQL, entonces pasa directamente a la verificación de la integridad del archivo mediante los métodos **ImportarARFF** o **ImportarSQL.**

Además de los métodos anteriormente mencionados se encuentran los siguientes métodos que ayudan al proceso de importación:

- **writeFile:** Se encarga de verificar que el archivo que el usuario intenta cargar no existe físicamente en el espacio reservado en el servidor para la carga de archivos, también verifica la consistencia externa del archivo, es decir, se encarga de la forma como esta escrito su nombre, la extensión que lo acompaña y el tamaño que tiene si todo es correcto entonces el archivo se traslada de la máquina cliente al servidor.

- **cargarTD:** Si el archivo tiene la extensión XLS o CSV este método se ejecuta con el fin de mostrar un listado de tipos de datos disponibles para ser asignados a cada atributo de la tabla que se encuentra en el archivo.

- **deleteFile:** este método se ejecuta cuando, a pesar de que el archivo este físicamente bien, es decir, haya sido trasladado de la máquina cliente al servidor mediante el método **writeFile**, el usuario procede a la cancelación de la importación del mismo. En este caso, este método elimina el archivo que ha sido movido al servidor.

En la figura 2 se muestra la presentación estándar para la transferencia de archivos al servidor.

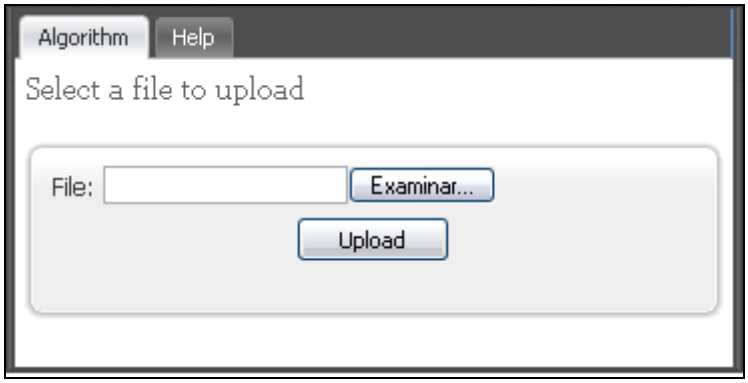


Figura 2. Ventana de selección de archivos

1.2.2 Paquete Cleaning

Este paquete se compone de las siguientes clases:

Number Null Clean: esta clase tiene como objetivo llenar todos los campos vacios de un atributo seleccionado por el analista en base a una serie de funciones estadísticas, por esta razón solamente es aplicable sobre campos de tipo numérico. Si el analista lo considera necesario es posible que el filtro pueda

ser aplicado sobre una nueva tabla de datos copia de la original manteniendo una historia de los procedimientos aplicados. Contiene los siguientes métodos:

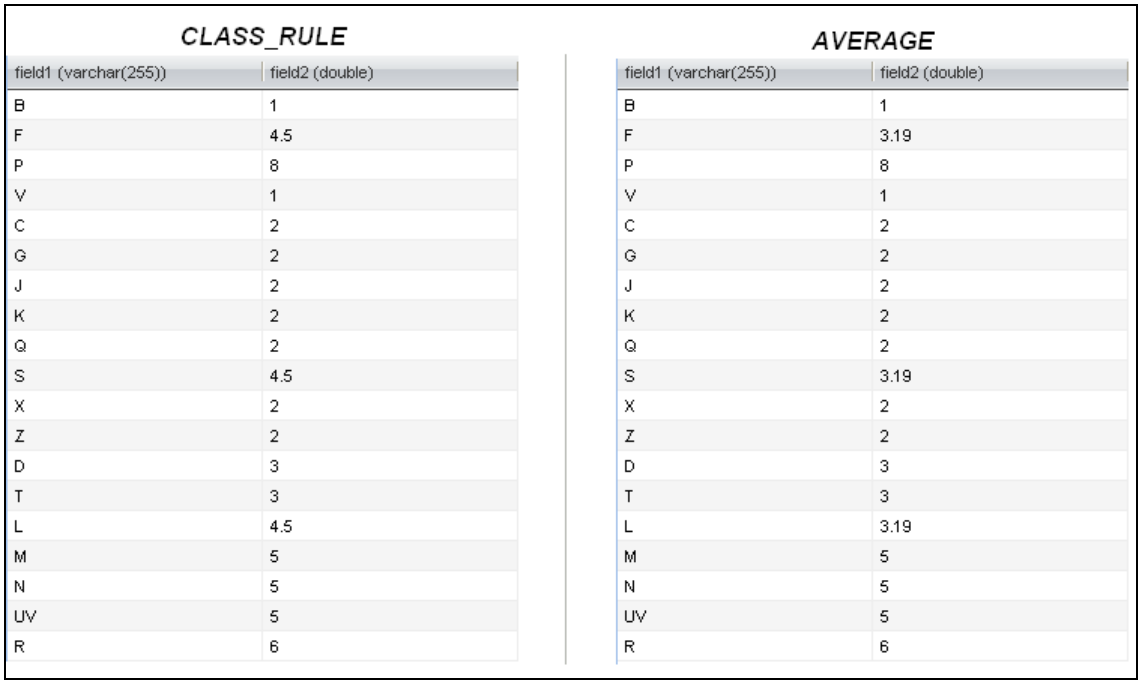
- **Average:** Completa todos los campos vacios de un atributo en base al promedio de los valores que este contiene.
- **Max:** Completa todos los campos vacios con el mayor valor encontrado en el atributo.
- **Min:** Completa todos los campos vacios con el menor valor encontrado en el atributo.
- **Supr:** Elimina todas los registros de una tabla en los cuales el atributo seleccionado contiene valores nulos.
- **Variance:** Completa todos los campos vacios de un atributo teniendo en cuenta la varianza de los valores que el atributo contiene.
- **Sum:** Completa todos los campos vacios de un atributo con el valor resultado de la suma de todos los valores que contiene el atributo.
- **Random_mid:** Completa todos los campos vacios de un atributo con un valor aleatorio entre el promedio mas o menos la desviación estándar de los valores que contiene dicho atributo.
- **Random_full:** Completa todos los campos vacios de un atributo con un valor aleatorio comprendido entre el menor y mayor valor que contenga dicho atributo.
- **Mode:** Completa todos los campos vacios de una atributo con la moda de los valores que contiene ese atributo.
- **Zero:** llena todos los campos vacios con el valor cero (0).
- **Class_rule:** Completa todos los campos vacios de un atributo con el valor resultado de dividir la suma del mayor y el menor valor que contiene el atributo entre dos (2).

En la figura 3 se muestra la tabla de datos de entrada en la que se pueden observar 3 campos vacios sobre el atributo de tipo numérico (double) field2, al aplicar el filtro NumberNullClean a través de las opciones **average** y **classRule** se

generan dos nuevas tablas de datos respectivamente, en las cuales se observa que los campos nulos han sido reemplazados en la primera por el promedio de los valores (3.19) que contiene el atributo field2 y en la segunda por la marca de clase de los mismos valores (4.5) como se puede observar en la figura 4

field1 (varchar(255))	field2 (double)
B	1
F	
P	8
V	1
C	2
G	2
J	2
K	2
Q	2
S	
X	2
Z	2
D	3
T	3
L	
M	5
N	5
UV	5
R	6

Figura 3. Datos de entrada antes de aplicar NumberNullClean



CLASS_RULE

field1 (varchar(255))	field2 (double)
B	1
F	4.5
P	8
V	1
C	2
G	2
J	2
K	2
Q	2
S	4.5
X	2
Z	2
D	3
T	3
L	4.5
M	5
N	5
UV	5
R	6

AVERAGE

field1 (varchar(255))	field2 (double)
B	1
F	3.19
P	8
V	1
C	2
G	2
J	2
K	2
Q	2
S	3.19
X	2
Z	2
D	3
T	3
L	3.19
M	5
N	5
UV	5
R	6

Figura 4. Datos de salida una vez aplicado el filtro NumberNullClean con average y class_rule

String Null Clean: esta clase permite llenar todos los campos vacios que contenga un atributo teniendo en cuenta la moda del mismo o teniendo en cuenta la moda pero basada en ciertas condiciones que uno o más del resto de atributos de la tabla deben cumplir. Esta clase se aplica para valores de tipo cadena y contiene los siguientes métodos.

- **attributeBased:** este método es el encargado de llenar todos los campos vacios del atributo con el valor más común que se encuentre en el mismo (moda). Ejemplo de funcionamiento del filtro String Null Clean attribute Based:

A continuación se presenta la tabla de datos de entrada, antes de ser aplicado el filtro String Null Clean Attribute Based. En la tabla se puede observar que en el atributo “field1” existen 4 campos vacios, como se muestra en la figura 5.

field1 (varchar(255))	field2 (varchar(255))	field3 (varchar(255))
Bajo	3	SI
Alto	5	NO
Medio	6	NO
Medio	7	SI
Bajo		NO
	9	NO
Medio	12	NO
Alto	98	SI
Medio		SI
Bajo	3	SI
	6	NO
	8	NO
Medio		SI
Alto	7	SI
	4	NO
Medio	1	NO
Alto	6	NO
Bajo	11	SI

Figura 5. Datos de entrada filtro String Null CLean

Al aplicar el filtro, se reemplazan todos los campos vacios por “Medio”, la tabla resultante una vez se ha aplicado el filtro se muestra en la figura 6.

field1 (varchar(255))	field2 (varchar(255))	field3 (varchar(255))
Bajo	3	SI
Alto	5	NO
Medio	6	NO
Medio	7	SI
Bajo		NO
Medio	9	NO
Medio	12	NO
Alto	98	SI
Medio		SI
Bajo	3	SI
Medio	6	NO
Medio	8	NO
Medio		SI
Alto	7	SI
Medio	4	NO
Medio	1	NO
Alto	6	NO
Bajo	11	SI

Figura 6. Datos de salida filtro String Null CLean

- **conditionBased:** este método se encarga de llenar los campos vacios de un atributo teniendo en cuenta las condiciones que el usuario establece y que

deben cumplir uno o mas atributos. Es decir, extrae el valor que teniendo en cuenta una serie de condiciones es el más común para ese atributo y lo llena en todos los campos vacios que contenga el mismo.

Teniendo en cuenta el mismo conjunto de datos de entrada tomado en el ejemplo anterior se establece como condición de que el atributo “field2” sea igual a 3. De esta forma los campos vacios del atributo “field1”, son reemplazados por “Bajo”, dado que el valor más común del atributo “field1” cuando el campo “field2” es igual a 3 es “Bajo”. La tabla resultante una vez se ha aplicado el filtro se muestra en la figura 7

field1 (varchar(255))	field2 (varchar(255))	field3 (varchar(255))
Bajo	3	SI
Alto	5	NO
Medio	6	NO
Medio	7	SI
Bajo		NO
Bajo	9	NO
Medio	12	NO
Alto	98	SI
Medio		SI
Bajo	3	SI
Bajo	6	NO
Bajo	8	NO
Medio		SI
Alto	7	SI
Bajo	4	NO
Medio	1	NO
Alto	6	NO
Bajo	11	SI

Figura 7. Datos de salida filtro String Null CLean (condition based)

Trim: esta clase permite eliminar todos los espacios en blanco que se encuentren al comienzo y al final de cadenas de texto de los atributos seleccionados por el analista.

Missing (Expert Rule Editor): Esta clase tiene como objetivo la revisión de cada una de las sentencias que digita el analista a manera de condicional “if”, analizando que estas sentencias no contengan código no adecuado, permitiendo comandos básicos para asignaciones, operaciones y observación de resultados.

El modulo principal se encuentra en el método **missing**, requiere como parámetros el nombre de la tabla que será objeto del o de los condicionales, además de el texto a analizar. Este método retorna una serie de consultas que son ejecutadas individualmente.

Hay un tipo de condiciones que se ejecutan de forma particular, y estas son las condiciones que involucran ver un resumen de los datos desplegados en pantalla, para esto se utiliza el método **ShowTmpTable** cuyo objetivo es crear una tabla temporal que contenga los datos resultado de la consulta, para que luego el modulo de interfaz se encargue de mostrar esta tabla en pantalla.

Un ejemplo del funcionamiento de esta clase se comenta a continuación, en donde el analista tiene como datos de entrada una tabla con 3 atributos (Nombre, Grado, Estudios) que se muestra en la figura 8, y su objetivo es completar los campos vacios del atributo Estudios, teniendo en cuenta las condiciones que se muestran en la figura 9.

