**Universidad Central "Marta Abreu" de Las Villas.**

**Facultad Matemática, Física y Computación**

**Licenciatura en Ciencia de la Computación**


# Trabajo de Diploma

# LPT-SQL v 1.4:

# Herramienta para insertar y modificar automáticamente reglas de negocio en bases de datos relacionales.

**AUTOR:** Ariel Alba Viego.

**TUTORES:** M.SC. Martha Beatriz Boggiano Castillo.

Lic. Ariel Calderón Solís.

Dictamen

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

**Firma del autor**

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

**Firma del tutor** **Firma del jefe del Seminario**

# Pensamiento

… creo bastante en la suerte.

Y he constatado que, cuanto más duro trabaje, más suerte tengo.

Thomas Jefferson.

# Dedicatoria

A mis abuelos,

A mis padres,

A mis tíos y tías,

A mi hermano,

A mis primos y primas,

Y a todos mis amigos.

# Agradecimientos

A mis tutores Martha Beatriz y Ariel, por su constante apoyo en este trabajo.

A mis amigos y compañeros de grupo.

A todos los profesores que a lo largo de estos años han influido en mi formación y madurez profesional.

A todos los que de una forma u otra contribuyeron a la culminación de este trabajo.

# Resumen

La automatización de las reglas de negocio en los Sistemas de información es un aspecto a considerar en el enfoque de reglas de negocio. En este sentido la herramienta LPT-SQL v1.3 traduce las reglas de negocios del Lenguaje de Patrones Técnico (LPT) a recursos de bases de datos activas; dicha versión satisface las expectativas de la generación automática del conjunto de categorías de reglas desde la perspectiva de los datos, pero no abarca la modificación de las reglas de negocio activas y su posible repercusión en los datos.

Con este trabajo se amplían las funcionalidades de LPT-SQL v 1.3, se utilizan y redefinen las modificaciones válidas en los patrones de reglas, para obtener la herramienta LPT-SQL v1.4 que permite modificar reglas activas dentro de bases de datos relacionales, sin tener en cuenta el historial del cambio. Cuando una regla es modificada, a partir de ese momento cualquier operación sobre los datos implicará la evaluación con la regla ya modificada. La nueva versión (LPT-SQL, 1.4); fue validada con pruebas de clases de equivalencias para encontrar defectos; que fueron enmendados para los casos detectados.

Así se cuenta con la opción de modificar las reglas de negocios pertenecientes al conjunto de categorías desde la perspectiva de los datos implementadas sobre una base de datos relacional.

# Abstract

Business rules automatization in Information Systems is an aspect to consider in the business rules approach. In this sense the LPT-SQL v1.3 tool translates Language Technical Patterns (LPT) business rules to active databases resources; such version meets the expectations of the automatic generation of the set of rules categories from the perspective of data, but it does not include the modification of active business rules and their potential impact on the data.

In this paper LPTSQL v1.3 functionalities are expanded, the valid modifications in the rule patterns are used and redefined, to obtain the LPT-SQL v1.4 tool that allows modifying active rules within relational databases, without taking into account the history of the change. When a rule is modified, from that moment on any operation on the data will imply the evaluation with the rule already modified. The new (LPT-SQL v 1.4) version; was validated with tests of equivalence classes to find defects; that were amended in the detected cases.

So, it counts on the option to modify the business rules belonging to the set of categories from the perspective of the data implemented on a relational database.

# Tabla de Contenidos

# Ilustraciones

# Tablas

# Introducción

En un Sistema de información (SI) la identificación de las reglas de negocio (RN) y su captación son procesos que están ligados al negocio en sí, asimismo la modificación de una regla proviene de modificar una política de negocio.

"Una regla de negocio es una regla que está bajo la jurisdicción de algún negocio."(Ross, 2009). Por tanto las reglas de negocio (RN) pueden ser activadas, modificadas o desactivadas atendiendo a las necesidades del negocio en un momento dado. Todas aquellas RN que no puedan controlarse desde el ámbito del negocio no podrán ser consideradas como tales.

Las RN reflejan la forma en que las empresas hacen los negocios. Al mismo tiempo actúan como un componente clave en el ciclo de vida de un sistema de información (SI). Si las reglas cambian el SI debe transformarse de alguna manera a través de una mayor independencia entre la inserción de las implementaciones de las reglas y los programadores de los SI. De modo que un cambio en las reglas no implica contratar servicios de mantenimiento de los sistemas(Boggiano Castillo et al., 2007).

Con la automatización de las RN no sólo se puede mejorar la toma de decisiones, sino que también brinda a las organizaciones mayor agilidad en la creación y cambios de la lógica del negocio, la cual se encuentra en constante evolución(Cientec, 2011).

El costo de la adaptación de sistemas de información tradicionales para varios cambios es muy alto. La mayoría de las organizaciones modernas exigen una capacidad para hacer modificaciones, para diversos aspectos y características de los sistemas, en toda la organización tan pronto como se produzca esta necesidad. Lograr esto es a menudo imposible debido a problemas de identificación y modificación de partes de los SI que son responsables de las normas requeridas(Halle Von, 2001).

En el laboratorio de Bases de Datos (BD) del Centro de Estudios Informáticos (CEI) se desarrolla una línea de investigación sobre la generación automática de RN en BD relacionales iniciada con (Boggiano Castillo et al., 2007), y liderada por dicha profesora.

Como resultado se ha obtenido un editor LPT-SQL que permite insertar automáticamente las reglas de negocio de las categorías con perspectiva de los datos, llamadas en Ríos Méndez, cercanas a los datos, para esto se utilizan un repositorio para almacenarlas y un compilador para generarlas.

Con las investigaciones "Aplicación de Reglas de Negocio" (Pérez Alonso, 2008b) y "Solución al problema de la cardinalidad en la generación automática de reglas de negocio" (Pereira Toledo, 2009) se logra una herramientas capaz de atender reglas de restricciones además de una independencia de la BD al recoger información necesaria para generar las RN desde el catálogo a través de procedimientos almacenados(Díaz De la Paz, 2011).

Con el trabajo de diploma "Modificación de las reglas de negocio tipo restricción y su implementación" (Marrero Lorenzo, 2009) se analizan las consecuencias de modificar RN tipo restricción en los datos de una BD relacional. En esta investigación se exponen algunas opciones sobre qué hacer con los datos que violan una regla modificada, pero no se analiza en profundidad que sucede con las reglas, ni se explican variantes que soporten el mantenimiento de reglas, ni qué cambios hay que hacer en los recursos de implementación para lograrlo(Díaz De la Paz, 2011).

En el trabajo de diploma "Traductor LPT-SQL para reglas de negocio en bases de datos relacionales" (Calderón Solís, 2011), se describe la sintáctica y semántica del lenguaje LPT y se mejora con respecto a trabajos anteriores la estructura del repositorio. Luego con "Extensión del Traductor LPT-SQL" (Pérez Pedraza, 2012) y el más reciente trabajo "LPT-SQL v1.3: Herramienta para la generación automática de reglas de negocio en bases de datos relacionales" (Ríos Méndez, 2013) se logra un software robusto y confiable, aunque estos no resuelven el problema de la modificación de reglas analizado en trabajos anteriores.

## Planteamiento del problema

Las reglas de negocio (RN) no solo deben ser funcionales a la hora de su creación sino también para su modificación (Díaz De la Paz, 2011), estas pueden cambiar de forma que sea necesario actualizarlas, lo que llevaría a modificaciones o eliminaciones en cualquiera de los datos involucrados, muchos de estos pueden entrar en contradicción con la nueva regla que se imponga.

Resulta imprescindible a partir de la última versión del software LPT-SQL descrito en (Ríos Méndez, 2013) teniendo en cuenta la sintáctica y semántica del lenguaje LPT y la estructura del repositorio, contar con una herramienta que pueda llevar a cabo la modificación de reglas dentro de bases de datos relacionales con la cual se lograría tener sistemas flexibles automáticamente cambios en las reglas de negocio.

## Objetivo General

Desarrollar una nueva versión de la herramienta LPT-SQL, basada en la extensión de sus funcionalidades que permita la modificación de reglas de negocio de las categorías desde la perspectiva de datos, haciendo uso de los recursos de bases de datos relacionales y aplicando una estrategia de prueba.

## Objetivos Específicos

1. Analizar las modificaciones válidas para cada tipo de regla del conjunto de categorías desde la perspectiva de los datos.
2. Rediseñar la herramienta para incorporar la funcionalidad de modificación de las reglas en la variante de modificación sin recordar el cambio.
3. Implementar la modificación de reglas basado en el nuevo diseño obtenido.
4. Aplicar una estrategia de prueba basado en técnicas de caja negra.

## Justificación de la Investigación

Teniendo en cuenta una de las limitaciones de la herramienta, que no incluye la modificación de las reglas precedentes sobre la generación de RN en BD relacionales, en el presente trabajo se pretende la extensión del LPT-SQL con la integración de una nueva funcionalidad que elimine esta limitante. Obtener un software más completo para la administración de las reglas de negocio desde la perspectiva de los datos, permitirá valorar el aporte de esta investigación en su justa medida.

## Preguntas de investigación

1. Qué modificaciones serán factibles considerar como válidas para cada categoría de regla?

2. ¿Cómo rediseñar la herramienta para incorporar la modificación de reglas y obtener una documentación completa?
3. ¿Cómo implementar en la herramienta la posibilidad de modificar las reglas de cada categoría?
4. ¿Qué estrategia de prueba aplicar para probar las funcionalidades de la herramienta?

# CAPÍTULO 1: Generalidades sobre reglas de negocio y fundamentos de la herramienta LPT-SQL.

En este capítulo se exponen aspectos generales sobre el enfoque de reglas de negocio, centrando la atención en la clasificación de reglas "con perspectiva de los datos". Se realiza un análisis de la definición formal del LPT, se estudian diferentes alternativas para la implementación de las RN y se profundiza en la modificación de reglas especificando los aspectos y conceptos fundamentales que servirán como base teórica para la investigación.

## 1.1 Definición de regla de negocio

Una regla de negocio es una "frase compacta sobre algún aspecto del negocio que puede expresarse en términos directamente relacionados con el negocio, utilizando un lenguaje simple y no ambiguo, accesible para todas las partes interesadas: desde el propietario del negocio hasta el arquitecto del software, pasando por el analista de negocio"(Morgan, 2002).

De acuerdo a Ronald G. Ross en (Ross, 2003) y (Ross, 2010) el cual se apoya en otros autores como son (Bajec et al., 2000) y (Morgan, 2002) se plantea:

"Una regla de negocio es:

- ✓ Una sentencia que define o restringe algunos aspectos del negocio.
- ✓ Establece restricciones a la estructura del negocio, controlando o influyendo en el comportamiento del mismo.
- ✓ No podrá ser fraccionada o descompuesta en reglas de negocio más detalladas.
- ✓ En caso de ser reducida perdería información importante sobre el negocio."

En (Morgan, 2002) se describe un conjunto de características universales que deben cumplir las declaraciones de reglas:

- ✓ Atómica: No puede ser dividida sin que se pierda información.
- ✓ Inequívoca: Tienen solamente una interpretación obvia.
- ✓ Compacta: Típicamente, una frase corta.

- ✓ Consistente: Juntas, ellas proporcionan una única y coherente descripción.
- ✓ Compatible: Usan las mismas condiciones en el resto del modelo de negocio.

### 1.1.1 Beneficios de usar reglas de negocio

Varios son los beneficios que pueden derivarse del uso de Reglas de Negocio. De todos, según Lowenthal los tres más importantes son(Lowenthal, 2005):

- ✓ **Agilidad**: respuesta simple y rápida a los requisitos dinámicos.
- ✓ **Reducción del Costo**: bajo costo para crear o actualizar las partes de aplicaciones que implementan las políticas del negocio.
- ✓ **Transparencia**: las reglas permiten fácilmente la auditoría que los servicios de software llevan a cabo en sus políticas de negocios correspondientes.

## 1.2 Formas de expresión de las Reglas de Negocio.

Según (Von Halle, 2002) son cuatro las formas de expresar las reglas de negocio, cada una para una audiencia diferente:

- ✓ Conversación informal del negocio
- ✓ Versión en lenguaje natural
- ✓ Versión en lenguaje de especificación de reglas
- ✓ Versión en lenguaje de implementación de reglas

Una regla comienza su vida en la conversación informal de las personas del negocio, luego se realiza una versión más disciplinada en lenguaje natural cuya audiencia es la comunidad del negocio. El lenguaje natural presenta algunas insuficiencias, tales como falta de precisión y redundancia, por lo que se requiere que se expresen las reglas en un lenguaje que ya reúne todas las cualidades requeridas. Tal lenguaje es el de especificación de reglas. Es declarativo y disciplinado, y está dirigido tanto al personal del negocio como al técnico. Pero este lenguaje solo nos dice qué acometerse según la regla, no cómo. Por ello las reglas se mudan del lenguaje de especificación al lenguaje de implementación, el cual tiene todo el potencial para ser ejecutado.

En (Morgan, 2002), se distinguen solo tres formas de expresión de reglas de negocio:

- ✓ Informal: representación en lenguaje natural dentro de una rango limitado de patrones.
- ✓ Técnico: combina referencias a datos estructurados, operadores y lenguaje natural controlado.
- ✓ Formal: proporciona sentencias conforme a una sintaxis definida con propiedades matemáticas particulares.

Ha de notarse algunas similitudes entre estos dos criterios expuestos. Aunque la investigación desarrollada por (Pérez Alonso, 2008) utiliza la clasificación de Tony Morgan, se ha creído conveniente utilizar la primera clasificación. Es mucho más clara, según criterio propio, con respecto a los estados de desarrollo de un sistema para el manejo de reglas de negocio.

## 1.3 Clasificaciones de reglas de negocio

De acuerdo a la diversidad y complejidad de las reglas de negocio, los autores tienden a agruparlas y clasificarlas siguiendo diferentes puntos de vista. En el análisis de las diferentes clasificaciones se revela similitud entre algunas de estas y la complementación de unas y otras (Perez Alonso, 2010). En el marco de este tema se han identificado diversas clasificaciones, entre las que podemos citar las de Solivares, Lowenthal y Morgan abordadas en (Pérez Alonso, 2008a), las de Weiden, Ashwell y Solivares tratadas en (Perez Alonso, 2010) y las de Ross, Ioana Matei y Coti Colop estudiadas en (Pérez Pedraza, 2012).

Por el pensamiento de Tony Morgan (Morgan, 2002), la forma más conveniente de crear sentencias de regla es seleccionar patrones adecuados desde una pequeña lista disponible. Por ejemplo: <sentencia>:= <sujeto> debe <restricción>.

En este trabajo se utilizan las categorías de reglas de negocio desde la perspectiva de los datos, surgidas como resultado de la investigación "Generación automática de reglas de negocio en un contexto relacional" encabezada por la profesora Martha Beatriz Boggiano (Boggiano Castillo et al., 2012), (Pérez Alonso, 2010), (Calderón Solís, 2011). Estas categorías de reglas se consideran en función de los datos del negocio, que en última instancia se almacenan en las bases de datos de los SI que ayudan al control de los mismos.

## 1.4. Descripción de los patrones de reglas

Algunos autores identifican los tipos de reglas con un patrón que facilita la escritura de las reglas ((Morgan, 2002), (Perkins, 2002), (Lowenthal, 2005)).

Para cada categoría de las reglas desde la perspectiva de los datos, se le asocia un patrón que la identifica. A continuación se describen los patrones de reglas con perspectiva de los datos. Los elementos variables que conforman los patrones se describen más adelante en la Tabla 1.1

Restricción

- ✓ <determinante> <sujeto> (no puede tener <características>) | (puede tener <características> solo si <hechos>).

Cómputo

- ✓ <determinante><resultado> es calculado como <expresión matemática>.
- ✓ <determinante> <resultado> en <sujeto> es calculado como <expresión matemática>.
- ✓ <determinante><resultado> en <sujeto> para <atributo> es calculado como <expresión matemática>.

Clasificación

- ✓ <determinante> <sujeto> [no] es definido como <clasificación> [( si | a menos que )<característica>]

Notificación

- ✓ Notificar <mensaje> si <hecho>.

| Elemento | Significado |
|---|---|
| <determinante> | Es el determinante para cada sujeto, por ejemplo: Una, Uno, El, La, Cada, Todos. Según el mejor sentido en la redacción de la regla. |
| <sujeto> | Es un término u objeto del negocio, tipo de entidad. |
| <hechos> | Son hechos relativos al estado o comportamiento del negocio, |

| | |
|---|---|
| | incluyendo o no al sujeto. |
| <características> | Describe las características del sujeto en el negocio, tanto internas como relacionadas con otras entidades. Pueden incluir hechos con el fin de caracterizar al sujeto. |
| <resultado> | Es cualquier valor numérico que tiene significado en el negocio y resulta de evaluar una expresión matemática. |
| <expresión matemática> | Es una expresión matemática que se define sobre combinaciones de términos del negocio junto con funciones y operadores disponibles, para obtener un resultado. |
| <atributo> | Es un atributo de una entidad del negocio. Un resultado puede destinarse para un atributo. |
| <clasificación> | Definición de un término del negocio, que se refiere a un subconjunto de instancias de un tipo de entidad del negocio que cumple determinadas características. |
| <mensaje> | Mensaje de información sobre estado de alerta del negocio. |

Tabla 1.1 Elementos variables de los patrones

### 1.4.1 Operadores lógicos, aritméticos y de comparación del LPT.

A continuación la Tabla 1.2 muestra los operadores que ofrece LPT para facilitar la construcción de sentencias complejas (Calderón Solís, 2011).

| Tipo | Operadores |
|---|---|
| Lógicos | OR, AND, NOT, XOR |
| Aritméticos | +, -, *, / |
| Comparación | <, >, <=, >=, =, <> |

Tabla 1.2 Operadores del LPT.

Como se explicó con anterioridad, al utilizar la notación punto es posible la obtención de elementos múltiples. Para la manipulación de colecciones de elementos LPT brinda un grupo de operadores de conjunto que se muestran en la Tabla 1.3. El resultado de aplicar uno de los operadores de conjunto a una colección de elementos es un elemento individual.

| Operadores | Significado |
|---|---|
| SIZEOF<colección> | Retorna cuántos elementos contiene la colección de elementos. |
| EMPTY<colección> | Retorna verdadero si la colección no contiene elementos. |
| EXISTS(<elemento>, <colección>) | Retorna verdadero si un elemento existe en una colección. |
| AVG<colección> | Retorna el promedio de una colección numérica. |
| SUM<colección> | Retorna la suma de una colección numérica. |
| MÍN<colección> | Retorna el mínimo de una colección numérica. |
| MÁX<colección> | Retorna el elemento máximo de la colección numérica |
| AVGDIF<colección> | Retorna el promedio de los elementos diferentes de una colección numérica. |

Tabla 1.3 Operadores de conjunto del LPT

## 1.5 Lenguaje de Patrones Técnicos

Los patrones para escribir las reglas de negocio inicialmente en u nivel informal o lenguaje natural estructurado surgen por la necesidad de captar las reglas en un lenguaje natural y de forma organizada y sus diferentes tipos cubren los requerimientos del negocio. Pero estos patrones pierden sentido si no pueden ser expresados técnicamente para una futura implementación (Perez Alonso, 2010).

Se evidencia la importancia de contar con un lenguaje, capaz de asimilar una regla casi natural en un lenguaje formal, más matemático y sencillo de implementar. Esta

idea es confirmada por varios autores(Heidenreich et al., 2005, Demuth, 2005, Zimbrão et al., 2002).

Para la escritura de las reglas en lenguaje técnico aparecen en la literatura algunos lenguajes OCL como reconoce (Calderón Solís, 2011) , los basados en XML ((Pérez Pedraza, 2012), (Pereira Toledo, 2009)) ; sin embargo, se hizo necesario una manera de expresar las reglas de negocio en función de las tablas, sus atributos, y las interrelaciones para BD relacionales basado en los patrones de reglas pertenecientes a la clasificación denominada "desde una perspectiva de datos", de una manera sencilla, por lo que se reconoce la necesidad de crear un lenguaje basado en notación punto que facilita escribir los elementos de los patrones usando los nombres de tablas interrelacionadas, separadas por punto (.), así como los atributos correspondientes a tablas.

En este lenguaje técnico que es simple, su funcionalidad se resume a: "expresar en nivel técnico tipos de patrones de RN, que serán convertidos a recursos en lenguaje SQL para BD" (Calderón Solís, 2011).

### 1.5.1 Análisis de la notación punto y la navegación.

En LPT la notación punto es el estilo que se usa, para establecer el medio de acceso a los atributos de tablas y posibilita la navegación. Esta notación le brinda consistencia al lenguaje (Calderón Solís, 2011).

Acceso simple a un atributo:

Tabla **1** [. Atributo]

Camino de navegación entre tablas:

Tabla **1.** Tabla **2**. (…) . Tabla **N** [. Atributo]

Es necesario destacar el carácter opcional de terminar o no en un atributo. En el primer caso se estaría haciendo referencia al atributo especificado perteneciente a las instancias resultantes, mientras que en el segundo caso al atributo(s) identificador(es). También es posible utilizar la palabra reservada sujeto, la cual hace referencia a la tabla o clasificación correspondiente al sujeto de la regla(Ríos Méndez, 2013).

Por ejemplo, si se desea acceder al carnet de identidad (atributo identificador) de un estudiante se puede obtener indistintamente.

“Estudiante.CI”

“Estudiante”

En (Calderón Solís, 2011) se extienden las posibilidades de expresión del camino de navegación de manera que hace posible restringir el resultado a partir del valor de algún atributo perteneciente a cualquier tabla que pertenezca al camino. Es permisible la utilización de un camino de navegación para restringir el valor de un atributo.

## 1.6 Implementación de Reglas de Negocio.

Según (Morgan, 2002) existen formas muy disímiles de implementar las reglas, incluso pueden existir varias técnicas para implementar una misma regla. Al considerar cada una de las alternativas para determinar cuál funciona mejor en una determinada situación, se debe tener en cuenta:

- ✓ La viabilidad a largo plazo de su estrategia.
- ✓ Rendimiento en tiempo de ejecución.
- ✓ El grado de cumplimiento de la regla.
- ✓ Flexibilidad.
- ✓ La capacidad para mantener las operaciones del negocio.

A continuación se muestran algunas de las técnicas a utilizar:

### 1.6.1 Lenguajes de Programación

La incorporación de reglas en el código de programa es probablemente la más utilizada. Es posible mediante sentencias de un lenguaje de programación implementar RN. Un requisito indispensable para esta alternativa es seleccionar entre todas las ramas del código, una alternativa basada en la condición dada. Esto es bastante fácil de satisfacer en cualquier lenguaje de programación, aunque los detalles pueden variar de uno a otro (Morgan, 2002).

Ejemplo JAVA:

```
//RN #1.
public boolean RN#1(Estudiante est){
        if(est.edad >35)
                //Código si se satisface la condición.
        else
                // Código si no se satisface la condición.
 }
```

### 1.6.2 Scripts

Un paso de avance en la capacidad de gestionar las reglas es separarlas en scripts. Independientemente de la elección del lenguaje la esencia del script es añadir un comportamiento variable en una estructura relativamente fija. Esta alternativa puede ser bastante útil para reglas que pueden cambiar en el tiempo con relativa facilidad. Un cambio en la regla solo implicaría la modificación correspondiente en el script sin afectación en el código (Morgan, 2002).

### 1.6.3 Motor de reglas

Un motor de reglas, o servidor lógico, es un componente diseñado con el único propósito de gestionar reglas. A diferencia de un componente general, un motor de reglas no se construye para resolver cualquier problema particular sino que brinda un conjunto genérico de capacidades para definir, almacenar y aplicar reglas.

Estos servidores, además de manejar la lógica del negocio proporcionan una serie de funciones útiles, facilitando el trabajo del desarrollador, proporcionando una manera conjunta de trabajo para las aplicaciones. Las funciones típicas son:

- ✓ Características de elasticidad para aumentar la disponibilidad del sistema.
- ✓ Almacenamiento en caché de datos para reducir los tiempos de respuesta.
- ✓ Las características adicionales de seguridad para proteger los sistemas centrales en un entorno global de usuarios.
- ✓ Diversas técnicas, tales como el uso común de recursos, para aumentar la escalabilidad de la aplicación.

- ✓ Conectores listos para interactuar con orígenes de datos específicos.
- ✓ Funciones transaccionales y de bloqueo de registro

Los aspectos que aspectos que caracterizan a los motores de reglas son:

- **Expresiones de las reglas**: El lenguaje que se presenta como principal candidato para este tipo de reglas es Prolog. El uso del motor de reglas ya incluye el lenguaje que se usa para expresar las reglas y generalmente una interfaz interactiva para su construcción.
- **Mecanismos de inferencia.**

Una regla es un pequeño fragmento de conocimiento del negocio. Una colección de estos fragmentos trabaja de manera coordinada para lograr los objetivos generales del negocio. Las reglas pueden agruparse en conjuntos de reglas para centrarse en la búsqueda de una solución a un problema específico. Un motor de reglas proporciona un entorno cómodo, donde pueden compartir un ambiente y un medio de expresión común. También comparten el mecanismo de inferencia, ya sea guiado por datos o por objetivos.

- **Despliegue.**

Existen varias opciones de tecnologías para la construcción de motor de reglas dentro de un sistema operacional. La manera más obvia puede ser poner toda la maquinaria de regla en un solo módulo, de modo que todas las partes puedan cooperar eficazmente. En la práctica, este enfoque simplista no funciona muy bien. Por lo que se utilizan una serie de estrategias ingeniosas para abordar estos problemas potenciales. Estas estrategias incluyen tanto la clonación hacia varios servidores como brindar el motor de reglas como un conjunto de funciones pertenecientes a una biblioteca (Morgan, 2002).

En este trabajo se toma como punto de partida una herramienta, LPT-SQL v1.3, que utiliza los mecanismos de BD para implementar la regla, pero con la característica más moderna de generarlos automáticamente, manteniendo un repositorio independiente al sistema las reglas de negocio; este modo de trabajo se identifica en ese sentido con el enfoque de reglas de negocio.

Los cambios en el entorno empresarial de una organización casi nunca suceden de manera espontánea, sin ningún tipo de la razón, estos suelen ser impulsados tanto desde las decisiones internas de la gestión de la organización o de fuerzas externas, como las leyes y regulaciones del gobierno. Estos cambios casi siempre llevan a la adaptación de los procesos de negocio existentes y con frecuencia requieren reflejarlo en los sistemas nuevos o modificados(Bajec et al., 2005).

Generalmente los cambios en los procesos de negocio y en los sistemas de apoyo están en las RN y sus implementaciones, que son revisados y modificados de acuerdo a los nuevos objetivos, metas y políticas. Esto requiere que los cambios sean coordinados a nivel de empresa. Una RN en particular puede estar involucrada en los procesos de negocio de varios sistemas y apoyada por un conjunto de subsistemas. Además, en un subsistema particular, cada Estado podrá llevarse a cabo una serie de formas diferentes (por ejemplo, como una BD con disparadores, procedimientos almacenados, etc.) Con el fin de ser capaz de mantener el apoyo a los sistemas en consonancia con los requerimientos del negocio, debe tenerse documentación de cómo las RN evolucionan a partir de su origen en el entorno empresarial para su aplicación en el SI(Bajec et al., 2005).