Nombre (varchar(255))	Grado (double)	Estudios (varchar(255))
Eduardo	5	
Juan	4	
Franco	8	
Marco	3	
Miguel	11	

Figura 8. Datos de Entrada (Rule Editor)

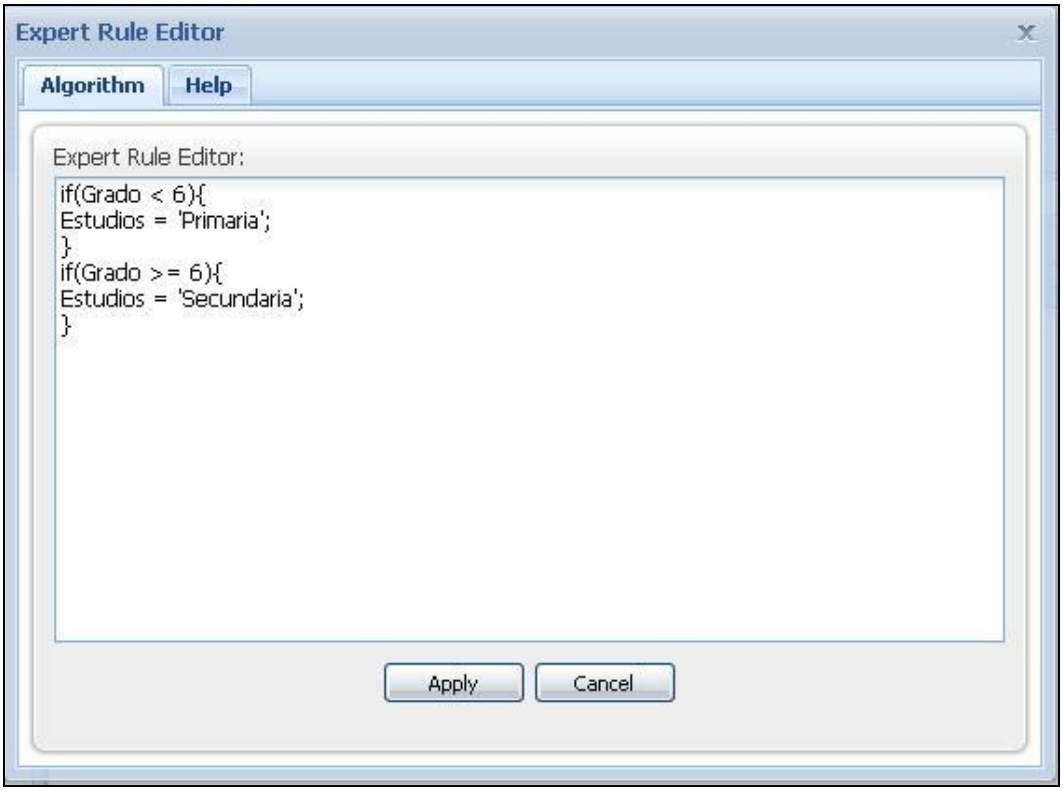


Figura 9. Condiciones (Rule Editor)

Una vez aplicado el filtro, la tabla resultante se muestra en la figura 10, en donde todos los estudiantes que pertenecen a los grados inferiores, es decir, del grado 5 hacia abajo, son de Primaria y los demás son de Secundaria.

Nombre (varchar(255))	Grado (double)	Estudios (varchar(255))
Eduardo	5	Primaria
Juan	4	Primaria
Franco	8	Secundaria
Marco	3	Primaria
Miguel	11	Secundaria

Figura 10. Datos de Salida (Rule Editor)

1.2.3 Paquete Transform

Este paquete está conformado por las siguientes clases con sus respectivos métodos:

Discretize: Esta clase se encarga de ejecutar el filtro de discretización sobre un atributo de tipo numérico mediante el método **Discretizar** el cual recibe como parámetros el nombre del atributo que se desea discretizar, la tabla a la cual pertenece dicha columna, el número de intervalos (bins) en los cuales se desea discretizar el atributo y una conexión al sistema gestor de base de datos usando los métodos abrir(), query(), fetch() y cerrar() de la clase **MyConn,** a través de ella se extraen los datos originales para ser tratados y posteriormente ser actualizados de acuerdo a los resultados que genere este método durante su ejecución.

Un ejemplo de aplicación del filtro se observa en la figura 11 en donde un atributo de tipo numérico (field2) es seleccionado en base a la tabla de la figura 12, los resultados una vez aplicado el filtro se muestran en la figura 13.

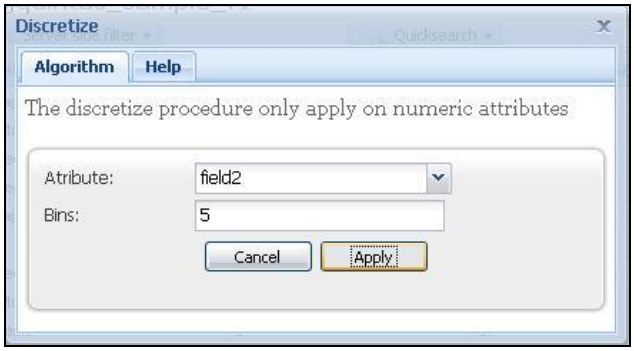


Figura 11. Filtro Discretize

field1 (varchar(255))	field2 (double)
Bajo	3
Alto	5
Medio	6
Medio	7
Bajo	0
	9
Medio	12
Alto	98
Medio	0
Bajo	3
	6
	8
Medio	0
Alto	7
	4
Medio	1
Alto	6
Bajo	11

Figura 12. Datos de entrada filtro Discretize

field1 (varchar(255))	field2 (varchar(255))
Bajo	[0 - 19.6]
Alto	[0 - 19.6]
Medio	[0 - 19.6]
Medio	[0 - 19.6]
Bajo	[0 - 19.6]
	[0 - 19.6]
Medio	[0 - 19.6]
Alto	(78.4 - 98]
Medio	[0 - 19.6]
Bajo	[0 - 19.6]
	[0 - 19.6]
	[0 - 19.6]
Medio	[0 - 19.6]
Alto	[0 - 19.6]
	[0 - 19.6]
Medio	[0 - 19.6]
Alto	[0 - 19.6]
Bajo	[0 - 19.6]

Figura 13. Datos de salida filtro Discretize

Una vez aplicado el filtro sobre el conjunto de datos de la figura 12, se observa en la figura 13 que a pesar de haber sido especificados 5 intervalos (bins) la tabla resultante solamente contiene dos intervalos, todo esto, debido a que la cantidad

de intervalos generados son completamente independientes de la cantidad de valores, pero los límites superiores e inferiores si dependen exclusivamente de los valores máximo y mínimo que contiene el conjunto de datos. Por esta razón esos dos intervalos hacen referencia al primer y último intervalo dado que no existen valores que se acomoden dentro de los 3 intervalos restantes.

Normalize: Esta clase ejecuta el filtro de normalización de valores de un atributo de tipo numérico mediante el método **Normalize** el cual recibe como parámetros el atributo al cual se le desea aplicar el filtro, el nombre de la tabla a la cual pertenece el atributo y una conexión en base a la cual se ejecutan las consultas necesarias para establecer el promedio y la desviación estándar de los valores que componen el atributo para posteriormente actualizarlos teniendo en cuenta la fórmula de normalización que se muestra en la figura 14.

$$Z = \frac{X - \mu}{\sigma}$$

Figura 14. Fórmula de normalización

En donde:

Z = Valor a obtener

X = Valor a normalizar

μ = Media

σ = Desviación Estándar.

CharReplace: Esta clase permite la aplicación del filtro de transformación para reemplazar caracteres que se encuentren en uno o más atributos, por un caracter o cadena de caracteres proporcionada por el analista, todo esto con el fin de transformar y estandarizar una tabla de datos. Para desarrollar esta tarea, se utiliza el método **CharReplace** el cual recibe como parámetros el o la cadena de caracteres a buscar, el o la cadena de caracteres por la cual serán reemplazadas las apariciones del primer atributo, un vector con los nombres de los atributos

sobre los cuales se aplicará la búsqueda, el nombre de la tabla a la cual pertenecen esos atributos y una conexión que servirá para realizar la consulta de búsqueda de los caracteres y su posterior reemplazo. En la figura 15 se muestra el conjunto de datos de entrada antes de ser aplicado este filtro. En él se encuentran 4 valores iguales dentro del atributo field1 (V___V), que serán reemplazados por la cadena (AA), el resultado una vez aplicado este filtro se muestra en la figura 16

field1 (varchar(255))	field2 (double)
V___V	1
F	3.19
P	8
V___V	1
C	2
G	2
J	2
K	2
Q	2
S	3.19
X	2
V___V	2
D	3
T	3
L	3.19
M	5
V___V	5
UV	5
R	6

Figura 15. Datos de entrada antes de ejecutar el filtro CharReplace

field1 (varchar(255))	field2 (double)
AA	1
F	3.19
P	8
AA	1
C	2
G	2
J	2
K	2
Q	2
S	3.19
X	2
AA	2
D	3
T	3
L	3.19
M	5
AA	5
UV	5
R	6

Figura 16. Datos de salida una vez aplicado el filtro CharReplace

UpperCase: Esta clase se encarga de transformar todos los valores de los atributos de tipo cadena seleccionados por el analista a Mayúsculas, todo esto como un punto de inicialización para lograr la estandarización de los valores que contiene una determinada tabla de datos. La transformación se realiza mediante el método **toUpper** el cual recibe como parámetros los atributos a los cuales se les quiere hacer la transformación y una conexión mediante la cual se realizará la consulta al Sistema Gestor de Base de Datos usando los métodos abrir(), query(), fetch() y cerrar() de la clase **MyConn,** para realizar la transformación de los datos. En la tabla 1 se muestra un ejemplo de transformación de cadenas de texto a mayúsculas.

Tabla 1. Tabla de ejemplo para el filtro UpperCase

Valor Inicial	Valor Final
Jhonnathan	JHONNATHAN
RoBeRt	ROBERT
LiliaN	LILIAN
uPPer cAse convert	UPPER CASE CONVERT

LowerCase: Esta clase se encarga de transformar todos los valores de los atributos de tipo cadena seleccionados por el usuario a Minúsculas, este resulta otro método de estandarización de valores que contiene una determinada tabla de datos. La transformación se realiza mediante el método **toLower** que recibe como parámetros al igual que el método toUpper de la clase **UpperCase,** los atributos a los cuales se les quiere hacer la transformación y una conexión mediante la cual se realizará la consulta al Sistema Gestor de Base de Datos usando los métodos abrir(), query(), fetch() y cerrar() de la clase **MyConn,** para realizar la transformación de los datos. En la tabla .2 se muestra un ejemplo de transformación de cadenas de texto a minúsculas.

Tabla 2. Tabla de ejemplo para el filtro LowerCase

Valor Inicial	Valor Final
Jhonnathan	jhonnathan
RoBeRt	robert
LOUIS	lilian
uPPer cAse convert	upper case convert

TruncateString: Esta clase se encarga de aplicar el filtro de transformación mediante el cual se cortan las cadenas de un determinado atributo en base a un tamaño fijo de caracteres y teniendo en cuenta la orientación en la cual serán cortados dichos caracteres. Este filtro se encuentra en el método **TruncateSTR** el cual recibe como parámetros, el atributo sobre el cual será aplicado el filtro, el nombre de la tabla a la cual pertenece dicho atributo, la cantidad de caracteres que serán eliminados de las cadenas que componen el atributo, la orientación de la eliminación que hace referencia si los caracteres serán eliminados por la parte izquierda o derecha de la cadena y una conexión con el Sistema Gestor de Base de Datos para actualizar la tabla de datos que se esta utilizando. Un ejemplo del funcionamiento de eta clase se muestra en la tabla 3 la cual consta de cuatro columnas, la primera muestra la cadena original, la segunda el número de caracteres a eliminar y en las dos restantes el resultado de la eliminación por derecha y por izquierda respectivamente.