La herramienta LPT-SQL v1.3 genera las implementaciones de las reglas de negocio desde la perspectiva de los datos, en forma de recursos activos de BD relaciones, Para las regla de restricción y notificación se generan vistas y disparadores dentro; para las de cómputo primera variante y clasificación, funciones, las reglas de cómputo en su segunda y tercera variante se implementa además usando vistas y disparadores. Esta versión de la herramienta no permite implementar las modificaciones de las reglas, lo cual se considera crucial y constituye la base para desarrollar este trabajo,

## 1.7 Arquitectura general para implementar automáticamente reglas de negocio en bases de datos.

En la Figura 1.1 se muestra el esquema general del proceso de generación de las reglas. La entrada del traductor es la regla en LPT que se extrae del repositorio de reglas Durante el proceso de traducción se consulta la información del catálogo que está almacenada en el repositorio del catálogo. Este repositorio contiene la

información de las tablas, atributos, triggers, vistas y funciones que están implementadas en la base de datos física. Posteriormente se genera la información necesaria para generar la regla que es almacenada en el repositorio de generación.

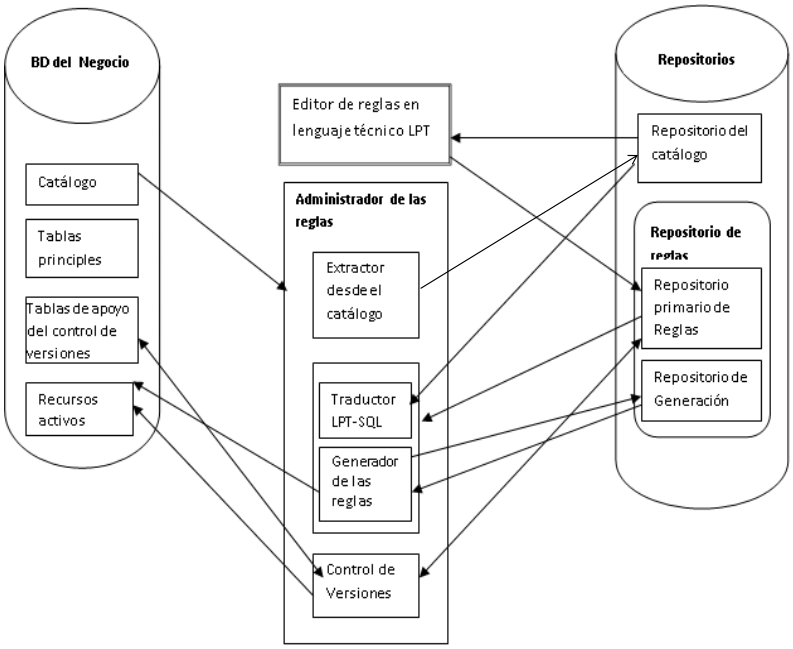



Figura 1.1 Arquitectura de la administración de reglas de negocio en BD relacionales. Fuente (Boggiano Castillo, 2014)

Para los tipos de reglas, sea de tipo Restricción, Clasificación, Computo o Notificación, esta información consiste en la consulta SQL base y la lista de eventos, los eventos que posibilitan un cambio de estado relacionado con cierta regla. Vale destacar que dicha información es suficiente para generar la regla de acuerdo al tipo con el que se esté tratando y desde cualquiera de los dos modos analizados: mediato o inmediato en el caso de Restricción. Finalmente se genera la regla a partir de la información extraída del repositorio de generación de acuerdo al enfoque que se maneje, y se actualiza el repositorio de reglas obteniéndose la representación de la regla en lenguaje natural, técnico y formal. La regla es implementada en la base de datos física a partir de la representación formal en forma de recursos.

## 1.8 Gestión de reglas de negocio

Las RN tienen un efecto significativo en la escalabilidad de la aplicación del negocio. Mientras se define el ambiente del negocio se encuentra mucha dificultad para gestionar y mantener las RN si no están debidamente tratadas, especialmente si están escondidas en el código del programa(Marrero Lorenzo, 2009).

Como plantea (Bajec et al., 2000), se tienen las siguientes consideraciones. Los problemas más comunes que se derivan son:

- ✓ Cada cambio de las RN requiere programación.
- ✓ Las RN son distribuidas a través de una aplicación lógica; así el lugar donde hay que hacer el cambio es difícil de encontrar.
- ✓ Las RN son un grupo lógico dependientes e interrelacionadas. Por tanto ellas deben ser modificadas cuidadosamente considerando los posibles efectos en otras reglas.
- ✓ Los requerimientos para el cambio primario se alejan de las necesidades del negocio con las cuales los desarrolladores no están familiarizados, y corren el riesgo de que las RN no sean entendidas e implementadas correctamente.
- ✓ Es muy difícil controlar las RN una vez que no existe un patrón único y común para ello.

Un método que permite adicionar flexibilidad y adaptabilidad a una aplicación es adicionar parámetros a la aplicación y sus componentes. Estos parámetros pueden darse en un conjunto de ficheros o en una BD y pueden ser administrados a través de un utilitario de configuración. De modo que la aplicación pueda ser adaptada a diferentes ambientes y situaciones sin ningún esfuerzo de programación. Se ha demostrado que esta técnica es aplicable al manejo de las RN. Aunque las políticas del negocio se quedan escondidas en la lógica de la aplicación, pueden ser modificadas a través de parámetros sin ninguna necesidad de cambiar el código del programa. La adición de las modificaciones a las RN pueden ser hechas por personas del negocio, si existen herramientas de configuración simples y de fácil manejo(Bajec et al., 2000).

La versión 1.3 de LPT-SQL así como las versiones anteriores, no cuentan con la opción de modificar las reglas activas.

### 1.8.1 Repositorio de reglas: componente esencial en la Administración de las reglas de negocio.

Hay varias reglas de negocio basado en modelos de la arquitectura (Halle Von, 2001), pero el papel de repositorio de reglas sigue siendo el mismo. Su objetivo es

contribuir a una mayor gestión de reglas fáciles. Tradicionalmente, las reglas se aplican con un enfoque orientado a procesos, donde las reglas se encuentran principalmente en el código del programa. Esto se traduce en una flexibilidad muy limitada. La aplicación de reglas de negocio en forma de manuales y listas de verificación ofrece más flexibilidad, pero no proporciona el control sobre las acciones reales de la SI de usuario.

Físicamente, el repositorio de reglas es una colección de reglas de negocio autónomo, que puede ser modificado en cualquier momento utilizando las herramientas relativamente fáciles. Estos son dos soluciones para el almacenamiento de las reglas:

1. Enfoque basado en el parámetro. En este caso las reglas se almacenan en la base de datos en los que se caracterizan por los valores de varios atributos. Se ha demostrado por diferentes investigadores, que repositorio de reglas puede ser diseñado como una base de datos independientes (Plotkin, 1999) o como parte del modelo lógico principales (Perkins, 2002). Sin embargo, la primera solución ofrece más flexibilidad y más opciones para almacenar las reglas de negocio complicado.

2. Enfoque independiente basada en procesos. Este enfoque es similar a las tradicionales metodologías que las reglas se aplican directamente en el código del programa, sólo en este caso el código, lo que representa las reglas, se almacena de forma independiente de otras capas del SI y por lo tanto las reglas se expresan sólo una vez en el sistema.

La aplicación de todo el sistema de las reglas almacenadas es administrada por el mecanismo de regla de interpretación especial denominado Motor de Reglas de Negocio.

El repositorio de reglas que se tendrá en cuenta para este trabajo se basa en el diseño de repositorio que aparece en el Anexo 4, donde se destaca la estructura en dos secciones: repositorio primario de reglas y repositorio de generación (Calderón Solís, 2011).

Cada regla es representada de acuerdo a los tres niveles de expresión utilizados: informal, técnico y formal. La etiqueta PeriodosActivación contiene el conjunto de PeríodoActivación que representa un período de tiempo en que la regla está activa y

se describe con los atributos id (identificador), fecha_inicio y fecha_ fin. La abstracción del repositorio se concibe como un árbol.

## 1.9 Control de cambios y versiones de reglas de negocio

Según Bajec (2005), además de las actividades que se realizan durante el modelado de la empresa (ME) y el desarrollo de SI, el escenario manejador de RN prescribe tareas adicionales que se ocupan de los cambios de RN a través de su ciclo de vida. Estas actividades se pueden realizar a nivel de empresa o SI. Ellos incluyen:

- ✓ **Control de cambios**: El propósito de esta actividad es coordinar los cambios de RN. En general, el motivo de cambios en las RN siempre se plantea desde el entorno de negocio de la empresa. Si parece que una RN ha cambiado debido a algún problema técnico o a causa de algún nuevo requisito del SI, y desde el punto de vista comercial no es necesario para el cambio, entonces esto no es realmente una RN. Las RN son propiedad de los negocios y siempre están estrechamente relacionadas con el entorno empresarial. En consecuencia, por cada cambio en el modelo de RN debe haber una explicación a nivel empresarial, describiendo por qué el cambio es necesario. Además, para ser capaz de controlar los cambios, la información tiene que ser dirigida sobre quién ha solicitado los cambios, quién los ha aprobado y cuando serán implementados(Díaz De la Paz, 2011).
- ✓ **El control de versiones:** Debido a su naturaleza dinámica, las RN pueden tener varias versiones sobre el tiempo. En algunos casos, incluso pueden existir varias versiones de la misma RN en uso del SI de la empresa. Por ejemplo, una versión de la regla puede ser utilizada en un subsistema, mientras que otros subsistemas están utilizando una versión diferente. A fin de conocer las versiones que se utilizan en un determinado sistema, este último debe ser capaz de realizar evaluaciones de diferentes versiones de regla y darle seguimiento a la historia de las RN(Díaz De la Paz, 2011).
- ✓ **Control de impacto:** Las RN rara vez son independientes, lo que significa que un cambio en una regla en particular pueden causar cambios en otras reglas. Para gestionar los cambios, todas las dependencias entre las reglas y otros componentes se debe conocer las interdependencias, o sea darle

seguimiento. Antes que una RN sea cambiada, se debe hacer un análisis de impacto para averiguar si existe algún obstáculo en el cambio de la regla(Díaz De la Paz, 2011).

Los negocios se encuentran en constante evolución, por lo que las políticas prohibitorias sobre las que se basan también están propensas al cambio. Esto provoca que sea esencial investigar sobre las versiones de reglas en el enfoque de RN(Díaz De la Paz, 2011). Del estudio realizado sobre este tema la Dra. Silvie Spreeuwenberg tiene experiencia con la modelación de RN, lo cual ha sustentado el desarrollo de herramientas y técnicas para aumentar la calidad de las mismas. Silvie expresa, "Creemos que uno debe concentrarse en la calidad de la gestión de reglas de negocio para lograr el propósito del enfoque de reglas de negocio" (Spreeuwenberg, 2008, Spreeuwenberg, 2007a, Spreeuwenberg, 2007b) . Esta autora ha investigado sobre las versiones e historial de las reglas, lo cual resulta de interés en el presente trabajo.

En (Spreeuwenberg, 2008, Spreeuwenberg, 2007a, Spreeuwenberg, 2007b), se propone resolver el problema de versiones de reglas de una manera declarativa, partiendo de que todas las reglas están sujetas a ser versionadas y como condición extra se tiene que pueden estar activas en un período de tiempo determinado. Utiliza un atributo de marca de tiempo (markdate) con la fecha de comienzo y fecha fin en que la regla estuvo activa.

En (Spreeuwenberg, 2007a) se muestran los siguientes enunciados que describen el problema de versiones de reglas:

- ✓ Las reglas son sentencias declarativas que son usadas en una o más tareas para determinar un cierto valor para un atributo.
- ✓ Algunas reglas solo pueden ser aplicadas en un período de tiempo determinado. Este período puede ser descrito con fecha de inicio y fecha fin. La fecha fin no es obligatoria. Las reglas que solo tienen fecha de inicio son aplicables por lo menos hasta la fecha actual.
- ✓ Las reglas aplicables sobre cierta fecha deben ser consistentes.
- ✓ Un caso del pasado puede ser retomado aplicando las reglas que estaban activas en ese período.

- ✓ La fecha que se usa para recuperar las reglas correctas pueden ser diferentes de acuerdo con:
    - La tarea que se va a llevar a cabo.
    - El caso (situación) sobre el cual las reglas se aplicarán.
    - El evento que produjo la activación de las reglas.

En (Spreeuwenberg, 2007b) en aras de la gestión de reglas estas se agrupan en el conjunto de reglas RulesetPostRules, que pueden ser activadas en el mismo período de tiempo. Utiliza algoritmos de encadenamiento hacia adelante y hacia atrás; relacionados con los motores de reglas que tienen un mecanismo de inferencia. La desventaja de esta solución está en el mantenimiento del conjunto de regla. El cual contendrá una buena cantidad de redundancia y reglas cuando hay poca superposición en el período de aplicabilidad de las reglas.

En (Spreeuwenberg, 2008) se discute una estrategia para tratar con versiones de reglas, la cual trabaja si hay lotes de diferentes versiones de nuevas reglas. Para versionar reglas complejas recomienda ver las reglas como un objeto con atributos (meta información) y crear un objeto para cada regla. Esto permite especificar una información extra acerca de la regla en su modelo de objeto.

Entre los principales atributos se encuentran:

- ✓ StartDate fecha a partir de la cual la regla es aplicable.
- ✓ EndDate fecha hasta la cual la regla es aplicable.
- ✓ PostRule atributo booleano que indica si la regla esta activa o no.
- ✓ RuleStatement sentencia o referencia de la regla actual.

## 1.10 Análisis sobre la modificación de las reglas de negocio de restricción.

Para realizar un análisis exhaustivo de las posibles transformaciones a realizar en una regla de negocio, se asume la representación de estas en lenguaje técnico usando la notación Punto(Pérez Alonso, 2008b).

Según (Marrero Lorenzo, 2009) Cuando una política del negocio se modifica o cambia, la regla asociada también debe modificarse. Para lograr la modificación de una regla, se tienen en cuenta ciertas consideraciones muy importantes:

- ✓ El identificador de la regla no puede ser modificado.
- ✓ El sujeto de la regla no se modifica

Estas limitaciones de no permitirse la modificación del sujeto y el identificador están dados porque, en caso de permitirlo se estaría refiriendo a una nueva regla y no a una modificación. EL identificador de la regla y el sujeto, identifican a la regla que se desea modificar(Marrero Lorenzo, 2009).

Cuando se aplica una regla de negocio, es porque se necesita que los datos cumplan dicha regla. Con el tiempo las restricciones asociadas al negocio pueden cambiar; y por tanto esto lleva a modificar una regla que está siendo utilizada por la base de datos.

La modificación de una regla, puede o no entrar en contradicción con alguna de la información contenida en la base de datos. Es necesario analizar, qué hacer con los datos que cumplen la restricción original de la regla que se propone transformar; pero que entran en contradicción con la modificación que se quiere implantar (Marrero Lorenzo, 2009).

Debido a su naturaleza dinámica, las reglas de negocio pueden cambiar con el tiempo (Bajec and Krisper, 2001) , ante las modificaciones de las reglas se admiten dos maneras de considerar estas modificaciones:

1. Cambiar la regla sin guardar la versión anterior.
2. Cambiar la regla guardando las versiones de la regla, en el tiempo.

Cuando una regla se modifica y pertenece a una de las categorías estudiadas en el presente trabajo, los datos que se inserten o modifiquen pueden ser tratados con la nueva regla a partir del momento del cambio. La problemática se reduce a sustituir la regla anterior por la actual, y administrar los datos nuevos y los antiguos con la nueva versión de la regla ante los eventos sobre la base de datos.

En determinados casos, puede ser necesario tratar los datos con las versiones de las reglas de negocio que estaban activas cuando estos fueron creados en la base de

datos y para esto es necesario trabajar con las versiones e historial de las reglas de negocio.

### 1.10.1 Versiones de reglas de negocios.

Guardar la versión de cada regla, permite conocer la historia de las versiones: el camino de derivación. Un esquema de estas derivaciones se propone en (García Pérez, 1997) , representada en forma de árbol, como se muestra en la Figura 1.2. Los nodos del árbol identifican las versiones y los arcos representan las interrelaciones de derivación entre las versiones de reglas.

Se propone crear un árbol cuya raíz es un nodo de nivel cero denominado RN (0), así todas las reglas se derivan de este nodo. Se establece el valor cero para el atributo versión del nodo raíz; este valor nunca cambia.

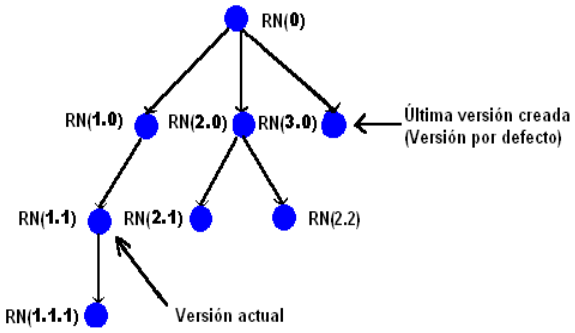



Figura 1.2 Historia de versiones de reglas de negocio.

Los nodos que se encuentran en el nivel uno corresponden a las reglas que se crean antes de insertar datos en la BD, son las primeras versiones de las reglas, los que se encuentran en el nivel dos corresponden a las distintas versiones (los cambios) de las reglas del nivel uno, se tienen tantos niveles como existan versiones de una misma regla, de esta manera se define la relación padre-hijo. Se tienen tantas versiones de reglas como se necesite en el negocio.

Al insertar una regla nueva, su atributo padre_versión es igual a cero pero si esta regla se modifica, el nodo hoja tomará en padre_versión el valor del atributo versión, de la regla que esta se derivó.

### 1.10.2 Operaciones sobre las reglas de negocio

Las reglas, almacenadas en un repositorio, pueden ser insertadas, actualizadas y eliminadas. Las reglas deben ser implementadas por primera vez cuando pasan a estado activo; cuando se inserten datos, estos se van a regir por las reglas activas en el momento de creación.

### 1.10.3 Mantenimiento de las reglas de negocio de restricción y su correspondencia con los datos.

La concepción de implementar reglas de negocio requiere también del análisis acerca de la modificación de estas en el tiempo. Cuando una regla es modificada, usando la variante de guardar las versiones de las reglas, y considerando la necesidad de conservar la coherencia de los datos con las mismas reglas utilizadas para su creación , se necesita conocer cuáles tuplas de qué tablas (representantes de entidades del negocio) fueron insertadas con cuál versión de qué regla.

Se analizan diferentes variantes para realizar el mantenimiento de las reglas de negocio, con el objetivo de poder aplicar las reglas vigentes cuando se insertó cada tupla, independientemente si las versiones de estas reglas están activas o no en el momento en que los datos son evaluados. Se estudian tres maneras organizadas en dos variantes.

1. Asociación datos-tiempo.
2. Asociación datos-reglas mediante:
    a. Una bitácora.
    b. Tablas de relación tupla-regla.

La variante Asociación datos-reglas mediante: Tablas de relación tupla-regla en la BD, es seleccionada porque resulta más ventajoso con respecto a la velocidad para hacer búsquedas en la base de datos.

### 1.10.4 Asociación datos-reglas mediante: Tablas de relación tupla-regla

Se propone adicionar a la base de datos la tabla Regla, y tablas de asociación tupla-regla, procedentes de interrelaciones muchos-muchos (N: M) entre una regla y las tuplas insertadas bajo su acción.

Regla constituye una tabla con los identificadores de las reglas, y sus atributos principales, las interrelaciones RelacionTablaiRegla conformarán tablas que tendrán la llave primaria compuesta por los atributos llaves de las tablas de la BD del negocio y de la tabla Regla.

Cuando se realiza una operación de inserción, eliminación o actualización de una regla se actualiza la tabla Regla, y se mantienen actualizadas las tablas de asociación tupla-regla, para cada uno de los eventos sobre las tablas implicadas.

## 1.11 Conclusiones parciales del capítulo.

En este capítulo se ha brindado un cuadro teórico sobre el enfoque de RN, haciendo énfasis en la clasificación de reglas "con perspectiva de los datos". Fueron analizadas las características del LPT. Se mostraron diferentes tendencias para la implementación de las RN y se profundizó en la modificación, tomando en cuenta las diversas variantes que existen para implementarlas.

# CAPÍTULO 2: Fundamentos teóricos para la modificación de reglas de negocio desde la perspectiva de los datos.

Los patrones de las reglas de negocio analizados en este trabajo, son denominados como cercanos a los datos en (Ríos Méndez, 2013), en el curso actual de la investigación se han denominados patrones de reglas de negocios desde la perspectiva de los datos.
Cada patrón está descrito por elementos, cuyos significados se expresan en la Tabla 1.1
En este capítulo se describen los patrones de reglas y los elementos que conforman el patrón y se analizan las modificaciones válidas en una regla de negocio activa.

## 2.1 Análisis las modificaciones válidas para las reglas de negocio de los patrones en las categorías desde la perspectiva de los datos.

Para cada categoría a continuación se describen los patrones, la cuáles elementos pueden ser o no modificados, y se presentan ejemplos de cada tipo de regla con una modificación válida.

### 2.1.1 Patrón de restricción

determinante sujeto ‘no puede tener’ características |determinante sujeto ‘puede tener’ características ‘solo si’ hechos

**determinante**::= 'El' |'La' |'Los' |'Las' | 'Un' | 'Uno' | 'Una' | 'Cada' | 'Todos';

**sujeto** ::= identificador

**características::**= exp

**hechos::= características**

Ver Anexo1 con Sintaxis del LPT.

En este patrón el identificador de la regla, el **determinante** y el **sujeto** no pueden sufrir cambios ya que una modificación de estos cambiaría el sentido de la regla y en las características no se aceptan modificaciones en el camino de navegación. A continuación se exponen las modificaciones válidas que se le pueden hacer a la regla (Díaz De la Paz, 2011).

1. En las características y en los hechos.

- ✓ Cambiar la <Funcion> por otra de las existentes.

- ✓ Cambiar el <Operador> por otro de los existentes.
- ✓ Modificar el <VALOR>.
- ✓ Cambiar el <OperLogico> por otro de los existentes.
- ✓ Adicionar restricciones de la siguiente forma (<OperLogico> [<Neg>] <Exp>).
- ✓ Eliminar una o todas las restricciones de la siguiente forma (<OperLogico> [<Neg>] <Exp>).

2. Cambiar la variante del patrón
   - ✓ Se parte de la primera variante:
     <determinante> <sujeto> puede tener <características> solo si <hechos>
     Modificación:
     - se elimina el solo si y el o los hechos.
     - se cambia el puede tener por no puede tener (así queda como la segunda variante).
   - ✓ Se parte de la segunda variante:
     <determinante> <sujeto> no puede tener <características>
     Modificación:
     - Adicionar el solo si y el o los hechos.
     - Se cambia él no puede tener por puede tener (así pasa a la primera variante).

**Ejemplo:**

Un estudiante puede tener sizeof(sujeto.Participacion_Cultura)>0 solo si sujeto.promedio>3.5

**determinante** ::= Un

**sujeto::=** estudiante

**características**::= sizeof(sujeto.Participacion_Cultura)>0

**hechos**::= sujeto.promedio>3.5

**operador_conjunto**::=sizeof

**camino_navegacion** ::= sujeto.Participacion_Cultura, sujeto.promedio

**Modificación Válida:**

Un estudiante puede tener sizeofdif(sujeto.Participacion_Cultura)>=1 solo si sujeto.promedio<>2.0

**determinante** ::= Un

**sujeto::=** estudiante

**características**::= sizeofdif(sujeto.Participacion_Cultura)>=1

**hechos**::= sujeto.promedio<>2.0

**operador_conjunto**::= sizeofdif

**camino_navegacion** ::= sujeto.Participacion_Cultura, sujeto.promedio

### 2.1.2 Patrón de Clasificación

determinante sujeto 'es definido como' clasif 'si' características | determinante sujeto 'no es definido como' clasif 'a menos que' características

**clasif** ::= identificador

**características::**= exp

En este patrón se consideran modificaciones inválidas el **determinante**, **sujeto**, **clasif** y en las características el camino de navegación. Las válidas se comportan similares al patrón de restricción.

**Ejemplo**:

Un estudiante es definido como desaprobado si sujeto.promedio<3

**determinante**::= Un

**sujeto** ::= estudiante

**clasificación**::=desaprobado

**característica** ::= sujeto.promedio<3

**Modificación Válida:**

Un estudiante es definido como desaprobado si sujeto.promedio=2

**determinante**::= Un

**sujeto** ::= estudiante

**clasificación**::=desaprobado

**característica** ::= sujeto.promedio=2

### 2.1.3 Patrón de Notificación

**notificación** ::= 'notificar' mensaje 'si' hechos

**mensaje**::= 'mensaje de información'

**hecho** ::=caracteristicas

En este Patrón el mensaje de información puede sufrir cambios. En el hecho las modificaciones válidas son similares a las modificaciones permitidas en las **características** de la regla de restricción; en ningún caso se permite hacer modificaciones a los caminos de navegación. El 'notificar' no es editable ya que es parte de la estructura que lleva la regla.

**Ejemplo:**

Notificar 'Estudiante desaprobado' si NOT (Empty(Estudiante.promedio<3))

**mensaje** ::='Estudiante desaprobado'

**hecho :**: = NOT (Empty(Estudiante.promedio<3))

**operador_conjunto** ::= Empty

**camino_navegacion** ::= Estudiante.promedio

**Modificación Válida:**

Notificar 'Estudiante no aprobado' si (Empty(Estudiante.promedio>3))

**mensaje** ::="Estudiante no aprobado '

**hecho :**: = (Empty(Estudiante.promedio>3))

**operador_conjunto** ::= Empty

**camino_navegacion** ::= Estudiante.promedio

### 2.1.4 Patrón de Cómputo

determinante resultado ‘es calculado como’ expresión_matemática| determinante resultado ‘en’ sujeto ‘es calculado como’ expresión_matemática | determinante resultado ‘en’ sujeto ‘para’ atributo ‘es calculado como’ expresión_matemática

**resultado**::=string

**expresión_matemática**::= exp_valor

**atributo**::=identificador

En este patrón el determinante, el sujeto, el resultado y el atributo no son modificables, en sus correspondientes variantes del patrón. En la **expresión_matemática** se consideran modificaciones válidas los operadores lógicos y de conjunto así como los operadores de comparación y valores e inválidas el camino de navegación.