La conexión con el Sistema Gestor de Base de Datos se realiza a través del uso de la clase **MyConn.**

Tabla 3. Ejemplo TruncateString

Original String	Length	Left-Truncate	Rigth-Truncate
12345ASD	3	45ASD	12345
UNIVERSALPIC	6	SALPIC	UNIVER
WIRELESS	2	RELESS	WIRELE

NonPrintable: Esta clase es la encargada de buscar dentro de una tabla de datos seleccionada por el analista, todos aquellos caracteres que resulten ser no imprimibles, es decir, que no son caracteres alfa-numéricos, con el fin de que sean mostrados dentro de una tabla temporal editable que permitirá el reemplazo de todos y cada uno de los caracteres encontrados por los que resulten ser convenientes para el analista, todo esto se logra gracias a la interacción entre el método **nonprintable** perteneciente a esta clase y los métodos **algDialog** de la clase **AlgDialog** y **standargrid** y **define_grid** de la clase **gridClass**. La descripción de los métodos de las clases **algDialog** y **gridClass** se describen en la sección **1.2.5 Paquete GUI.**

Los parámetros que recibe el método **nonprintable** son la lista de atributos en los cuales se buscarán los caracteres no imprimibles, el nombre de la tabla a la cual pertenecen tales atributos y una conexión con el Sistema Gestor de Base de Datos usando los métodos abrir(), query(), fetch() y cerrar() de la clase **MyConn,** que permite la creación de la tabla temporal y la búsqueda campo a campo de todos los caracteres no imprimibles. En la tabla 4 se muestra un listado con algunos de los posibles caracteres no imprimibles contemplados en EXDACLET

Tabla 4. Tabla de caracteres no imprimibles

CARACTER	ASCII	CARACTER	ASCII	CARACTER	ASCII	CARACTER	ASCII
☺	1	“	34	ÿ	152	┬	194
☻	2	#	35	ø	155	├	195
♥	3	$	36	£	156	─	196
♦	4	%	38	Ø	157	┼	197
♣	5	‘	39	×	158	ã	198
♠	6	(	40	ƒ	159	╚	200
•	7	)	41	ª	166	╔	201
◘	8	*	42	º	167	╩	202
○	9	+	43	¿	168	╦	203
◙	10	-	45	®	169	╠	204
♂	11	<	60	¬	170	═	205

♀	12	=	61	½	171	╬	206
♪	13	>	62	¼	172	¤	207
♫	14	¿	63	¡	173	ð	208
☼	15	[	91	«	174	Ð	209
►	16	\	92	»	175	┘	217
◄	17	]	93	░	176	┌	218
↕	18	^	94	▒	177	█	219
‼	19	~	126	▓	178	▄	220
¶	20	⌂	127	│	179	¦	221
§	21	â	131	┤	180	▌	223
▬	22	å	134	©	184	ß	225
↨	23	ê	136	╣	185	õ	228
↑	24	è	138	║	186	µ	230
↓	25	î	140	╗	187	þ	231
→	26	æ	145	╝	188	Þ	232
←	27	Æ	146	¢	189	□	233
∟	28	ô	147	¥	190	ý	236
↔	29	ò	149	┐	191	±	241
▲	30	û	150	└	192	¾	243
▼	31	ù	151	┴	193	÷	246

ChangeAttributeType: Esta clase permite la transformación del tipo de datos que tiene un atributo a uno especificado por el analista, el principal objetivo se centra en la estandarización de los datos que componen dicho atributo. Los métodos que hacen parte de esta clase son:

***ToString*:** el cual transforma todos los valores que contenga el atributo a tipo cadena, este método no realiza ningún tipo de verificación sobre los datos dado que cualquier tipo de datos al cual se encuentre asociado un atributo puede ser transformado a cadena.

ChangeColumnToDouble: recibe como parámetros el nombre del atributo que se quiere verificar, el nombre de la tabla de datos a la cual está asociado el atributo y una conexión con el Sistema Gestor de Base de Datos mediante la clase **MyConn**. Es el encargado de generar una tabla temporal con todos los datos que tras un análisis interno sobre la consistencia de los mismos, no cumplieron con el tipo de formato especificado por el analista, esta tabla temporal es editable solamente en el atributo que se quiere transformar. Una vez tratados los valores que en un principio no cumplieron con el nuevo tipo de datos, se procede a analizar a través de este método si nuevamente todos los valores cumplen con el tipo de dato, el proceso se repite indefinidamente hasta que todos los valores del atributo estén correctos.

Si al pasar por este método no se encontraron valores que no coincidieran con el nuevo tipo de datos especificado o si los hubo, estos fueron tratados (corregidos o eliminados) entonces mediante la conexión se actualizan los valores del atributo y su tipo.

Cuando el analista requiere transformar un atributo a tipo Date, DateTime o Time (formatos de fecha, fecha y tiempo o tiempo), esta clase ejecuta un método intermedio llamado **VerifyChange**, el cual genera una interfaz de usuario que se muestra en la figura 17 en la cual se especifican los delimitadores que tiene los valores que componen el atributo a transformar, una vez especificados estos valores, la clase lanza los métodos correspondientes al cambio de tipo de datos seleccionados. Si el cambio es a tipo Date, entonces se ejecuta el método **ChangeColumnToDate** que recibe como parámetros, el nombre del atributo a transformar, el nombre de la tabla asociada al atributo, el formato en el cual se encuentra la fecha, el delimitador de fecha y una conexión con el Sistema Gestor de Base de Datos resultado de ser instanciada mediante el método abrir () de la clase **MyConn**. Este método se encarga de verificar que todos los valores que se encuentran en el atributo cumplan con alguno de los formatos válidos existentes para la escritura de un valor de tipo fecha. Si durante este proceso aparecen

valores que no cumplen con la estructura de fecha válida entonces este método genera una tabla temporal editable con los valores que no cumplieron con el formato del nuevo tipo de datos, cuando estos valores son corregidos o no se encontraron errores durante el proceso mencionado anteriormente, entonces se llama al método **TransformToDate** que es el encargado de convertir todos los valores de fecha encontrados en base a los delimitadores y formatos en los cuales fueron especificados, al formato estándar utilizado en EXDACLET para el manejo de valores de tipo fecha (date). Una vez transformados, se procede a cambiar el tipo de datos.

Si el cambio es a tipo Datetime o Time, entonces se ejecutan los métodos **ChangeColumnToDateTime** y **ChangeColumnToTime** respectivamente, que de la misma forma que el método **ChangeColumnToDate** verifican la integridad de los valores que se encuentran en el atributo. Si se encuentran inconsistencias, entonces el analista revisa y corrige dichas inconsistencias. Una vez que todos los valores son correctos, entonces se pasa a los métodos **TransformToDateTime y TransformToTime** respectivamente los cuales transforman todos esos valores y los estandarizan para ser ingresados en la tabla de datos.

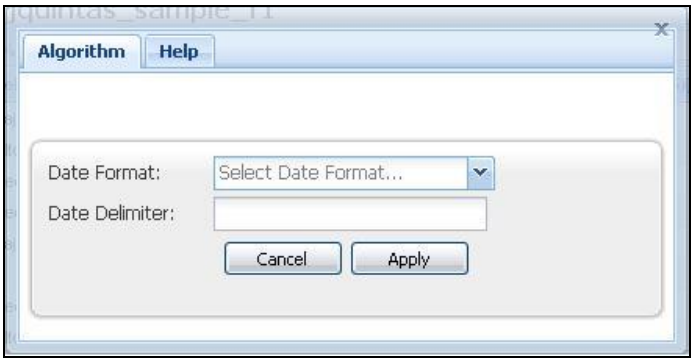


Figura 17. Interfaz de definición de delimitador de fechas.

ChangeAttributeName: Esta clase es la encargada de cambiar el nombre de un atributo mediante el método **ChangeAttName,** el cual recibe como parámetros el nombre del atributo a transformar, el nombre de la tabla a la cual se encuentra

asociado el atributo, el nuevo nombre especificado por el analista y una conexión. El curso normal de los eventos empieza cuando el usuario selecciona el atributo al cual le quiere cambiar el nombre y específica el nuevo nombre que le será asignado, ese nuevo nombre es verificado con el fin de que no exista un atributo igual. El paso siguiente consiste en determinar que la nueva cadena este correcta en su estructura y finalmente se procede a actualizar el atributo si durante todo el proceso ocurre un error, la operación es abortada y se muestra un mensaje de error.

AddAtribute: Esta clase permite agregar un atributo a una determinada tabla de datos mediante el método **AddAtt** que se compone de los siguientes parámetros:
TableName: hace referencia al nombre de la tabla en la cual será adicionado el atributo.
AttributeName: hace referencia al nombre del atributo que será adicionado.
AttributeType: en el que se define el tipo de datos que identifica al nuevo atributo.
Connection: mediante la cual se establece una conexión con el Sistema Gestor de Base de Datos para actualizar la tabla de datos que se está modificando, usando el método query() de la clase **MyConn**.

El flujo normal de los eventos comienza cuando el usuario define el nombre que identificará al nuevo atributo y el tipo de datos que le será asignado, durante el evento de escritura del nuevo nombre se verifica que ese nombre no exista dentro de la misma tabla de datos, posteriormente y mediante la conexión se establecen los cambios pertinentes sobre la tabla con una sentencia SQL. Si todo es correcto la tabla de datos se recargará nuevamente de otro modo se mostrará un mensaje de error.

DeleteAttribute: Esta clase se encarga de eliminar un atributo de una determinada tabla de datos mediante el método **DeleteAtt** el cual recibe como parámetros el nombre del atributo a eliminar, la tabla de datos a la que pertenece ese atributo y una conexión con el Sistema Gestor de Base de Datos mediante la

cual se ejecuta la sentencia SQL que elimina el atributo de esa tabla de datos usando el método query() de la clase **MyConn**. El curso normal de los eventos se desarrolla cuando el analista carga una tabla de datos, selecciona el filtro de transformación **Delete Attributes** y escoge el atributo que desea eliminar, si todo es correcto se recargará la tabla de datos. Si ocurre un error se abortará la operación y se mostrará un mensaje de error.

ClearAttribute: Esta clase se encarga de limpiar todo el contenido de un determinado atributo sin tener en cuenta el tipo de datos que lo identifica a través del método **AttClear** el cual se ejecuta teniendo en cuenta los siguientes parámetros:
*AttName**:*** que contiene el nombre del atributo al cual se le quiere vaciar su contenido.
TableName: que contiene el nombre de la tabla en donde se encuentra el atributo a limpiar.
Connection: que es una conexión con el Sistema Gestor de Base de Datos con el fin de ejecutar la consulta SQL para limpiar el contenido del atributo y actualizar posteriormente la tabla de datos utilizando el método query() de la clase **MyConn***.*

El curso normal de los eventos inicia en el momento en que el usuario carga una tabla de datos y selecciona el filtro de transformación **Attribute Clear**, posteriormente selecciona el atributo al cual quiere vaciarle su contenido y procede a aplicar el filtro. Si durante el proceso hubo un error se aborta la operación y se muestra un mensaje de error de otro modo se actualiza la tabla de datos cargada.

Binarize: Esta clase es la encargada de transformar los atributos seleccionados por parte del analista en una nueva tabla la cual se compone de valores unos (1) y ceros (0) y de una serie de atributos basados en los diferentes valores que se encuentren en los atributos seleccionados. Esta clase se compone del método **MakeIt** el cual recibe como parámetros el nombre de la tabla de datos original, el

nombre de la nueva tabla de datos binarizada, un vector con los atributos que se van a binarizar y una conexión al Sistema Gestor de Base de Datos instanciada por el método abrir() de la clase **myConn**.

El curso normal de los eventos se desarrolla cuando el analista carga una tabla de datos, selecciona el filtro de transformación **Binarize,** luego escoge los atributos que desea binarizar, en base a estos se crea la nueva tabla de datos, cuando aplica el filtro, el método **MakeIt** toma los atributos e identifica todos los diferentes valores que estos contienen con el fin de ir creando una estructura de tabla en donde cada valor se convertirá en un nuevo atributo, hecho esto, este método recorre toda la tabla de datos original fila a fila y compara los valores con la nueva tabla creada, esto significa que, cada valor de la tabla original es buscado como un atributo de la nueva tabla, si existe alguna coincidencia entonces para ese atributo es insertado el valor uno (1) de otro modo, se inserta el valor cero (0).
Una vez es terminado el proceso, se carga la nueva tabla de datos con todos los valores binarizados.

Un ejemplo del funcionamiento se muestra en las tablas 5 y 6 en donde se aprecia el resultado de binarizar los datos de la tabla 5 en la tabla 6.

Tabla 5. Datos de entrada (binarize)

C1	C2	C3	C4	C5
A	B	D	B	A
B	C	A	C	D
C	D	A	B	D

Tabla 6. Datos de salida (binarize)

C1_A	C1_B	C1_C	C2_B	C2_C	C2_D	C3_D	C3_A	C4_B	C4_C	C5_A	C5_D
1	0	0	1	0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	1	0	1	0	1
0	0	1	0	0	1	0	1	1	0	0	1

Table Encoder/Decoder: Esta clase se encarga de transformar una tabla de datos en una tabla con valores codificados, estos valores son números consecutivos teniendo en cuenta los diferentes valores de la tabla original y son soportados mediante una tabla denominada **Diccionario de Datos** en la cual se describen los diferentes valores encontrados en la tabla original y su correspondiente código asociado a la nueva tabla creada. Además se encarga de transformar una tabla ya codificada a sus valores originales teniendo en cuenta una tabla de **Diccionario de Datos** en especial. Los métodos que hacen parte de esta clase son los siguientes:

EncodeAttribute: el cual recibe como parámetros un vector con los atributos a codificar, el nombre de la tabla asociada a esos atributos y una conexión. Su funcionamiento se basa en extraer todos los diferentes valores que existen en los atributos seleccionados y asignarle un código único y consecutivo a cada uno. Estos datos (valor, código y atributo) son consignados en un **Diccionario de Datos** que a su vez es una tabla. Posteriormente se reemplazan todos los valores en los atributos seleccionados, por los códigos ya asignados en el **Diccionario de Datos.**

DecodeAttribute: el cual recibe como parámetros el nombre de la tabla de datos a decodificar, el nombre del diccionario de datos que tendrá como base la decodificación y una conexión al Sistema Gestor de Base de Datos. Su funcionamiento se centra en reemplazar los códigos que se encuentran en la tabla codificada con los valores originales que se encuentran consignados en el diccionario de datos.