**Ejemplos:**

Ejemplo1:

El lugar en Criollos para lugar es calculado como sujeto.edicion+100

**Determinant**e ::= El

**sujeto ::=** Criollos

**atributo ::=** lugar

**expresión_matemática** ::=sujeto.edicion+100

**Modificación Válida:**

El lugar en Criollos para lugar es calculado como sujeto.edicion

**determinant**e ::= El

**sujeto ::=** Criollos

**atributo ::=** lugar

**expresión_matemática** ::= sujeto.edicion

Ejemplo2:

El promedio en Estudiante es calculado como avg(sujeto.notas)

**determinante** ::= El

**resultado ::=** promedio

**sujeto ::=** Estudiante

**expresión_matemática ::**= avg(sujeto.notas)

**Modificación Válida:**

El promedio en Estudiante es calculado como sum(sujeto.notas)/5

**determinante** ::= El

**resultado ::=** promedio

**sujeto ::=** Estudiante

**expresión_matemática ::**= sum(sujeto.notas)/5

Ejemplo3:

El promedioDeportivo es calculado como avg(Equipo.promedio)

**determinante ::**= El

**resultado ::=** promedioDeportivo

**algoritmo ::**= avg(Equipo.promedio)

**Modificación Válida:**

El promedioDeportivo es calculado como avgdif(Equipo.promedio)+100

**determinante ::**= El

**resultado ::=** promedioDeportivo

**algoritmo ::**= avgdif(Equipo.promedio)+100

## 2.2 Elementos de programación en Java para implementar el proceso de traducción de reglas de LPT a recursos SQL.

Según (Calderón Solís 2011) en el proceso de traducción de las reglas representadas en lenguaje técnico pueden destacarse tres fases fundamentales:

1. Extracción de la regla desde el repositorio de reglas y su traducción.

2. Almacenamiento del código generado por el traductor en el repositorio de generación.

3. Generación de la regla como recursos de bases de datos y la actualización del repositorio de reglas.

### 2.2.1 Extracción de la regla desde el repositorio de reglas y su traducción.

Como antecedentes a esta primera fase, es necesario contar con un repositorio de reglas, en caso de no existir se puede crear uno nuevo, estos pasos serán explicados en el epígrafe 3.4.

El primer paso de esta fase es cargar desde el repositorio de reglas la regla escrita en LPT. Para leer y escribir en los diferentes repositorios se utiliza el paquete XML. En este paquete se ofrece la interfaz RuleRepository que permite operaciones básicas sobre las reglas almacenadas en los repositorios.

La sintaxis abstracta de un programa consiste en la estructura que poseen sus componentes teniendo en cuenta solamente los terminales que aportan algún valor semántico, es decir, se ignoran los símbolos separadores y de agrupación (tales como punto y coma, paréntesis, etc.). En esta sintaxis, a cada componente de un programa y al programa completo le corresponde una forma de representación llamada árbol de sintaxis abstracta (ASA) que expresa la estructura de ese componente, en este caso el programa es la regla.

El analizador sintáctico o parser, a partir del programa procesado, lexicológicamente (fase donde se agrupan los tokens reconocidos) y sintácticamente (fase donde se determina si la secuencia de tokens es correcta), debe generar el ASA del mismo, tal y como se muestra en la Figura 2.1:

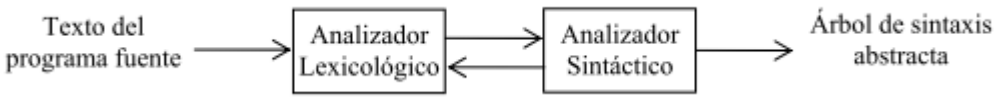



Figura 2.1 Esquema general del proceso de análisis sintáctico

### 2.2.2 Almacenamiento del código generado por el traductor en el repositorio de generación.

Obtenidos los recursos necesarios para la implementación de la regla, dígase sentencia SQL y una lista de eventos, es necesario almacenar esta información en el repositorio de generación. Para este proceso son utilizadas las siguientes clases que heredan de StoreKeeper: véase Anexo 2

- ✓ StoreKeeperRestriction
- ✓ StoreKeeperComputo
- ✓ StoreKeeperNotification
- ✓ StoreKeeperClasification

Cada una de estas tiene un atributo de tipo RuleGenerationRepositoryXml, que permite almacenar información en el repositorio de generación. Estas clases son responsables de almacenar la información necesaria para generar las reglas, redefiniendo en cada uno de los casos el método store (). Para realizar modificaciones sobre la estructura de este repositorio solo bastaría con modificar este método.

### 2.2.3 Generación de la regla como recursos de bases de datos y la actualización del repositorio de reglas.

Para la generación de una regla se crean los recursos de base de datos a partir de los datos que se inserten o modifiquen, estos pueden ser tratados con la nueva regla a partir del momento del cambio. La problemática se reduce a sustituir la regla anterior por la actual, y administrar los datos nuevos y los antiguos con la nueva versión de la regla ante los eventos sobre la base de datos

La clase Generator (véase Anexo2 ), es la responsable de generar la regla a partir de la información almacenada en el repositorio de generación. Para ello se utiliza la clase abstracta Assembler, en ella se declarar el método assembly, en el que se realiza todo el proceso de generación de los recursos que se implementaran en la base de datos, este método será implementado de manera particular en todas las clases hijos de acuerdo al gestor y al tipo. De Assembler heredan clases como AssemblerCalculus y AssemblerRestrictionImmediate del tipo restricción y cálculo respectivamente.

El código final se genera en una subclase de acuerdo al tipo de regla, enfoque y gestor para los cuales fue definida, como se observa en la Figura 2.2.

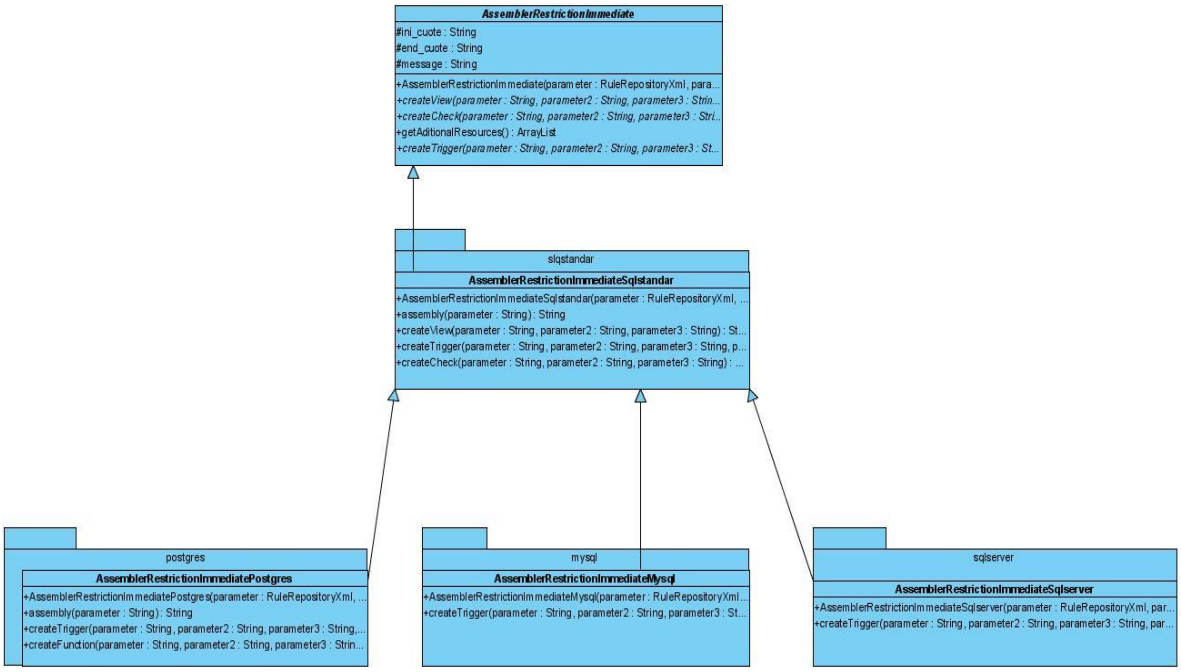



Figura 2.2 Parte del diagrama de Clases. Regla de restricción.

La clase AssemblerRestrictionImmediate es una clase abstracta, en ella se declaran métodos como createView y createTrigger que posteriormente serán implementados de acuerdo al tipo de gestor con que se trabaje, sea SqlServer o PostgresSql.

La clase AssemblerRestrictionImmediateSqlstandar hereda de la clase AssemblerRestrictionImmediate, esta contiene el método nombrado assembly, como ya se expuso responsable de la generación de los recursos en BD, estos recursos se crean de acuerdo a la declaración de la clase abstracta Slqestándar.

Posteriormente de acuerdo al gestor de BD, se llama a su clase correspondiente dentro del paquete, decir que todas estas clases heredan de AssemblerRestrictionImmediateSqlstandar, en las que se sobrescriben la implementación de los recursos de acuerdo a las exigencias de cada gestor. Todo método createTrigger, createView y createFunction si es postgres o createTrigger y createView en el caso de SqlServer devuelve un string con el código de generación, estos métodos como ya se mencionó se llaman y se utilizan desde el método assembly, el cual les pasa datos como las restricciones de la regla, el nombre de la tabla donde restringe la regla y el nombre que tomara el recurso.

Una vez almacenada toda la información necesaria en el repositorio de generación, es posible generar la regla como recursos de BD. El primer paso de este proceso es leer desde el repositorio de generación los datos generales de la regla (SGBD, identificador de la regla, etc.) y la información necesaria para su generación. El acceso a esta información es posible mediante la clase RuleGenerationRepositoryXml. Esta información es utilizada de acuerdo al tipo de regla, el enfoque y el SGBD para el cual se está generando. (Figura 2.3)

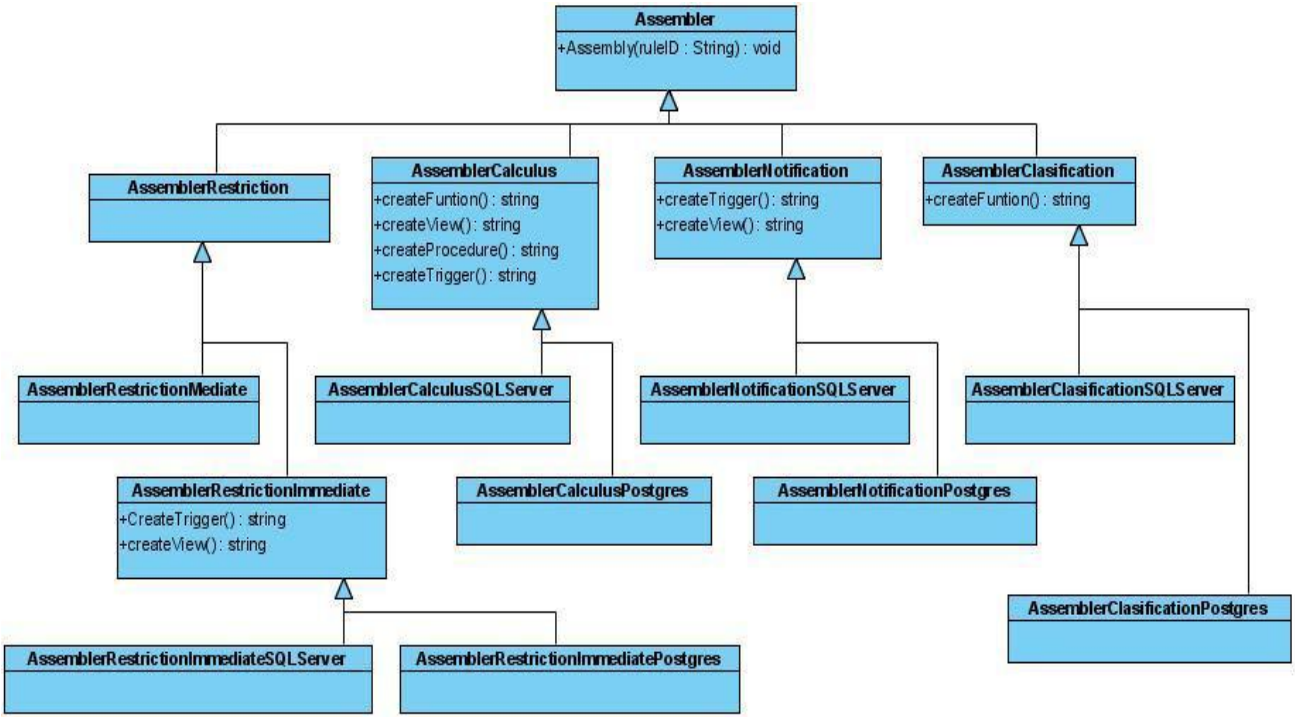



Figura 2.3 Diagrama de Clases de generación general

### 2.2.4 Métodos de Generación para la Regla de Restricción

La generación de la regla de manera automática, se realiza a través de los métodos **createTrigger** createfunction, createView y assembly de las clases AssemblerRestrictionImmediatePostgres y AssemblerRestrictionImmediateSqlserver en dependencia del tipo de gestor utilizado. A continuación se muestra el código de los métodos **createTrigger,** createfunction, createView y assembly para el caso de AssemblerRestrictionImmediatePostgres.

public String createTrigger(String triggerName, String event, String table, String funcionName) {

```
    return String.format("CREATE TRIGGER %s AFTER %s ON %s FOR EACH
ROW EXECUTE PROCEDURE %s();", triggerName, event, table, funcionName);
  }
  public String createFunction(String functionName, String statement, String[]
params, String viewName, String id) {
    String baseStatement = "CREATE OR REPLACE FUNCTION %s()
RETURNS trigger AS $$ BEGIN if(EXISTS(" + statement + ") ) then RAISE
EXCEPTION '%s';END IF;RETURN NEW; END;  $$ LANGUAGE plpgsql;";
    return String.format(baseStatement, functionName, viewName, message);
  }
  public String createView(String nombre, String sujeto, String exp) {
    return String.format("CREATE VIEW %s AS SELECT * FROM %s sujeto
WHERE ( %s ) ", nombre, sujeto, exp);
  }
```

Véase Anexo 3 con los métodos generación de la Regla en el caso de Sql Server.