Un ejemplo del funcionamiento de esta clase se presenta en las tablas 7, 8 y 9 en donde se muestran la tabla original, la tabla codificada y el diccionario de datos utilizado para la codificación.

Tabla 7. Tabla de datos original (encode)

Estado_Civil	Grado_Estudios
Casado	Primaria
Soltero	Secundaria
Casado	Secundaria
Viudo	Primaria

Tabla 8. Tabla de datos codificada (encode)

Estado_Civil	Grado_Estudios
1	4
2	5
1	5
3	4

Tabla 9. Diccionario de datos codificada (encode)

Codigo	Atributo	Valor
1	Estado_Civil	Casado
2	Estado_Civil	Soltero
3	Estado_Civil	Viudo
4	Grado_Estudios	Primaria
5	Grado_Estudios	Secundaria

Todas las clases que se encuentran dentro de los paquetes de Cleaning (limpieza) y transform (transformación) tienen un paso intermedio de verificación en donde se pregunta si el filtro será aplicado sobre la misma tabla de datos que se este utilizando en el momento, o si se aplicará en una copia de la misma con un nombre definido por parte del analista como se muestra en la figura 18. Todo esto con el fin de que se mantenga una historia de todos los procedimientos que el analista ha aplicado sobre la tabla desde el momento en que esta es cargada en la herramienta mediante alguno de los formatos de archivo valido soportado.

Este procedimiento lo realiza el método **Verify** que se encuentra en la clase **Algorithms**, encargada de crear todas las instancias de clases requeridas durante

todo el proceso de limpieza y transformación dependiendo de los filtros y procedimientos que el analista desee utilizar. En caso de ser confirmado el evento de aplicación del filtro sobre una nueva tabla en el método **Verify**, se ejecutan los métodos **CreateDup** y **genDup** los cuales se encargan de mostrar la interfaz de usuario para definir el nuevo nombre de la tabla sobre la cual será aplicado un filtro y crear físicamente dentro de la herramienta la tabla de datos definitiva sobre la cual será aplicado el filtro seleccionado por el analista.

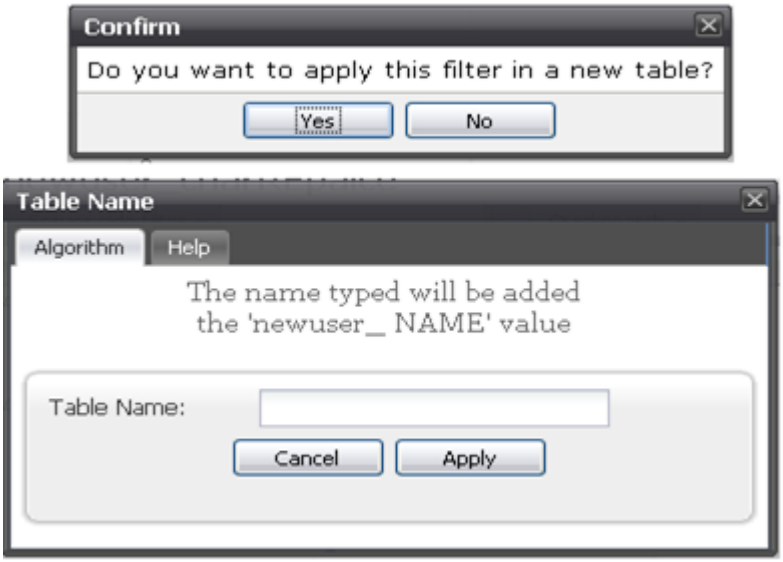


Figura 18. Interfaces de confirmación y creación de una nueva tabla

1.2.4 Paquete Selection

MaxMinLength: Esta clase es la encarga de ejecutar los filtros de selección **MaxLength** y **MinLength** mediante el método **MMLength.** Este método es el encargado de generar una tabla de datos temporal que se muestra a través de la clase **StandarGrid** y su método **defineGrid**. Esta tabla temporal contiene todas las tuplas que cumplen con la condición que se establece al ser aplicado alguno de los dos filtros. Esta condición viene definida por uno de los parámetros que recibe el método MMLength denominado ***type***, en el cual se establece si se aplicará el filtro MaxLength o MinLength. A demás de este parámetro, el método recibe el nombre del atributo en el cual se aplicará el filtro, el nombre de la tabla

asociada a dicho atributo y una conexión que permite la búsqueda y la posterior creación de la tabla de datos temporal.

El curso normal de los eventos para este filtro comienza cuando el analista carga una tabla de datos en la herramienta y procede a seleccionar el filtro de selección **MaxLength** o **MinLength,** en donde se debe especificar el atributo de búsqueda y el tamaño mínimo que deben tener las cadenas de caracteres para ser mostradas dentro de la tabla temporal. Si todo es correcto entonces se generará dicha tabla la cual será editable, de otro modo se abortará la operación y se mostrara un mensaje de error.

JaroWinkler: esta clase es la encargada de generar una tabla temporal que contiene varios conjuntos de tuplas, cada conjunto separado por una fila delimitadora. Estos conjuntos son el resultado de todas las posibles coincidencias que pueden existir en base al algoritmo Jaro-Winkler, entre los valores que se encuentran dentro de uno o más atributos seleccionados por el analista. Esta tabla se muestra a través de la clase **StandarGrid** y su método **defineGrid.**

La clase **JaroWinkler** se compone de los métodos: **MakeJaroWinkler, GetDistance y ClearString.** El primero de ellos es el encargado de realizar todos los procedimientos requeridos para la extracción de cada valor de los atributos seleccionados, luego llama al método **ClearString** que elimina la posible basura que pueda contener cada valor y posteriormente al método **GetDistance** el cual ejecuta internamente el algoritmo de Jaro-Winkler, extrae las distancias y analiza la posible similitud entre dos valores si esos dos valores tienen una similitud lo suficientemente alta, entonces son insertados con todos sus respectivos atributos en la tabla temporal de otro modo seguirá buscando por toda la tabla de datos original hasta encontrar valores similares.

Además, la tabla de datos temporal generada tiene la posibilidad de ser tratada, es decir, una vez ejecutado este filtro, el analista verifica y toma medidas sobre los datos que realmente son considerados duplicados.

En la figura 19 se muestra el pseudocódigo de la implementación de este algoritmo.

```
Array = Lista de Transacciones
Fila = Transacción actual
ResToShow = Objeto con la lista de resultados
Contador = 0
while (Fila = ResToShow){
    Array = Fila
    for(z = 0; z < Contador; z++){
        Word = Array[z]
        Longitud1 = len(Word)
        Longitud2 = len(Fila)
        if(Longitud1 > Longitud2){
            aux = Longitud2     Longitud2 = Longitud1       Longitud1 = aux
            auxstr = Fila       Fila = Word     Word = auxstr
        }
        Lmin = Longitud1
        Lmax = Longitud2
        for(i = 0; i < Longitud1; i++){     f1[i] = false       }
        for(i = 0; i < Longitud2; i++){     f1[i] = false       }
        m = ceil((Lmax/2)-1)
        common = 0
        tr = 0
        a1 = str_split(Word)        a2 = str_split(Fila)
        for (i = 0; i < Longitud1; i++){
            if(m >= i){
                f = 0
                l = i + m
            }else{
                f = i - m
                l = i + m
            }
            if(l > lmax){   l = lmax    }
            for (j = f; j < l; j++){
                if(a2[j] == a1[i] && f2[j] == false){
                    common = common + 1
                    f1[i] = true
                    f2[j] = true
                    break
                }
            }
        }
        l = 0
        for(i = 0; i < Longitud1; i++){
            if(f1[i] == true){
                for(j = l; j < Longitud2;j++){
                    if(f2[§j] == true){
                        l = j + 1;
                        if(a1[i] != a2[j]){     tr = tr + 0.5       }
                        break
                    }
                }
            }
        }
        wcd = 1/3       wrd = 1/3       wtr = 1/3
        if(common != 0){
            jaro = wcd * common / l1 + wrd * common / l2 + wtr * (common - tr) / common
        }else{
            jaro = 0;
        }
    }
    Contador++
}
```

Figura 19. Pseudocódigo algoritmo Jaro-Winkler

DoubleMetaphone: Esta clase tiene el mismo principio de funcionamiento que tiene la clase **JaroWinkler** la diferencia se enmarca en los métodos que son llamados a ejecución y el procedimiento de ejecución del algoritmo, puesto que los resultados son generados y pueden ser tratados de la misma forma que la clase anteriormente mencionada.

Los métodos que componen a esta clase son: **DMP,** el cual recibe como parámetros una lista con los atributos sobre los cuales se aplicará el filtro, la tabla asociada a ese grupo de atributos y una conexión con el Sistema Gestor de Base de Datos para poder crear e insertar registros en la tabla temporal generada mediante el método query de la clase **MyConn**. Su objetivo es ir extrayendo y cargando en memoria todas las tuplas de la tabla original en base a los atributos seleccionados por el analista para proceder a ejecutar el método **GetDoubleMetaPhone** en donde se pasa como parámetro el valor de cada atributo para ser codificado según el algoritmo DoubleMetaphone y posteriormente buscar todas las posibles coincidencias con el nuevo valor generado. Este procedimiento se repite hasta haber extraído todas las tuplas de la tabla original, haberlas cargado en memoria y analizarlas para su evaluación. En la figura 20 se muestra una sección del código para la implementación del algoritmo. En ella se puede apreciar que existen cerca de 100 líneas de código para el caso de que una palabra contenga la letra “C”, y también, que se contemplan una gran variedad de casos para posibles variaciones solo de esa letra. Dentro del código completo se estableció que el total de líneas de código para el caso de la letra “C” es un total de 180 y en general para todo el algoritmo aproximadamente 1000. En las que se consignan una gran cantidad de posibles variaciones para todas las letras del abecedario contemplando también diferencias de idiomas.

```
201             // special case 'caesar'
202             if (($this->current == 0)
203                 && $this->StringAt($this->original, $this->current, 6,
204                             array("CAESAR"))) {
205               $this->primary   .= 'S';
206               $this->secondary .= 'S';
207               $this->current += 2;
208               break;
209             }
210             // italian 'chianti'
211             if ($this->StringAt($this->original, $this->current, 4,
212                             array("CHIA"))) {
213               $this->primary   .= 'K';
214               $this->secondary .= 'K';
215               $this->current += 2;
216               break;
217             }
218             if ($this->StringAt($this->original, $this->current, 2, array("CH"))) {
219               // find 'michael'
220               if (($this->current > 0)
221                   && $this->StringAt($this->original, $this->current, 4,
222                             array("CHAE"))) {
223                 $this->primary   .= 'K';
224                 $this->secondary .= 'X';
225                 $this->current += 2;
226                 break;
227               }
228               // greek roots e.g. 'chemistry', 'chorus'
229               if (($this->current == 0)
230                   && ($this->StringAt($this->original, $this->current + 1, 5,
231                             array("HARAC", "HARIS"))
232                       || $this->StringAt($this->original, $this->current + 1, 3,
233                                 array("HOR", "HYM", "HIA", "HEM")))
234                   && !$this->StringAt($this->original, 0, 5, array("CHORE"))) {
235                 $this->primary   .= 'K';
236                 $this->secondary .= 'K';
237                 $this->current += 2;
238                 break;
239               }
240
241               // germanic, greek, or otherwise 'ch' for 'kh' sound
242               if (($this->StringAt($this->original, 0, 4, array("VAN ", "VON "))
243                    || $this->StringAt($this->original, 0, 3, array("SCH")))
244                   // 'architect' but not 'arch', orchestra', 'orchid'
245                   || $this->StringAt($this->original, $this->current - 2, 6,
246                             array("ORCHES", "ARCHIT", "ORCHID"))
247                   || $this->StringAt($this->original, $this->current + 2, 1,
248                             array("T", "S"))
249                   || (($this->StringAt($this->original, $this->current - 1, 1,
250                             array("A","O","U","E"))
251                        || ($this->current == 0))
252                       // e.g. 'wachtler', 'weschsler', but not 'tichner'
253                       && $this->StringAt($this->original, $this->current + 2, 1,
254                             array("L","R","N","M","B","H","F","V","W"," "))))) {
255                 $this->primary   .= 'K';
256                 $this->secondary .= 'K';
257               } else {
258                 if ($this->current > 0) {
259                   if ($this->StringAt($this->original, 0, 2, array("MC"))) {
260                     // e.g. 'McHugh'
```

Figura 20. Sección de código del algoritmo DoubleMetaphone

Levenshtein: al igual que las clases **JaroWinkler** y **DoubleMetaphone** genera una tabla temporal tratable por el analista con el fin de buscar posibles

coincidencias entre valores de tipo cadena. Se compone del método **getDistance** el cual analiza toda la tabla de datos original buscando de valores de dos en dos con el fin de encontrar la mayor cantidad posible de coincidencias. Los parámetros de entrada que recibe este método son: un listado de los atributos sobre los cuales el usuario desea aplicar la búsqueda, el nombre de la tabla de datos en la cual se encuentran estos atributos y una conexión con el Sistema Gestor de Base de Datos.

El pseudocódigo de este algoritmo se muestra en la figura 21.

```
int LevenshteinDistance(char s[1..m], char t[1..n])
   // d is a table with m+1 rows and n+1 columns
   declare int d[0..m, 0..n]

   for i from 0 to m
       d[i, 0] := i
   for j from 1 to n
       d[0, j] := j

   for i from 1 to m
       for j from 1 to n
           if s[i] = t[j] then cost := 0
                          else cost := 1
           d[i, j] := minimum(
                                d[i-1, j] + 1,     // deletion
                                d[i, j-1] + 1,     // insertion
                                d[i-1, j-1] + cost   // substitution
                            )

   return d[m, n]
```

Figura 21. Pseudocódigo algoritmo Levenshtein Distance

Un ejemplo del funcionamiento de las clases **Levenhtein, JaroWinkler y DoubleMetaphone** se muestra en la figura 22, donde los resultados pueden ser tratados mediante menús de contexto, guardados o exportados a los formatos de archivo soportados por EXDACLET

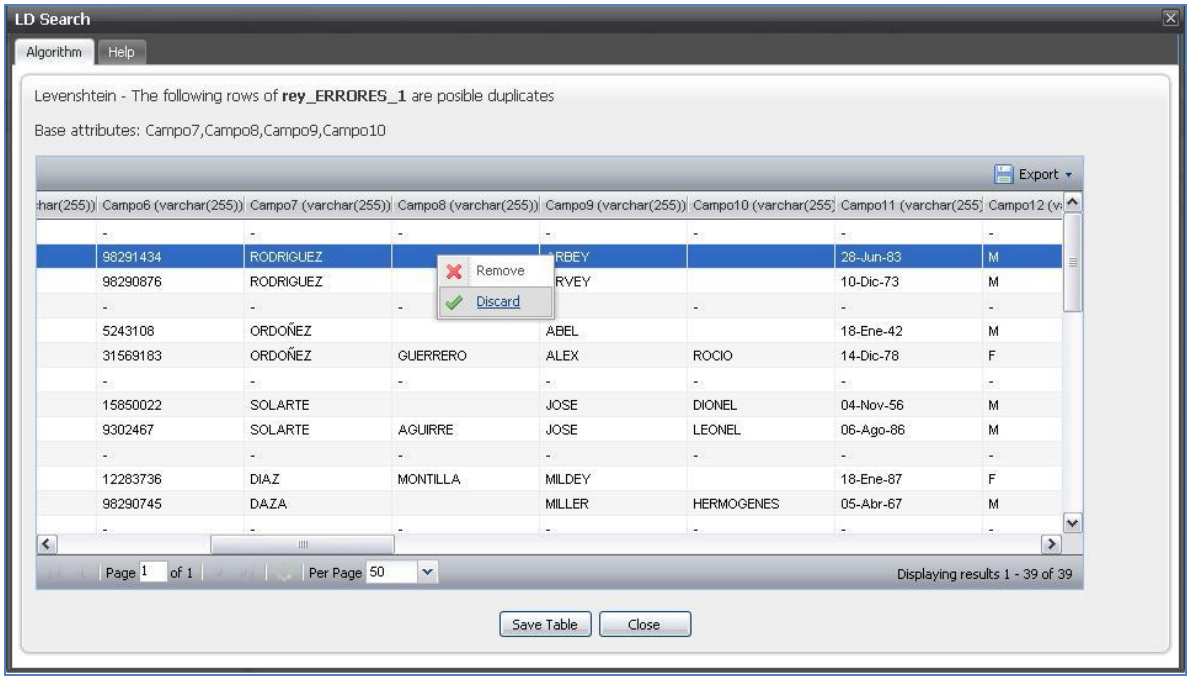


Figura 22. Tabla para tratamiento JaroWinkler, DoubleMetaphone, Levenshtein

Soundexx: Esta clase es la encargada de ejecutar el algoritmo para la búsqueda de similitud entre cadenas denominado Soundex, este procedimiento se ejecuta mediante el método **Soundex** el cual recibe como parámetros una lista con los atributos sobre los cuales se buscará posibles similitudes, el nombre de la tabla a la cual se encuentran asociados tales atributos y una conexión con el sistema gestor de base de datos. El objetivo de este método es generar una tabla de datos temporal en donde se encuentren todos los registros que según el algoritmo de Soundex hayan tenido una alta probabilidad de ser iguales, esta tabla contiene además de los atributos seleccionados por el analista, todos los demás atributos que componen a la tabla de datos original. Durante todo el proceso de ejecución de este filtro aparece una alta interacción entre la herramienta y el sistema gestor de base de datos dando pie a que el tiempo de ejecución del mismo resulte ser muy eficiente. En la figura 23 se muestra el seudocódigo de la implementación de este algoritmo.

```
S = Cadena a codificar
SIZE = 4 //Tamaño de la cadena codificada
x = explode(toUpper(S))//convertir la cadena a mayusculas
firstLetter = x[0]
// convertir letras a codigos numericos
for (int i = 0; i < len(x); i++) {
    switch (x[i]) {
        case 'B':
        case 'F':
        case 'P':
        case 'V': { x[i] = '1' break }
        case 'C':
        case 'G':
        case 'J':
        case 'K':
        case 'Q':
        case 'S':
        case 'X':
        case 'Z': { x[i] = '2' break }
        case 'D':
        case 'T': { x[i] = '3' break }
        case 'L': { x[i] = '4' break }
        case 'M':
        case 'N': { x[i] = '5' break }
        case 'R': { x[i] = '6' break }
        default:  { x[i] = '0' break }
    }
}
// remove duplicates
output = firstLetter
last = x[0]
for (i = 1; i < len(x); i++) {
    if (x[i] != '0' && x[i] != last) {
        last = x[i];
        output .= last;
    }
}
// pad with 0's or truncate
for (i = len(output); i < SIZE; i++) {
    output .= '0';
}
output = substr(output,0, SIZE);
```

Figura 23. Pseudocódigo algoritmo SOUNDEX

La tabla temporal que se genera puede ser tratada de la misma manera como se trata en los filtros tales como **Levenshtein, JaroWinkler y DoubleMetaphone** (ver figura 22).

Duplicates: Esta clase es la encargada de generar una tabla temporal en la cual se encuentran todos los registros duplicados teniendo en cuenta los atributos clave para realizar la búsqueda, esta clase se compone del método **duplicates** el cual recibe como parámetros una lista con los atributos clave de la búsqueda, es decir, con los atributos sobre los cuales se buscará una coincidencia exacta entre los valores que contengan, el nombre de la tabla de datos a la cual se encuentran asociados dichos atributos y una conexión con el Sistema Gestor de Base de

Datos. La tabla temporal que se genera a partir de esta clase puede ser tratada de la misma manera que los anteriores filtros de selección que aplican algoritmos para búsqueda de coincidencias entre cadenas.

El curso normal para la ejecución de esta clase y de igual manera para las clases **Levenshtein, JaroWinkler, DoubleMetaphone y Soundexx** comienza cuando el analista carga en la herramienta una tabla de datos sobre la cual desea aplicar el filtro, posteriormente selecciona cualquiera de los siguientes filtros de selección, **Soundex Search, Duplicates search, LD Search, Jaro-Winkler Search, DoubleMetaphone Search,** paso siguiente a esta acción el analista selecciona los atributos clave, esto se refiere a escoger sobre que atributos se realizar la búsqueda, cabe aclarar que si por ejemplo son seleccionados 4 atributos, entonces se aplicara la búsqueda que tomará a esos 4 atributos como a uno solo, es decir que, si al hacer un análisis con dos registros, 3 de los cuatro atributos seleccionados pasan dicho análisis, estos dos registros no serán insertados en la tabla temporal. Ahora bien, si son seleccionados solamente 3 atributos, entonces los dos registros anteriormente mencionados si serán insertados.

Una vez hecho el procedimiento de selección de atributos la herramienta procede a analizar la tabla de datos original y los atributos seleccionados y ejecuta el filtro mediante el método correspondiente al filtro seleccionado, si todo es correcto, se mostrará en pantalla la tabla de datos con los resultados encontrados, si no hubo resultados la tabla de datos estará vacía, de otro modo se mostrara un mensaje de error informando del tipo de error que pudo ocurrir durante el proceso.

Details: Esta clase permite generar una tabla temporal la cual contiene todos los atributos de la tabla original y adiciona uno más, que muestra información referente a la cantidad de registros con los mismos datos en todos sus atributos. Su funcionamiento se da a partir del método **tableDetails** que recibe como parámetros los atributos que se requieren mostrar y el nombre de la tabla de datos asociada a esos atributos.

Union: Esta clase es la encargada de realizar la unión de 2 tablas teniendo en cuenta los atributos que seleccione el analista y funciona gracias a los siguientes métodos.

initStartUnion: Es el encargado de generar la interfaz gráfica sobre la cual se desarrollará todo el procedimiento de unión de tablas.

*StartUnion**:*** Muestra un listado con todas las tablas de datos que tiene el usuario con el fin de que seleccione dos de ellas.

VerifyUnion: Este método permite al analista escoger los atributos de las dos tablas de datos seleccionadas, los cuales se convertirán en los atributos de la nueva tabla de datos.

DoUnion: Es el encargado de verificar la consistencia de los atributos seleccionados en las dos tablas de datos. Esto hace referencia a que verifica que la cantidad de atributos seleccionados en la primera tabla sea la misma que la de los atributos seleccionados en la segunda tabla, también de que los atributos seleccionados y que serán unidos, cumplan con el mismo tipo de datos. Posteriormente se encarga de preguntar al analista el nombre de la nueva tabla de datos que se generará a partir de las dos primeras.

ExecUnion: Es el método encargado de llevar a cabo el proceso de unión, extrayendo los valores de los atributos seleccionados en las dos tablas de datos, creando la nueva tabla de datos con la misma cantidad de atributos seleccionad y con el nombre suministrado por el analista y finalmente insertando todos los valores en la nueva tabla de datos.

1.2.5 Paquete GUI

Este paquete se conforma de una gran variedad de clases para el manejo de la interfaz de usuario las cuales permiten hacer de EXDACLET una herramienta lo suficientemente amigable para el usuario final al momento de cargar tablas de datos y aplicar sobre ellas proceso de limpieza, transformación y selección.

MainCode: En esta clase se desarrolla la interfaz principal de la herramienta, además interactúa con otras clases de la librería EXTJS mediante las cuales se

logran los efectos y se establece la distribución del espacio de trabajo de la herramienta. Su implementación se muestra en la figura 24.

Dentro de la interfaz principal se distinguen tres secciones fundamentales a saber: en la parte izquierda se encuentra el **Selector** en el cual el analista puede seleccionar y cargar una tabla de datos. En la parte central se encuentra el **Espacio De Trabajo** sobre el cual se cargará la tabla de datos seleccionada por el analista a través del **Selector**, también, en esta sección, una vez cargada una tabla de datos, aparece una pestaña que permite la consulta de la estadísticas de la tabla de datos, esto con el fin de que el analista tenga un método gráfico para observar la consistencia de los atributos que la componen. Y finalmente en la parte derecha se encuentra el **Algorithms List** que es un menú desplegable dentro del cual se listan todos los filtros implementados dentro de EXDACLET agrupados según su objetivo en Procesos de Pre-carga, Limpieza, Transformación y Selección; aparece también dentro de este menú la opción para salir de la herramienta.