Para lograr la adecuada implementación de la regla, de modo que para una operación sobre los datos se chequee el cumplimiento de la regla solamente sobre el datos afectado, se modifica el método createFunction (de la clase AssemblerRestrictionImmediatePostgres en el caso de BD en postgres.), la modificación consiste en tratar con una variable statement que contiene los valores llaves de las filas involucradas en la operación (sobre los datos) afectada por una regla modificada.

A continuación se muestra un fragmento del método assembly, de la clase en el que se utilizan los métodos descritos anteriormente:

```
public String assembly(String id) {
    …
Iterator<Element> it = listaEventos.iterator();
Iterator<Element> eit = temp.getChild("TipoEvento").getChildren().iterator();
```

```
Element function = new Element("Recurso");
function.setAttribute("tipo", "FUNCTION");
function.setAttribute("nombre", functionName + "()");
recursos.addContent(function);
sqlCode=createFunction(functionName,
temp.getChild("CaminoAlSujeto").getText(), null, viewName, id);
function.setText(sqlCode);
…
while (eit.hasNext()) {
    Element curr = eit.next();
    String event = curr.getText();//evento

    String triggerName = String.format("T%s%s_%d", event.substring(0,
2).trim(), id, counter);
    triggerName = triggerName.toLowerCase();
    Element trigger = new Element("Recurso");
    trigger.setAttribute("tipo", "TRIGGER");
    trigger.setAttribute("nombre", triggerName);
    trigger.setAttribute("tabla", table);
    recursos.addContent(trigger);
    sqlCode = createTrigger(triggerName, event, table, functionName);
    trigger.setText(sqlCode);
    formal += sqlCode + sep;
}
```

**Ejemplo de la generación de Regla tipo Restricción**

**Lenguaje Natural:**

Un estudiante puede participar en el festival solo si su promedio es mayor que 3.5

**Lenguaje Técnico:**

Un estudiante puede tener sizeof(sujeto.Participacion_Cultura)>0 solo si sujeto.promedio>3.5

**Compilación:**

((SELECT COUNT ( b."id_estudiante" ) FROM public."estudiante" a , public."participacion_cultura" b WHERE (sujeto."id_estudiante"=a."id_estudiante") AND (a."id_estudiante"=b."id_estudiante")) > 0) AND NOT ((SELECT a."promedio" FROM public."estudiante" a WHERE (sujeto."id_estudiante"=a."id_estudiante")) > 3.5)

**Generación:**

CREATE OR REPLACE FUNCTION public.**frn3_1**() RETURNS trigger AS $$ BEGIN if(EXISTS(SELECT * FROM public."**vrn3**" a WHERE (a."id_estudiante"=NEW."id_estudiante"))) then RAISE EXCEPTION 'Un estudiante puede participar en el festival solo si su promedio es mayor que 3.5 ';END IF;RETURN NEW; END; $$ LANGUAGE plpgsql;

;

CREATE TRIGGER **turn3_1** AFTER UPDATE ON public."estudiante" FOR EACH ROW EXECUTE PROCEDURE public.frn3_1();

;

CREATE TRIGGER **tirn3_1** AFTER INSERT ON public."estudiante" FOR EACH ROW EXECUTE PROCEDURE public.frn3_1();

;

CREATE OR REPLACE FUNCTION public.**frn3_2**() RETURNS trigger AS $$ BEGIN if(EXISTS(SELECT * FROM public."vrn3" a, public."participacion_cultura" b WHERE (a."id_estudiante"=b."id_estudiante") AND (b."id_estudiante"=NEW."id_estudiante") AND (b."festival_numero"=NEW."festival_numero")) ) then RAISE EXCEPTION 'Un

estudiante puede participar en el festival solo si su promedio es mayor que 3.5 ';END IF;RETURN NEW; END;  $$ LANGUAGE plpgsql;

;

CREATE TRIGGER **turn3_2** AFTER UPDATE ON public."participacion_cultura" FOR EACH ROW EXECUTE PROCEDURE public.frn3_2 ();;

CREATE TRIGGER **tirn3_2** AFTER INSERT ON public."participacion_cultura" FOR EACH ROW EXECUTE PROCEDURE public.frn3_2 ();

;

CREATE VIEW public."**vrn3**" AS SELECT * FROM public."estudiante" sujeto WHERE ( ((SELECT COUNT( b."id_estudiante") FROM public."estudiante" a , public."participacion_cultura" b WHERE (sujeto."id_estudiante"=a."id_estudiante") AND (a."id_estudiante"=b."id_estudiante")) > 0) AND NOT ((SELECT a."promedio" FROM public."estudiante" a WHERE (sujeto."id_estudiante"=a."id_estudiante")) > 3.5))

**;**

Cada trigger tirn# o turn# corresponde a las operaciones de inserción o actualización respectivamente dentro de la BD y en el símbolo # la versión a la que se refiere por si existe más de un disparador del mismo tipo de los dos descritos.

## 2.3 Dependencias entre las Reglas

Una posible interdependencia entre reglas de negocio de estas categorías, es una regla de restricción dependiente de reglas de cálculo o clasificación. Para implementar una dependencia entre estas reglas se crea una función para una regla de cálculo o de clasificación, que es utilizada por la regla de restricción que la incorpora en su escritura en LPT. A continuación se muestra un ejemplo:

**Regla 1:**

Lenguaje Técnico:

Un estudiante es definido como desaprobado si sujeto.promedio<3

**Regla 2:**

Un estudiante no puede tener sizeof(sujeto.estudiante_investigacion)=1 and desaprobado(sujeto)

La regla 2 utiliza la función desaprobado que implementa la regla 1.

Este tipo de problemas se tratan dentro de la herramienta, primero se verifica a la hora de activar la regla si esta presenta alguna dependencia dentro de las activas. El siguiente fragmento de código recoge dentro de una lista las dependencias de la regla actual que se está modificando:

```
RuleXml re = r.getRule((String) rules.getValue());
   ArrayList dependencies = (ArrayList) gen.getDependenciesTo(re.getId());
   StringBuilder verboseList = new StringBuilder();
   if(!dependencies.isEmpty ())
      {
         Iterator<String> it = dependencies.iterator();
         while (it.hasNext()) {
               String theRule = it.next();
               RuleXml theRuleXml = r.getRule(theRule);
               if (theRuleXml.getStatus().equalsIgnoreCase("activa")) {
                  verboseList.append(theRule);
                  reglasdepend.add(theRule);
                  if (it.hasNext()) {
                  verboseList.append(",");
                  }
           }
      }
```

En la lista reglasdepend y en verboseList de tipo StringBuilder quedan registradas las reglas correspondientes, se utiliza la lista ya que esta se hace atributo de la clase y

permite que otros métodos como el guardar la modificación puedan saber si de la regla actual dependen otras.

En el caso que existan reglas, el usuario puede escoger si desactivar las reglas momentáneamente mientras se realiza la modificación o denegar la acción como se muestra en la Figura 2.4.

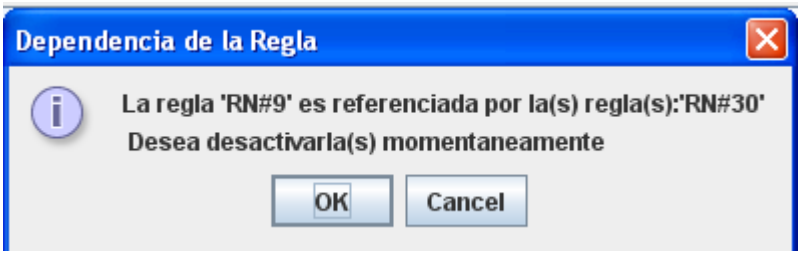



Figura 2.4 Dependencia entre Reglas

Si la opción es positiva se hace un paso intermedio antes de la modificación, la desactivación de todas las reglas que están en la lista reglasdepend mediante el método public void desact(String rules), a este se le pasa el id de la regla.

Después de realizado el cambio, guardar llama al método llenarDependencia donde se activan todas las reglas de la lista reglasdepend como se muestra a continuación:

```
public void llenarDependencia(){
  if(!reglasdepend.isEmpty())
        {
            RuleXml aux;
            for (int i = 0; i < reglasdepend.size(); i++) {
                aux= r.getRule(reglasdepend.get(i));
                System.out.println(reglasdepend.get(i));
               guardarRegla(2,reglasdepend.get(i));
                aux.setStatus("activa");
                aux.setVersion("1.0");
                aux.update();
                r.updateRule(aux);
```

```
                fillRuleData();
                updateMetadataRepository(aux);
            }
        }
    }
```

Es bueno aclarar que el caso que solo se quiera desactivar la regla la interfaz le dará la opción de desactivar la regla en cascada.

## 2.4 Descripción de las Modificaciones de la Clase Principal de LPT-SQL

La clase principal de la herramienta, nombrada Syntax se describe a continuación.

Los principales atributos de esta clase principal son: r y gen del tipo de clase RuleRepositoryXml y RuleGenerationRepositoryXml respectivamente, estos posibilitan el proceso de compilación y generación de una regla a partir de un formato xml. Por su parte Compiler y generator son las clases encargadas de llamar al compilador y al generador en cada caso.

El String data, valor nuevo agregado a la aplicación, guarda en cada caso la regla sin realizarle la modificación para en caso que el tipo de cambio no sea válido poder mantenerse en la versión real de la regla.

El arreglo de tipo String chSujeto conserva la regla dividida por partes como <determinante> <Sujeto> no puede tener <características> para así controlar los posibles cambios del determinante o del sujeto que no son válidos.

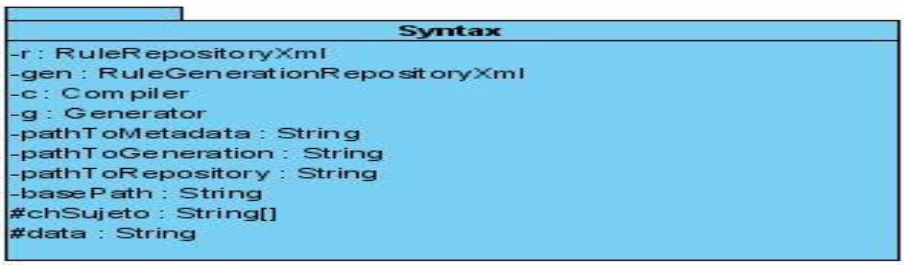



Figura 2.5 Atributos de Clase Principal

Los métodos básicos son compileRule, compilar, addRule, generateRule, newRule, newRepo. Todos ellos como describe su nombre trabajan con las funciones básicas

sobre la regla como crearla, compilarla, realizar el proceso de generación y añadir la regla al repositorio. Todos estos trabajan por igual para cualquier patrón o sea para algún tipo de regla sea de restricción o clasificación por ejemplo el proceso de compilar y generar en esta clase es el mismo. La diferencia en la generación de cada regla de negocio no es apreciable solo hasta que es activada la clase correspondiente que si tiene en cuenta el tipo en cada caso. Estos cambios serán explicados posteriormente.

Para tratar los correspondientes a la modificación se muestran los siguientes: modifReglaRestriccion, modifReglaClasificacion, modifReglaNotificacion, modifReglaCalculo. Todos estos de acuerdo al patrón analizan los cambios realizados y ven si estos están dentro de los previstos, estos ya han sido analizados anteriormente. A su vez estos métodos llaman a otros como modifcaminoaveg y cambioPatron, en los cuales el primero se encarga de que de las características lo referente al camino de navegación no sufra cambios ya que esta cambiaria como se muestra en el tipo restricción el significado de la regla y por lo tanto sería una nueva no una simple modificación y el segundo de que en caso de que sea posible partir de una forma de patrón y llegar a su homologo este se encarga de se haya realizado correctamente.

Todos estos métodos son recogidos y son llamados a partir del tipo de regla con la que se esté trabajando en el momento, ellos devuelven una variable de error que es cero si no hubo problemas y mayor que cero en caso contrario. En el segundo caso la regla regresa a su estado anterior guardada en data como se explicó anteriormente y en el espacio de salida de la aplicación es escrito el tipo de error y la descripción del mismo. En el primero se procede a analizar si la regla posee algún tipo de dependencia ya que en ese caso no es posible la modificación, en la aplicación se ofrece una variante en la que las reglas son desactivadas momentáneamente y activadas luego de la regla sufra el cambio, decir que este proceso en el que las reglas que la utilizan pasan por este proceso es invisible para el usuario, solo se le da posibilita la opción de aprobar esta acción. Si todo va bien es llamado el método guardarRegla el cual se encarga no solo de actualizar todos los mecanismos que generan la regla en la base de datos sino también de realizar los cambios pertinentes

dentro del repositorio de reglas, se muestra en la salida el proceso de generación de la regla y actualización satisfactoria si no ha ocurrido ningún error.