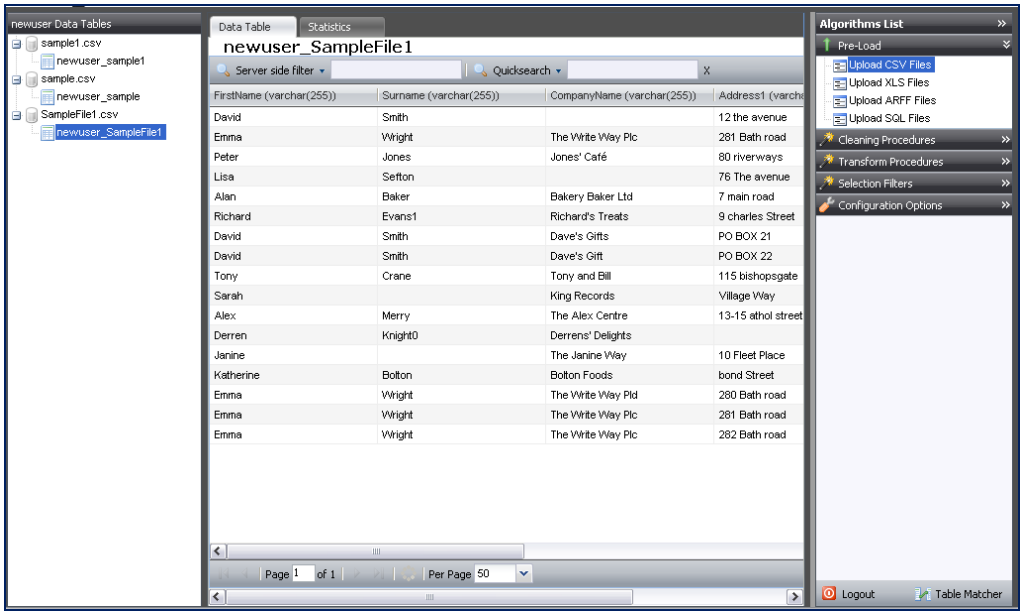


Figura 24. Clase MainCode. Interfaz Principal de la Herramienta

El método **init_app** es el encargado de iniciar y establecer instancias desde el servidor hacia el cliente mediante métodos de la clase **EXTJS** que serán utilizados

durante toda la herramienta. Dentro de este método se encuentra el llamado al sub-método **fadeOut** el cual permite que una vez iniciada la sesión para el ingreso a la herramienta, se muestre un efecto amigable de carga sobre la pantalla mientras se inician todos los componentes necesarios para la presentación de la interfaz principal. Posteriormente es llamado el método **init_container** dentro del cual se generan las tres secciones de la interfaz de la herramienta y las dos pestañas que se muestran dentro del espacio de trabajo referentes a la tabla de datos y a las estadísticas de la misma. Todo esto mediante los sub-métodos de la clase **EXTJS, Ext.BorderLayout** y **Ext.TabPanel.**

Una vez se han definido las tres secciones que componen la herramienta, entonces se procede a llenar las secciones **Selector** mediante el llamado a la clase **TreePHP y Algorithms List** mediante el método **init_accordion** de la clase **MainCode**.

- *Init_accordion():* es el encargado de generar toda la estructura de los menús desplegables y todo el contenido de los mismos. Este método interactúa con el método **Ext.ux.Accordion** de la clase **EXTJS** para generar el acordeón. Internamente llama a los métodos *init_algTree() y* fireAlg() encargados de generar los contenidos de cada sección del acordeón y de ejecutar el filtro que seleccione el analista respectivamente. En la figura 25. Se muestra el acordeón.

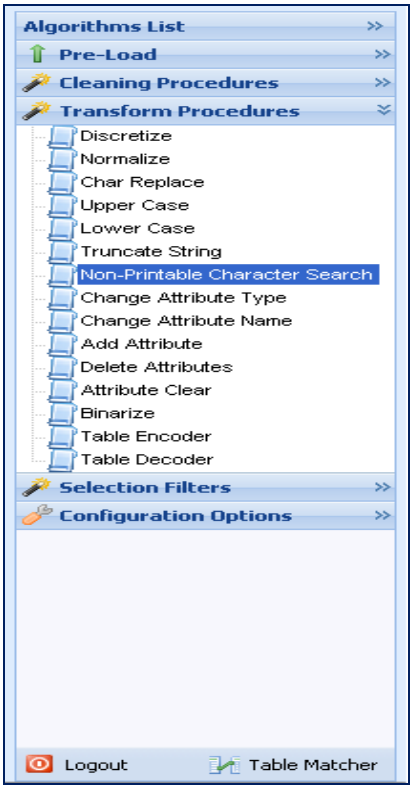


Figura 25. Acordeón

TreePHP: Es la clase encargada de generar una árbol compuesto por todos los archivos que el analista a cargado dentro de la herramienta. Cada archivo (nodo) se subdivide en hojas las cuales hacen referencia a todas las tablas de datos que se han generado tras aplicar procesos de limpieza, selección o transformación. En la figura 26 se muestra el árbol y las acciones que se pueden realizar sobre él.

Los métodos mediante los cuales se logra la generación de dicho árbol son:

- *Tree():* Genera todo el código JavaScript que será ejecutado para mostrar el árbol. Características internas dentro de este método también lo son la generación del código para el despliegue de una menú contextual para cada nodo del árbol en donde, si el nodo es hoja puede, ser eliminado siempre y cuando hayan más nodos hoja junto a el, y también puede ser exportado a cualquiera de los formatos de archivo soportados por EXDACLET. Si el nodo es un nodo padre, entonces este puede ser eliminado pero no exportado. Al

ser eliminado, se eliminan también todas las hojas asociadas a ese nodo, es decir, eliminar el archivo y todas las tablas que se encuentren asociadas a el.

- *Crear():* Es el encargado de generar toda la estructura interna del árbol, es decir, la creación de la estructura de nodos padre y nodos hijo mediante la interacción con el Sistema Gestor de Base de Datos. Este método es llamado por el método *Tree()* para completar toda la estructura del árbol.
- *Del_F_T():* Es el encargado de ejecutar los procedimientos de eliminación de hojas o nodos del árbol y al igual que el método anterior tiene una alta interacción con el Sistema Gestor de Base de Datos, de tal forma que no solo se eliminen los datos del árbol sino también físicamente de la herramienta.

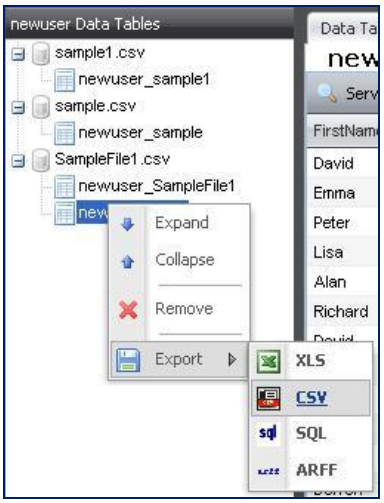


Figura 26. Sección Selector

Otra de las principales características que tiene la clase **TreePHP** es que a través de ella se cargan las tablas de datos seleccionadas por el analista al hacer doble clic sobre algún nodo hoja del árbol cargado. Cuando este evento ocurre, la sección **Espacio de Trabajo** es cargada con dos pestañas. La primera llamada **DataTable,** en donde se carga la tabla de datos asociada al nodo seleccionado por el analista a través de la clase **GRID** y la segunda, llamada **Statistics**, que contiene un resumen estadístico de la tabla de datos cargada, el cual es presentado mediante un gráfico y una tabla de ponderados, esta sección es trabajada por la clase **Statistics.**

Statistics: Esta clase es la encargada de generar toda la interfaz gráfica para mostrar las estadísticas detalladas de una tabla de datos e interactúa con las clases **Graph** para la generación de la gráfica estadística y con la clase **TableGrid** para la creación de la tabla de resumen estadístico. Esta clase se compone de los siguientes métodos:

- *Statistics():* Es el encargado de la creación del espacio de trabajo dentro del cual serán ubicados, el formulario para la selección de atributos que tendrá la gráfica estadística, la tabla de datos de resumen estadístico de los atributos seleccionados y la gráfica estadística.
- *GenGraph():* Establece una conexión con el Sistema Gestor de Base de Datos mediante una instancia de la clase **MyConn** para extraer información de los atributos seleccionados. A demás, es el encargado de la creación de la tabla de datos de resumen estadístico con la información recopilada anteriormente a través de una instancia de la clase **TableGrid.** y finalmente hace un llamado a la clase **Graph** encargada de devolver un mapa de bits que representa la gráfica estadística de la tabla de datos.

TableGrid: Es una clase que extiende de la clase **ext.grid** que a su vez se extiende de la clase **EXTJS** y su objetivo se centra en la creación de una tabla de datos de solo lectura a partir de una tabla HTML básica. Esta tabla HTML es generada a través del método **getInfo** el cual toma todo el contenido de la información extraída por el método *GenGraph()* de la clase **Statistics** y construye la estructura HTML básica requerida para que esta pueda ser interpretada

Graph: Es la clase que crea la gráfica estadística a partir de los parámetros proporcionados por el método *GenGraph()* de la clase **Statistics** y extiende de las clases **jpgraph, jpgraph_bar y jpgraph_line**. Utiliza los siguientes métodos:

- *Graph():* Para iniciar un espacio de trabajo dentro del cual se creará la gráfica y para recolectar los datos que harán parte de la gráfica.

- *SetScale():* Para definir el tamaño de las leyendas.
- *yaxis->scale->SetGrace():* Para establecer una distancia de separación entre el límite superior de la gráfica y el límite superior del espacio de trabajo.
- *xaxis->SetTickLabels():* Inserta las etiquetas o títulos de cada uno de los atributos seleccionados.
- *xaxis->SetLabelAlign():* Define la orientación de las etiquetas.
- *xaxis->SetLabelAngle():* Define el ángulo de inclinación de las etiquetas.
- *xaxis->SetLabelMargin():* Estable un margen entre las etiquetas y el límite inferior del espacio de trabajo.
- *xaxis->SetFont():* Estable el tipo de letra utilizado para las etiquetas.
- *yaxis->SetLabelMargin():*Estable un margen entre las etiquetas y el límite izquierdo del espacio de trabajo.
- *SetShadow():* Asigna una sombra a la gráfica.
- *img->SetMargin():* Define las márgenes con respecto al espacio de trabajo a partir de las cuales será dibujada la gráfica.
- *title->Set():* Establece el título de la gráfica.
- *xaxis->SetTitle():* Establece el titulo para la coordenada X.
- *yaxis->SetTitle():* Establece el titulo para la coordenada Y.
- *title->SetFont(),yaxis->title->SetFont() y xaxis->title->SetFont():* Establecen el tipo de letra para los títulos.
- *BarPlot():* Establece que el tipo de gráfica será de barras.
- *LinePlot():*Establece que el tipo de gráfica será de lineas.
- *Add():* Agrega al espacio de trabajo el tipo de gráfica seleccionado.
- *Stroke():* Dibuja la gráfica.

En las figuras 27 y .28 se muestra el espacio de trabajo con una tabla de datos y el espacio de trabajo con las estadísticas de esa misma tabla de datos.