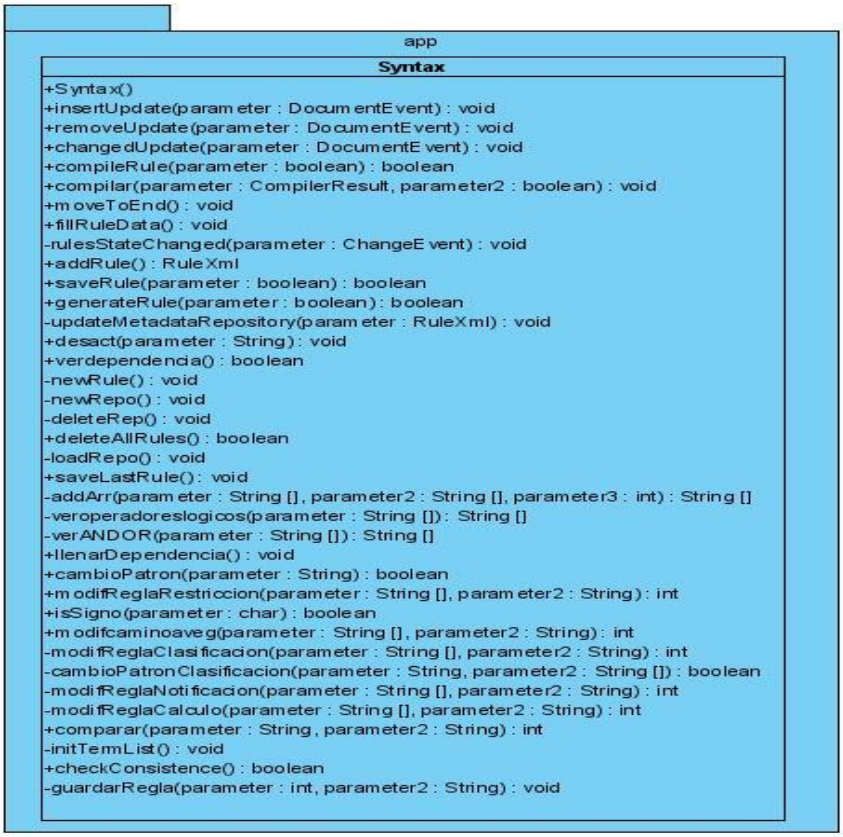



Figura 2.6 Métodos de la Clase Principal

## 2.5 Conclusiones parciales del capítulo.

En este capítulo se establecen las modificaciones válidas de los elementos de los patrones de las reglas para realizar un cambio de una regla por otra. Se describe la arquitectura propuesta en trabajo precedente, para la implementación automática de las reglas para describir los repositorios y componentes del proceso de generación. Asimismo se describen los métodos e interfaces utilizadas para la codificación en java de las implementaciones de las reglas.

# CAPÍTULO 3: Validación y prueba de la herramienta LPT-SQL.

En este capítulo se exponen los aspectos que se consideraron para conformar las clases de equivalencia. Se muestran los resultados del proceso de prueba para la validación del LPT-SQL. Además se describen algunos de los elementos para la utilización del LPT-SQL, que se consideran fundamentales..

## 3.1 Consideraciones para la creación de las clases de equivalencia y los datos de prueba.

Una prueba de caja negra debe ser capaz de obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Sin dudas un paso de mucha importancia en el diseño de la estrategia de prueba lo constituye la obtención del conjunto de condiciones de entrada (Sommerville, 2002).

En esta problemática el conjunto de condiciones de entradas debe estar integrado por un grupo de RN escritas en LPT que logren probar todas las funcionalidades del LPT-SQL.

Es lógico pensar que mientras más reglas conformen el conjunto de entradas, mejores resultados se pueden obtener con la prueba. Probar la aceptabilidad de LPT-SQL respecto a cada una de las posibles RN escritas en LPT, sin dudas sacaría a la luz todos los posibles errores en la implementación de esta herramienta, pero si se tiene en cuenta la diversidad de formas que pueden adoptar cada uno de sus elementos variables en los patrones de regla mostrados en la Tabla 1.1, es fácil apreciar que resultaría extremadamente complicado, convirtiéndose en una prueba de software exhaustiva, que puede llegar a contar con un enorme número de entradas a probar, resultando ser un proceso de prueba impracticable.

Como ejemplo de lo anterior a continuación se muestran dos reglas de restricción que comparten el mismo sujeto pero se están restringiendo sus características de manera diferente.

Lenguaje natural:

- ✓ RN#1: Un estudiante no puede tener más de 35 años.

- ✓ RN#5: Un equipo no puede tener fecha de inicio de un juego menor que '3/11/12'.

Lenguaje LPT:

- ✓ RN#1: Un Estudiante no puede tener sujeto.edad >35
- ✓ RN#5: Un Equipo no puede tener min( sujeto.Deporte.Celebra.fecha_inicio) <'3/11/12'

Como se puede observar el elemento <características> perteneciente al patrón restricción puede adoptar una gran diversidad de formas que van desde las más sencillas (utilización de un solo camino de navegación y un solo operador de comparación) hasta las más complejas (utilización de más de un camino de navegación, aparición de una o varias funciones de conjunto, etc.). En la Tabla 1.1 de los elementos variables de los patrones, se puede apreciar el amplio dominio que contienen algunos de ellos como el caso de las características y los hechos. La posible aparición de varios de estos elementos en una misma regla, dotan al LPT de una amplia diversidad de formas para expresar RN.

Por otro lado, si con el objetivo de obtener una prueba más simple se reduce considerablemente la cantidad de entradas del conjunto de pruebas, se pudiera omitir durante el proceso de búsqueda de defectos el chequeo de algunas de las funcionalidades del software y de este modo arribar a conclusiones inciertas. Se debe tener presente que el objetivo de toda búsqueda de defecto es precisamente encontrar defectos, mientras menos funcionalidades se prueben, menos errores pueden aparecer.

Atendiendo a lo anterior se debe lograr un equilibrio respecto al número de entradas que deben conformar el conjunto de pruebas. Con vista a satisfacer uno de los objetivos imprescindibles en esta investigación, que es la prueba de las funcionalidades del LPT-SQL, se tratará de obtener el menor número de entradas posibles, que logre probar todas las funcionalidades del software.

Seleccionar el adecuado conjunto de entradas a probar constituye el núcleo del diseño de la estrategia de búsqueda de defectos. Como resultado del estudio de las diferentes tendencias teóricas respecto a la creación de este conjunto, en esta investigación se adopta la creación de clases de equivalencia. Cada una de las clases o grupos de equivalencia estarán conformadas por aquellas entradas que provocan un

comportamiento similar en el software.

Una vez obtenidas las diferentes clases se seleccionan las entradas que más representen a su clase y las entradas menos comunes o más atípicas dentro de su clase o grupo. La unión de todas estas entradas constituye el conjunto de prueba para nuestra estrategia de búsqueda de defectos.

## 3.2 Propuesta de términos que agrupan comportamiento similar.

La elaboración de las clases de equivalencia se fundamenta en el análisis de la diversidad de formas en que pueden presentarse los elementos variables de los patrones de reglas escritas en LPT. Las reglas serán agrupadas atendiendo a la forma en que aparecen cada uno de sus elementos. Con esta idea las siguientes reglas pertenecen a una misma clase:

- ✓ RN#1: Un Estudiante no puede tener sujeto.edad >35
- ✓ RN#6: Un Participacion_Cultuta no puede tener sujeto.numero_actos >10

Nótese que en ambas reglas las <características> están conformadas por un camino de navegación simple (aparece una sola tabla en el camino), para acceder a un atributo entero, el cual es comparado con una constante numérica.

Dada la importancia de representar en el proceso de prueba cada una de las posibles formas que pueden tomar los elementos de los patrones escritos en LPT, se concede especial atención a la diversidad de estructuras de estos elementos mostrada en la Tabla 1.1 y se propone un análisis detallado para cada uno de ellos, conformando posteriormente las clases de equivalencia.

Cada uno de los elementos pertenecientes a los patrones de reglas pueden ser instanciados de muchas formas diferentes utilizando caminos de navegación (los cuales pueden a su vez expresarse de diversas formas de acuerdo con la notación punto), operadores, funciones, cadenas de caracteres, términos del negocio, etc.

Resulta conveniente generalizar cada una de estas formas de acuerdo al objetivo específico que se persigue con cada una de ellas. La idea radica en definir un conjunto de términos que generalicen un comportamiento similar. Por ejemplo, existen muchas formas de obtener un elemento individual, ya sea por la utilización de un camino de navegación, el resultado de una operación aritmética, el uso de una

constante, entre otras, pero el objetivo que se persigue es el mismo. A continuación se definen los términos generalizadores que se estiman necesarios para lograr esta generalización:

booleano:= EXIST(elemento, colección)

| EMPTY (colección)

| elemento OP_COMPARACION elemento

| NOT (booleano)

| booleano OP_LOGICO booleano

colección:= colección OP_COMPARCION elemento

| colección OP_COMPARCION colección

| camino

elemento:= OP_CONJUNTO (coleccion)

| camino

| CONSTATE

| elemento OP_ARITMETICO elemento

camino:= tabla

| tabla.camino

tabla:= NTABLA

| NTABLA.ATRIBUTO

| NTABLA (macheo)

Es válido destacar que estos términos generalizadores se obtienen como resultado de un proceso de análisis y transformación sobre la gramática del LPT.

Como se puede apreciar existen interrelaciones entre muchos de estos términos las cuales son necesarias para lograr una completa representatividad del lenguaje. Las interrelaciones entre sus términos se pueden expresar en forma de grafo como se muestra en la Figura 3.1. Como se puede observar las aristas dirigidas indican las dependencias que existen entre cada uno de los términos generalizadores.

De esta forma para la obtención de un booleano es necesario primeramente haber obtenido una colección y/o un elemento individual, estos a su vez pueden ser obtenidos a través de caminos de navegación, los cuales son conformados por tablas.

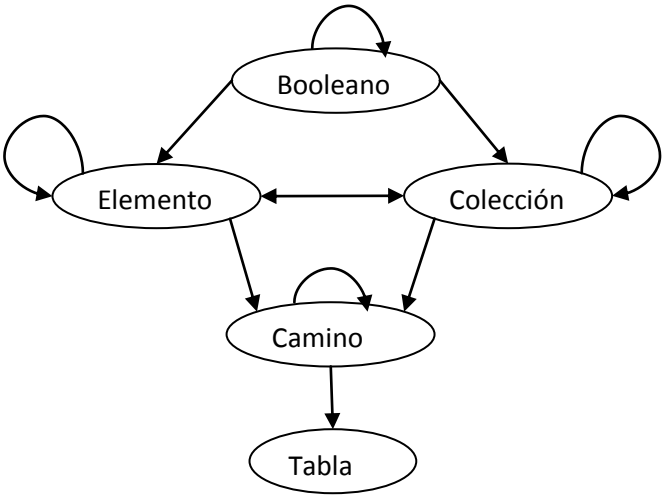



Figura 3.1 Interdependencias entre los términos generalizadores

## 3.3 Creación de las clases de equivalencia.

Como se puede apreciar los elementos variables de los patrones de regla que pueden influir en el comportamiento del software y su representación de manera general serían:

- ✓ <sujeto>: NTABLA
- ✓ <algoritmo> := elemento
- ✓ <características> :=booleano
- ✓ <hechos> :=booleano

NTABLA:=<nombre de una entidad del negocio >

El paso más importante en nuestra estrategia de prueba es conformar clases de equivalencias que logren agrupar las RN donde estos elementos sean similares.

Se puede apreciar que el elemento <sujeto> no posee un amplio dominio para ser instanciado, solo bastaría con conformar dos clases de equivalencias donde tome el valor correspondiente a una entidad de la base de datos o una clasificación de la misma. De este modo como casos representativos para cada una de las clases de equivalencia se tienen:

- ✓ RN#1: Un Estudiante no puede tener sujeto.edad >35
- ✓ RN#8: Un EstConDificultades no puede tener sizeof(sujeto.Participacion_Cultura)

Donde el <sujeto> en la primera regla es Estudiante y se corresponde con la tabla Estudiante de la BD del negocio y el <sujeto> de la segunda regla es EstConDificultades y se corresponde con dicha clasificación del negocio, generada mediante la regla:

RN#7: Un Estudiante es definido como EstConDificultades si sujeto.promedio< 3.5

También es necesario seleccionar de ambas clases de equivalencias aquellos casos más atípicos o poco comunes, bastaría con probar RN donde el <sujeto> de la regla se corresponda con una tabla de la BD del negocio cuyo nombre involucre caracteres poco comunes. También es válido introducir RN donde el <sujeto> no sea correcto, de este modo se prueba la existencia de un chequeo de errores.

De manera diferente a como ocurre en el elemento variable <sujeto>, los restantes elementos poseen un amplio margen de posibilidades para ser instanciados. Como se vio en el caso del <sujeto> solo basta con la creación de dos clases de equivalencias para cubrir todo el dominio de posibilidades, pero en los restantes elementos no sucede lo mismo, tal es caso de las <características> las cuales pretenden como objetivo la restricción de los atributos del elemento <sujeto>. Como se explicó con anterioridad, su resultado debe ser un elemento booleano y solo están limitadas a la existencia de un camino de navegación que parta del sujeto de la regla. En la Figura 3.1 se puede observar que la obtención de un elemento booleano depende de la obtención previa de una colección de elementos y/o un elemento individual, a su vez estos últimos en muchas ocasiones son obtenidos mediante la utilización de caminos de navegación. Por lo tanto la creación de las clases de equivalencias que logren agrupar las reglas donde la estructura de sus características provoque un comportamiento similar en el software debe partir de la base de la posible utilización de caminos de navegación que garanticen la obtención de un elemento individual o una colección de elementos, para posteriormente agruparlas por las formas de obtención del elemento booleano. De este modo cuando se desee probar la funcionalidad de <características>, se debe tener certeza del correcto funcionamiento

de todas las alternativas definidas para la obtención de un <elemento> o una <colección>, para lo que es necesario haber probado diversos caminos de navegación.