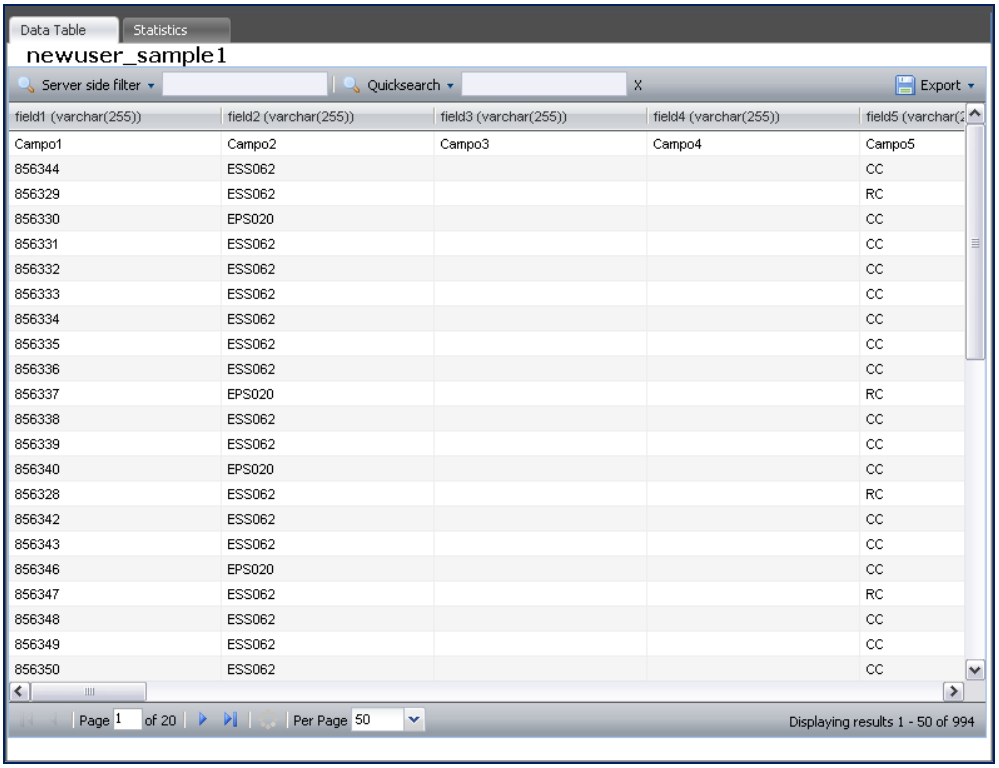


Figura 27. Espacio de trabajo con tabla de datos

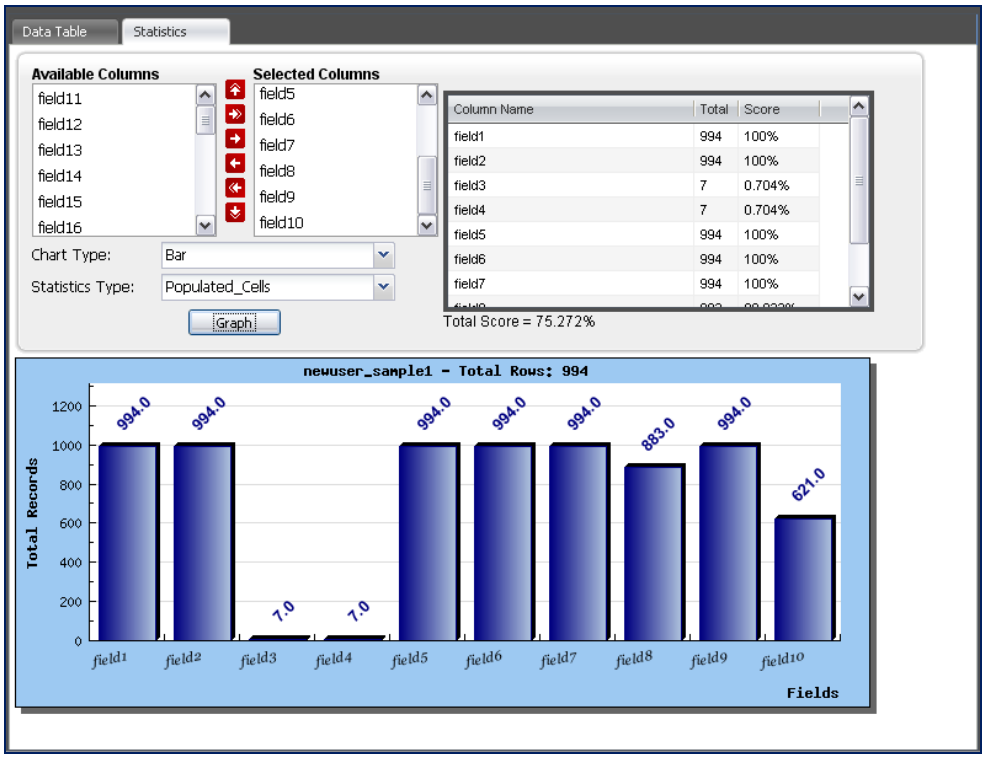


Figura 28. Espacio de trabajo con estadísticas

AlgorithmForm: Esta es una de las más fundamentales clases dentro de la interfaz gráfica de usuario de EXDACLET, debido a que por ella pasan todos y cada uno de los filtros que el analista desea aplicar sobre las tablas de datos. Se compone de los siguientes métodos.

- *__construct():* Define una instancia del método **Ext.LayoutDialog** de la clase **EXTJS** para mostrar una ventana de dialogo dentro de la cual se construirá todo el formulario. Dentro de esta ventana se generan dos pestañas para mostrar el formulario como tal y para mostrar un contenido de ayuda sobre como debe ser manipulado ese formulario.
- *resizeDialog():* Redimensiona el tamaño de la ventana de dialogo mediante el sub-método de la clase **EXTJS, setContentSize**.
- *addCombo():* Permite agregar al formulario una ComboBox y recibe como parámetros un id, la etiqueta, los valores, un mensaje de error y una descripción. Internamente utiliza los sub-métodos de la clase **EXTJS Ext.data.Record.create, Ext.data.SimpleStore y Ext.form.ComboBox** para definir el contenido y la estructura de la ComboBox.
- *setComboColumnStore():*Define un conjunto de registros y una almacenamiento para los valores que tendrá una ComboBox ubicada dentro de una columna mediante los sub-métodos de la clase **EXTJS Ext.data.Record.create, Ext.data.SimpleStore.**
- *setComboColumnCont():* Establece la estructura de la Combobox y asigna el contenido de la misma definido en el método *setComboColumnStore().*
- *setTextFieldColumn():* Permite crear una caja de texto ubicada dentro de una columna y recibe como parámetros un id, una etiqueta, una expresión regular si lo necesita para validación, un mensaje de error en caso de no cumplir con la expresión regular, una bandera para definir si se permiten campos vacios, un listado con los valores que no se permiten, un mensaje de error para esta última situación y una descripción. Internamente utiliza al sub-método de la clase **EXTJS Ext.form.TextField**.

- *setColumn():* Define una columna dentro del formulario, en ella se cargarán las cajas de texto o ComboBox definidas por los métodos *setComboColumnCont(). y setTextFieldColumn().*
- *addNumberField():* Permite agregar a un formulario una caja de texto que solo acepta valores numéricos, recibe como parámetros un id, una etiqueta, un valor a mostrar por defecto si lo tiene, una expresión regular si lo necesita para validación, un mensaje de error en caso de no cumplir con la expresión regular, una bandera para definir si se permiten campos vacios, una bandera para definir si el valor escrito en su interior aceptará o no decimales y una descripción. Internamente utiliza al sub-método de la clase **EXTJS Ext.form.NumberField** para definir la estructura de la caja de texto.
- *addTextField():* Permite agregar una caja de texto a un formulario. Recibe como parámetros los mismos valores que recibe el método *setTextFieldColum.*
- *addEditor():* Permite agregar una TextArea utilizada por el filtro **Expert Rule Editor**, internamente utiliza el sub-método de la clase **EXTJS Ext.form.TextArea**.
- *addMultiselect():* Permite agregar dos listas de selección dentro de las cuales pueden ser movidos ítems de una a otra. Revive como parámetros una etiqueta general, una etiqueta para la lista izquierda, una etiqueta para la lista derecha, los valores de la lista izquierda, los valores de la lista derecha, el ancho, el alto y una descripción. Internamente interactúa con la clase **Ext.ux.Multiselect** que a su vez se deriva de la clase **EXTJS** y utiliza el sub-método **Ext.ux.ItemSelector** para establecer la estructura y definir el contenido de las mismas.
- *addMsgDiv():* Permite agregar un espacio al final del formulario para la generación de mensajes de error que puedan ocurrir durante la interacción con el mismo.
- *setContainer():* Permite definir un espacio dentro del formulario sobre el cual se creará una cantidad determinada de Campos de texto, número, Selectbox o ComboBox.

- *ComboEvent():* Permite recargar el contenido de una ComboBox dependiendo del valor que se encuentre seleccionado en otra ComboBox mediante la manipulación de sus contenidos. Interactúa con el sub-método **findField** de la clase **EXTJS** para establecer la ComboBox base y la ComboBox destino.
- *getUploadForm():* Permite generar un formulario para la carga de archivos, incluyendo en su interior un campo de carga en el cual se mostrará la ruta del archivo a cargar y dos botones para seleccionar el archivo origen e importarlo respectivamente. Internamente interactúa con la clase **writeFile** encargada de verificar la consistencia del archivo y trasladarlo de la máquina cliente al servidor para su posterior proceso de importación a la herramienta mediante el Sistema Gestor de Base de Datos e interactúa también con los sub-métodos **Ext.form.Form, Ext.form.TextField, addButton, render, submitForm y checkMessage** de la clase **EXTJS** para validar y garantizar el correcto funcionamiento y desempeño del proceso de pre-carga de archivos.
- *getLoginForm():*Permite generar un formulario para el registro e ingreso de un usuario a la herramienta. Utiliza internamente los métodos **Ext.form.Form, Ext.form.TextField y addButton**.
- *getForm():* Este método es el encargado de generar todo el código JavaScript necesario para la construcción de un formulario. Para poder utilizar este método siempre debe haber al menos un previo llamado a cualquiera de los métodos *addNumberField(), addTextField(), addCombo() o addMultiselect().* Utiliza internamente los sub-métodos de la clase **EXTJS Ext.form.Form, addButton y render.**
- *getFormRule():* Permite construir el formulario para el método *addEditor().* Utiliza internamente los mismos métodos de la clase **EXTJS** mencionados en el método getForm().
- *getJS():* Es el método encargado de devolver el código JavaScript a ser ejecutado para construir la ventana de dialogo en la cual estará inmerso el formulario.
- *setFormWidthGen():* Permite establecer el ancho del formulario.

- *getFormStruct():* Retorna el código HTML encargado de construir la interfaz del formulario.
- *getFormStructTwo():*Retorna el código HTML encargado de armar la interfaz del formulario.
- *getHtmlHelp():* Retorna el código HTML para la definición de la estructura del contenido de ayuda del formulario.
- *setHtmlHelp():* Establece las características para la estructura del contenido de ayuda del formulario.
- *setDescription():* Extrae todas las descripciones definidas en cada elemento del formulario y las inserta dentro del contenido de ayuda.

En la figura 29 se observa un ejemplo de un formulario generado a través de esta clase.

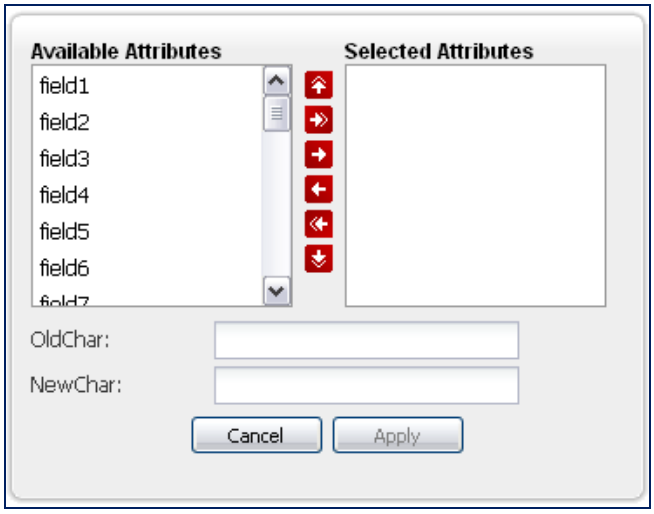


Figura 29. Formulario generado a partir de la clase AlgorithmForm

GRID: Esta clase es la encargada de generar la tabla de datos de un archivo seleccionado por el analista dentro del **Espacio de Trabajo.** Esta tabla de datos es una tabla editable y sobre ella se pueden aplicar filtros de búsqueda local y remota. Una búsqueda local se refiere a buscar valores dentro de uno o más atributos de los resultados que se estén presentando en pantalla mientras que una búsqueda remota hace referencia a buscar un valor dentro de un atributo

determinado en toda la tabla de datos. Los métodos que componen esta clase son los siguientes:

- *grid():* Este método es el encargado de generar toda la estructura de la tabla de datos que será cargada. Esta estructura viene definida por los términos **jsonreader, columnmodel y datastore** los cuales son la base para la creación de dicha tabla. Internamente crea una instancia de la clase **DATA** encargada de la construcción del código JSON que contiene todos los valores que componen la tabla de datos (filas y columnas). Además, este método interactúa con los métodos de la clase **EXTJS, Ext.menu.CheckItem**, **layout.getRegion, Ext.data.Store, Ext.data.ScriptTagProxy, Ext.data.JsonReader, Ext.grid.GridEditor, Ext.form, Ext.grid.ColumnModel, Ext.grid.EditorGrid, Ext.grid.RowSelectionModel, Ext.menu.Menu, Ext.PagingToolbar, Ext.data.SimpleStore, Ext.form.ComboBox, grid.getView, y grid.getHeaderPanel.** Todos ellos encargados de construir físicamente la tabla de datos en el cliente.
- *Grid_Update():* Este es el método encargado de actualizar la tabla de datos una vez ha sido editada, tanto para el cliente como para el servidor interactuando también con el Sistema Gestor de Base de Datos.
- *Grid_DeleteGroup():* Este método es el encargado de eliminar un conjunto de filas de la tabla de datos y al igual que el método *Grid_Update()* actualiza la tabla de datos para el cliente y para el servidor.
- *Grid_Delete() y Grid_Delete_B():* Elimina una fila de la tabla de datos y actualiza para el cliente y el servidor.

StandardGrid: Esta es la clase encargada de generar todas las tablas temporales que se utilizan en los filtros de selección y en algunos de limpieza y transformación y se compone de los siguientes métodos:

- *__construct():* inicializa una instancia para la creación de la tabla de datos temporal.

- *Define_grid():* Es el método encargado de construir toda la estructura y el contenido de la tabla de datos. Dependiendo del tipo de filtro que sea utilizado, este método incluye o excluye ciertas opciones que pueden ser adheridas a la tabla. Algunas de estas opciones son: Si los valores de todos a algunos atributos pueden ser editables o no, Si aparece dentro de un menú contextual utilizado en la tabla opciones de eliminación u omisión de registros, si se agregan opciones para el reemplazo múltiple de valores, etc. Internamente utiliza los mismos métodos de la clase **EXTJS** que se ejecutan en el método *grid()* de la clase **GRID**.
- *setForm():* Crea un formulario para realizar tares de análisis o ejecutar un procedimiento para guardar la tabla de datos temporales o simplemente cerrar la ventana de dialogo y por ende la tabla de datos generada en su interior.
- *getForm():* Retorna todo el código JavaScript generado por el método *setForm()* y lo ejecuta.
- *addReplace():* Permite agregar a la tabla de datos las opciones de reemplazo individual o múltiple de valores que se encuentran en la misma.
- *UseCorrections():* Es un método utilizado por el filtro **Email Cleaner** en el cual se actualiza un atributo la tabla de datos original en base a la tabla de datos temporal.
- *Grid_ReplaceCol():* Método utilizado para transformar una serie de valores, es ejecutado cuando en la tabla de datos temporal es agregada la opción de reemplazo múltiple.
- *Dup_tmp_tab():* Método utilizado a la hora de guardar la tabla de datos temporal. Es el encargado de generar la interfaz gráfica en la cual se solicita el nuevo nombre de la tabla que será guardada.
- *Create_dup_tmp():* es el paso siguiente a la ejecución del método *Dup_tmp_tab(),* en este método se procede a la creación de la nueva tabla de datos mediante la interacción con el Sistema Gestor de Base de Datos.

Un ejemplo de la creación de una tabla de datos a partir de la clase **StandardGrid** se muestra en la figura 22 **(**Tabla para tratamiento JaroWinkler, DoubleMetaphone, Levenshtein**).**

AlgDialog: Esta clase funciona en conjunto con la clase **StandardGrid** puesto que es la encargada de crear la ventana de dialogo y la estructura gráfica dentro de la cual será generada la tabla de datos temporal. Los métodos que la componen son los siguientes:

- *__construct():* Define la ventana de dialogo que será mostrada, interactúa con el método **Ext.LayoutDialog** de la clase **EXTJS** para lograr este objetivo.
- *getDLG():* Ejecuta el código generado en el método *__construct().*
- *getFormStruct():* Establece el espacio dentro del cual será cargada la tabla de datos temporal.
- *setHtmlHelpT():* Define la estructura del contenido de ayuda para la tabla temporal y los procesos que pueden ser realizados sobre ella.
- *getHtmlHelpT():* Ejecuta el código HTML generado por el método *setHtmlHelpT().*

formatForm: Es la clase que genera toda la estructura del formulario para los procesos de pre-carga de archivos de tipo CSV y XLS. Los métodos utilizados son:

- *__construct():* Define la ventana de dialogo sobre la cual se generará el formulario.
- *setForm():* Crea la estructura del formulario mediante código HTML.
- *getForm():* Ejecuta el código generado por setForm().
- *setRadioCsv():* Establece las opciones de radio button que se tendrán en cuenta dependiendo del tipo de formato de archivo que se este utilizando.
- *setTextQualifier():* Estable la opción para la selección del tipo de delimitador de cadena que caracteriza al archivo.

- *setDecimalSymbol():* permite crear la estructura para los delimitadores del símbolo decimal.
- *setTableResume():* Permite generar un espacio dentro del formulario para la carga de un contenido de resumen de la tabla de datos que se asocia al archivo que se este importando y al cual se le este definiendo su estructura.
- *setFieldResume():* Permite generar un espacio dentro del formulario para la carga de un contenido en el cual se muestra la estructura de los tipos de datos que se están asociando a cada atributo de la tabla de datos.
- *getJS():* Ejecuta el código generado por el método *__construct()* para la creación de la ventana de dialogo.
- *getFormStruct():*Establece el espacio dentro del cual será cargada la tabla de datos temporal.
- *setHtmlHelpF():* Define la estructura del contenido de ayuda para la tabla temporal y los procesos que pueden ser realizados sobre ella.
- *getHtmlHelpF():* Ejecuta el código HTML generado por el método *setHtmlHelpF().*

En la figura 30 se muestra un ejemplo del formulario generado mediante la clase **FormatForm**, en ella se muestran las opciones generadas y requeridas al momento de importar un archivo de tipo CSV.

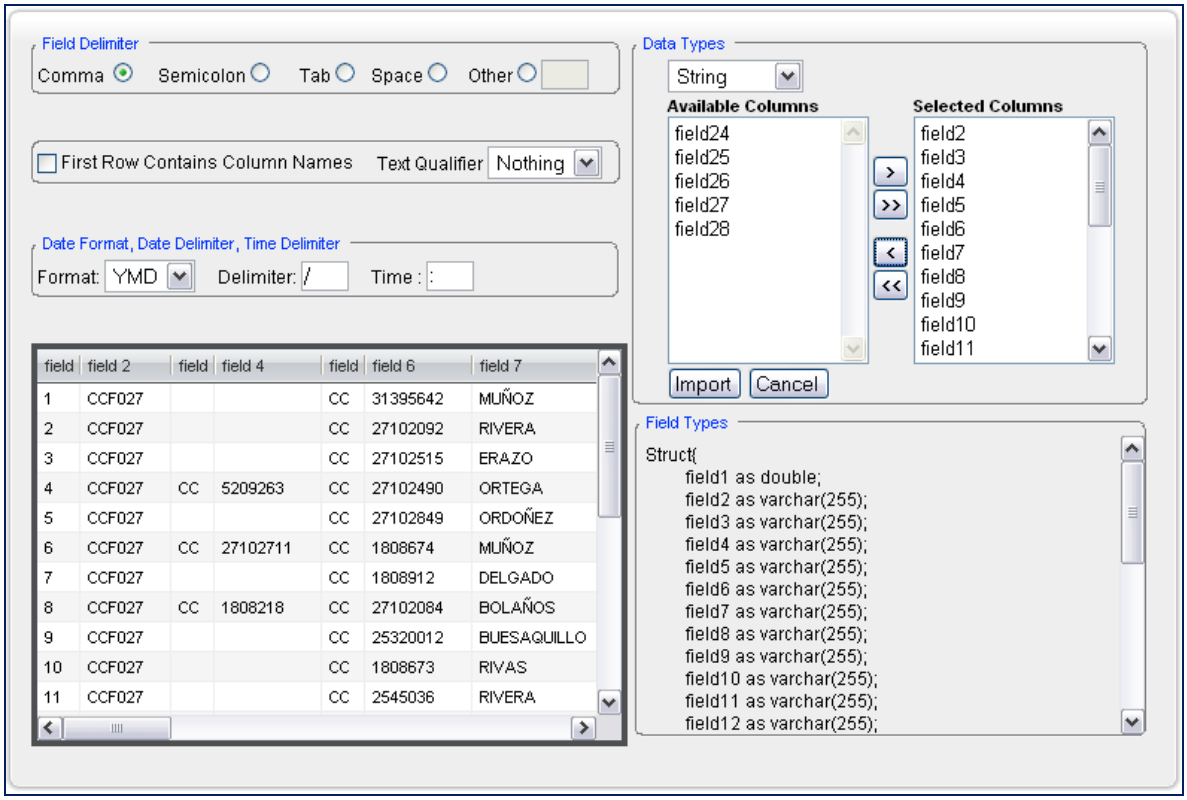


Figura 30. Ejemplo del formulario generado por la clase FormatForm

Dentro del paquete GUI cabe destacar la interacción constante con las librerías EXT a través de su clase **EXTJS** y utilizadas para la generación de las estadísticas de cada tabla de datos.

1.2.6 Paquete Connection

Este paquete es uno de los pilares fundamentales dentro de EXDACLET, debido a que absolutamente todos los filtros que reúne la herramienta y todos los procedimientos que ellos ejecutan tienen que interactuar constantemente con el Sistema Gestor de Base de Datos. Se compone de la siguiente clase con sus métodos y descripción de los mismos.

MyConn: esta clase es la encargada de establecer todas las conexiones entre la herramienta y el Sistema Gestor de Base de Datos MYSQL, es la más importante debido a que todas las acciones que realice un usuario tienen que ver con el

acceso, selección y transformación de diferentes tablas de datos que deben ser almacenadas de una forma persistente, si esta clase no cumple su objetivo toda la herramienta se vuelve obsoleta. Los métodos que llevan a cabo todos los procesos de conexión entre la herramienta y el Sistema Gestor de Base de Datos además de los procesos de ejecución de consultas para selección, modificación o eliminación de tablas y registros se muestran a continuación:

- *abrir():* Es el método encargado de establecer la conexión con el Sistema Gestor de Base de Datos mediante parámetros como el **host, nombre de usuario, contraseña y base de datos** destino de la conexión.
- *Cerrar():* Es el método encargado de dar por terminada la conexión con el Sistema Gestor de Base de Datos. Este método debe ser llamado tantas veces como sea llamado el método abrir(). Después de haberse ejecutad las consultas necesarias.
- *Query():* Es el método que ejecuta la consulta que le es pasada como parámetro, devuelve los resultados que genere la misma. Como exigencia se encuentra que debe estar establecida una conexión con el Sistema Gestor de Base de Datos.
- *Fetch():* Es el método encargado de extraer una a una las filas resultado de la consulta hecha mediante el método Query() y al igual que el, es necesario tener establecida una conexión con el Sistema Gestor de Base de Datos.
- *TotalRows():* Este método devuelve el número de filas resultantes al ser aplicada una consulta mediante el método Query(). Requieres tener establecida una conexión con el Sistema Gestor de Base de Datos.

En conjunto con los anteriores paquetes mencionados, cabe destacar la existencia de un paquete el cual no se encuentra directamente asociado con la herramienta pero que es el encargado de la administración de usuarios que pueden acceder a la misma. Este paquete es llamado **UserAdmin** y se compone de las siguientes clases:

UserTree: Esta clase es la encargada de generar el árbol mediante el cual se muestran todos los usuarios registrados en la herramienta en conjunto con todos los archivos que cada uno tiene cargado y las tablas asociadas a cada archivo. La principal característica que tiene esta clase es la generación de un menú contextual en el cual se pueden eliminar tablas, archivos o usuarios según sea la necesidad del administrador de la herramienta. Esta clase se compone de los siguientes métodos:

- *userTree():* Dentro del cual se crea toda la estructura y se inserta el contenido del árbol que será cargado.
- *Crear():* Encargado de generar todo el contenido del árbol que será insertado por el método *userTree().*
- *Del_F_T():* Es el encargado de actualizar el árbol cuando un nodo u hoja del mismo han sido eliminados e interactúa con el Sistema Gestor de Base de Datos para ejecutar los procedimientos requeridos al ser eliminada una tabla, un archivo o un usuario.
- *Usermodtree():* Este método se encarga de crear el árbol utilizado al momento de modificar usuarios.

Users: Esta clase es la encargada de generar todas las interfaces de usuario y procedimientos requeridos para la administración básica de usuarios. Se compone de los siguientes métodos con sus descripciones:

- *App():* Crea un efecto de carga antes de ingresar a la interfaz grafica de administración.
- *usersInit():* Genera y ejecuta el código HTML para crear la distribución física de las opciones del proceso de administración.
- *initAddUser():* Define el espacio dentro del cual será cargado el formulario para la creación de un usuario.
- *createAccount():* Es el método encargado de ejecutar la consulta para la creación de un usuario que pueda ingresar a trabajar con la herramienta. Esto

se logra mediante la interacción con el Sistema Gestor de Base de Datos y la clase **BD**.

- *initModUser():* Este método permite la generación del espacio en el cual se cargará el formulario para la modificación de usuarios.
- *modifyAccount():*Es el método encargado de ejecutar la consulta para la actualización de los datos de un usuario que por alguna razón olvido su contraseña o nombre de usuario. Esto se logra mediante la interacción con el Sistema Gestor de Base de Datos y la clase **BD**.

UserForm: Esta es la clase que permite la creación de los formularios para la inserción y modificación de usuarios.

- *__construct():* Inicializa los valores para la creación del formulario.
- *addUserForm():* Genera todo el código JavaScript a ser ejecutado que contiene todo el formulario para la creación de usuarios.
- *modUserForm():* Genera todo el código JavaScript a ser ejecutado que contiene todo el formulario para la modificación y/o actualización de usuarios.
- *setFormWidthGen():* Estable el ancho que tendrá el formulario generado.
- *getFormStruct():*Retorna el código HTML encargado de construir la interfaz del formulario.

En las figuras 31 y 32 se muestran los formularios generados para la adición y modificación de usuarios respectivamente.

Figura 31. Formulario para la adición de usuarios

Figura 32. Formulario para la modificación de usuarios