De acuerdo con lo anterior la primera clase o grupo a conformar serían los elementos más sencillos o que no dependen de ningún otro elemento. Atendiendo a la propuesta de elementos para expresar comportamiento general se está hablando de <tabla>. Esta clase agrupará todas las posibles entradas en las que se utilice una tabla de las formas:

- ✓ NTABLA, se hace referencia solo al nombre de la tabla con intención de acceder al atributo llave primaria.
- ✓ NTABLA.ATRIBUTO, se hace referencia a un atributo de la tabla.
- ✓ NTABLA (macheo), se restringen las filas de la tabla mediante un macheo, para acceder al atributo llave.

Para seleccionar el conjunto de entradas que representarán esta clase o grupo, bastaría con encontrar una entrada similar a cada alternativa y otras que pudieran ser consideradas casos extremos o poco comunes, los que quizás el programador haya olvidado.

Ejemplo de las formas anteriores:

- ✓ RN#9: Un Festival no puede tener sujeto >2013
- ✓ RN#10: Un Estudiante no puede tener sujeto.promedio < 3
- ✓ RN#11: Un Criollos no puede tener sujeto.edicion > Criollos(lugar=5)

Como casos poco comunes pueden ser probados tablas en que la llave primaria está compuesta por varios atributos o nombres de tablas que involucren caracteres poco comunes.

Ejemplo:

- ✓ RN#12: Un Participacion_Cultura no puede tener sujeto.cantidad_actos>6

Para probar tanto los casos atípicos como los representativos, las entradas seleccionadas pueden en muchas ocasiones carecer de sentido en el ámbito del negocio, pero cumplir un objetivo bien específico en el proceso de prueba.

Posteriormente se conforman las clases que agrupen las RN que utilizan caminos de navegación similares. Para este caso el número de clases de equivalencia a crear puede crecer tanto como nivel de detalles quiera alcanzar el probador, teniendo en cuenta que en un mismo camino de navegación pueden aparecer todas las tablas involucradas en la BD del negocio y que sobre cada una de ellas es posible realizar un macheo sobre cada uno de sus atributos. En esta investigación se propone la creación de las clases en función de las interrelaciones entre las tablas por las cuales se está navegando (1: M, 1:1) y en menor medida el número de tablas que aparecen.

Por tanto se debe conformar una primera clase de equivalencia que agrupe las reglas que utilicen una navegación por tablas con interrelación 1: M y/o 1:1, estos caminos de navegación pueden incluir varias tablas y estas a su vez incluir un macheo.

Las clases de equivalencia agruparían las reglas donde se utilice

*1-* <tabla>, reglas donde se utilicen caminos de navegación de una sola <tabla> en cualquiera de las alternativas vistas anteriormente.

*2-* <tabla>.<camino>, reglas que utilicen caminos de navegación con más de una tabla y existan interrelaciones de 1: M y/o 1:1.

De este modo como ejemplos representativos de casos se tienen:

*3-* RN#13: Un Estudiante no puede tener sujeto.año >5

*4-* RN#14: Un Estudiante no puede tener sum(sujeto.Participacion_Cultura.cantidad_actos)>4

*5-* RN#15: Un Estudiante no puede tener sizeof(sujeto.Equipo.Deporte('Pelota_M').Copite.Criollos.lugar)>5

Una vez conformadas las clases de equivalencia que logren probar el correcto funcionamiento de los caminos de navegación se crean las clases de equivalencia correspondientes a la obtención de una colección de elementos y un elemento individual.

Estas clases agrupan las reglas donde se intente obtener un elemento individual por alguna de las siguientes vías:

✓ OP_CONJUNTO (colección), reglas donde después de obtenida una colección de elementos se realiza una operación de conjunto (sum, sizeof, etc.) sobre ella.

✓ camino, reglas donde utiliza un camino de navegación que devuelve un elemento individual (en este caso los caminos de navegación no deben incluir interrelaciones 1: M).

✓ CONSTATE, reglas que utilizan una constante.

✓ elemento OP_ARITMETICO elemento, reglas donde aparece una operación aritmética (+, /,*,-) entre dos elementos.

De manera similar es posible la obtención de una colección de elementos por una de las siguientes alternativas:

✓ colección OP_COMPARCION elemento, reglas donde aparece una operación de comparación entre un elemento individual y una colección de elementos.

✓ colección OP_COMPARCION colección, reglas donde aparece una operación de comparación entre dos colecciones de elementos.

✓ Camino, reglas donde utiliza un camino de navegación que devuelve un elemento individual (debe existir al menos una interrelación 1: M entre dos tablas involucradas en el camino de navegación).

Tanto en estas clases de equivalencia como en las que representan la obtención de una colección de elementos, se debe tener en cuenta el tipo de datos sobre el cual se está trabajando. Es necesario probar que es posible obtener un elemento o una colección de elementos para cada uno de los tipos de datos almacenados en la BD del negocio.

Algunos de los casos representativos serían:

*1.* RN#16: Un Estudiante puede tener

sum (sujeto.Participacion_Cultura.cantidad_actos)>5

*2.* RN#17: Un Criollos no puede tener sujeto.lugar*5<25

*3.* RN#18: Un Deporte no puede tener min(sujeto.Celebra.fecha_inicio)<’18/11/2012’

Las reglas 1 y 3 están bajo la misma clase de equivalencia (reglas donde se obtiene un elemento mediante la aplicación de un operador de conjunto sobre una colección de elementos), pero se muestran como casos separados por el tipo de datos sobre el cual se está operando. En la primera regla sobre datos tipo numérico y en la tercera sobre datos fecha.

Una vez asegurado que es posible la obtención de una colección de elementos y un elemento individual, se procede a agrupar las reglas que realicen de manera similar las operaciones sobre ellos para obtener un elemento booleano. Las posibles operaciones son:

- ✓ EXIST (elemento, colección): Pregunta si existe el elemento en la colección.
- ✓ EMPTY (colección): Pregunta si la colección está vacía.
- ✓ elemento OP_COMPARACION elemento: Realiza una operación de comparación entre dos elementos.
- ✓ NOT (booleano): Niega un resultado booleano.
- ✓ booleano OP_LOGICO booleano: Realiza una operación booleana entre dos elementos booleanos.

Debido al orden lógico que se ha establecido durante la creación de las clases de equivalencia, en este nivel solo es necesario probar el funcionamiento de los operadores de existencia (EMPTY, EXISTS), la comparación entre dos elementos individuales, los operadores booleanos y las posibles combinaciones de los mismos; cualquier vía de obtención de un elemento individual o una colección de elementos ya ha sido probada mediante las clases de equivalencias correspondientes a esos elementos. De este modo, del grupo de reglas siguientes, las reglas 20 y 21 pertenecen a una misma clase de equivalencia (reglas de restricción en su primera alternativa donde se combinan operadores de comparación y operadores booleanos) no así en el caso de la regla 1 (reglas de restricción en su primera alternativa donde se usa el operador EMPTY):

1. RN#19: Un Estudiante no puede tener EMPTY (sujeto.Participacion_Cultura)
2. RN#20: Un Criollos no puede tener sujeto.edición >2010 and sujeto.lugar >10

*3.* RN#21: Un Festival no puede tener sujeto.presupuesto > 200 and sujeto.lugar>5

En los ejemplos mostrados solo aparecen reglas de tipo restricción pero este análisis es similar para cada uno de los tipos de reglas, donde aparece el elemento variable <características>.

Para probar la funcionalidad de los <hechos> el procedimiento es similar, aunque no es obligatorio la existencia de al menos un camino de navegación que parta de la tabla sujeto, y en el caso del <algoritmo> se reduce a probar las posibles combinaciones de obtener un elemento individual del tipo numérico.

Entre las regla de negocio con una perspectiva de datos pueden presentarse interdependencias. Una vez probadas las funcionalidades de cada una de las reglas por separado se deben analizar las posibles interdependencias entre ellas. Para este análisis se es posible la creación de clases de equivalencias que agrupen la interrelación de un tipo de regla con otra. Por tanto se deben chequear los siguientes casos mostrados.

| **Tipo de Regla** | **Posibles Reglas a depender** |
|---|---|
| Clasificación | Clasificación<br>Cómputo |
| Cómputo | Cómputo |
| Notificación | Clasificación<br>Cómputo |
| Restricción | Clasificación<br>Cómputo |

Tabla 3.1 Interrelaciones entre los tipos de regla

### 3.3.1 Resultados de la prueba

A partir de la creación de las clases de equivalencia se conforman un conjunto de entradas para probar las funcionalidades del LPT-SQL. Durante la ejecución de la prueba se deben tener en cuenta que el LPT-SQL es capaz de generar e insertar las RN en bases de datos sobre los gestores PostgresSQL y SQL Server, por tanto cada una de las reglas que conforman el conjunto de entradas a probar, debe ser generada e insertada adecuadamente para cada uno de los gestores. A continuación de muestra el conjunto de entrada:

En esta investigación se trabaja con las Bases de Datos FEU Y Menú, diseñadas en trabajos anteriores.

- <sujeto>: NTABLA

**Clase 1**

- ✓ RN#1: Un Estudiante no puede tener sujeto.edad >35
- ✓ Un estudiante no puede tener sujeto.participacion_cultura.cantidad_actos >sujeto.festival.numero_actos
- ✓ Un deporte no puede tener sujeto.min_integrantes>sizeof(Equipo)+5
- ✓ Un estudiante no puede tener sujeto.numero_estudiante< 2
- ✓ Un equipo no puede tener sujeto.promedio+promediodeportivo6>7
- ✓ Los ingredientes_adicionales no puede tener sujeto.cant_total > sujeto.producto.cant_existente and sujeto.fundamental='si'
- ✓ Un plato_consumir no puede tener sujeto.ingredientes_adicionales.cant_total> sujeto.ingredientes_adicionales.producto.cant_existente and sujeto.ingredientes_adicionales.fundamental='si'

**Clase 2**

- ✓ Un estudiante puede tener sizeof(sujeto.Participacion_Cultura)>0 solo si sujeto.promedio>3.5
- ✓ Un estudiante no puede tener sizeof(sujeto.estudiante_investigacion)=1
- ✓ Un estudiante_investigacion no puede tener sujeto.estudiante.anno>5
- ✓ Un estudiante no puede tener sizeof(sujeto.estudiante_investigacion)=1 and desaprobado(sujeto)
- ✓ Un menu no puede tener sizeof(sujeto.plato_consumir.id)>12
- ✓ Un menu no puede tener

sizeof(sujeto.plato_consumir.plato.categoria(nombre='Potajes'))>1

- ✓ Un menu no puede tener sizeof(sujeto.plato_consumir.plato.categoria(nombre='Cereales'))>1 or sizeof( sujeto.plato_consumir.plato.categoria(nombre='Pastas'))>1 or (sizeof(sujeto.plato_consumir.plato.categoria(nombre='Cereales')) + sizeof( sujeto.plato_consumir.plato.categoria(nombre='Pastas')))>1

  - <características> :=booleano

**Clase 3**

- ✓ RN#13: Un Estudiante no puede tener sujeto.año >5
- ✓ RN#17: Un Criollos no puede tener sujeto.lugar*5<25

**Clase 4**

- ✓ RN#14: Un Estudiante no puede tener sum(sujeto.Participacion_Cultura.cantidad_actos)>4
- ✓ RN#15: Un Estudiante no puede tener sizeof(sujeto.Equipo.Deporte('Pelota_M').Copite.Criollos.lugar)>5
- ✓ RN#18: Un Deporte no puede tener min(sujeto.Celebra.fecha_inicio)<'18/11/2012'

**Clase 5**

- ✓ RN#19: Un Estudiante no puede tener EMPTY (sujeto.Participacion_Cultura)
- ✓ RN#20: Un Criollos no puede tener sujeto.edición >2010 and sujeto.lugar >10
- ✓ RN#21: un festival no puede tener sujeto.presupuesto>200 and sujeto.presupuesto<5

  - <algoritmo> := element

**Clase 6**

- ✓ El PromedioDeportivo3 es calculado como sizeof(Equipo)+5
- ✓ El PromedioDeportivo6 es calculado como avg(Equipo.promedio)
- ✓ El Promedio_Estudiantes es calculado como sum(Estudiante.promedio)/sizeof(Estudiante)
- ✓ El promedio_integral es calculado como (avg(Equipo.Estudiante.promedio) + avg(Participacion_Cultura.Estudiante.promedio)+avg(Estudiante_Investigacion.Estudiante.promedio))/3

**Clase 7**

- ✓ El PromeEquipo en Deporte es calculado como avg(sujeto.estudiante.promedio)+5
- ✓ La Matricula en Centro_Investigacion es calculado como sizeof(sujeto.Grupo_Investigacion)

**Clase 8**

- ✓ El LUGAR2 en Criollos para lugar es calculado como sujeto.edicion+100
- ✓ El WWW en Deporte para lugar_obtenido es calculado como sujeto.max_integrantes*max(Estudiante.promedio)
- ✓ El RRR en Participacion_Cultura para cantidad_actos es calculado como max(sujeto.Estudiante.Equipo.Deporte.Compite.Criollos.lugar)+100
- ✓ El total_ingredientes en ingredientes_adicionales para cant_total es calculado como ((sujeto.cant_requerida) * (sujeto.plato_consumir.menu.cant_comensales) * (sujeto.plato_consumir.racion))
    - <hechos> :=booleano

**Clase 9**

- ✓ Un Estudiante es definido como Desaprobado si sujeto.promedio<3
- ✓ Un Estudiante es definido como EstConDificultades si sujeto.promedio< 3.5

**Clase 10**

Notificar 'Existen estudiantes desaprobados' si not(Empty(estudiante.promedio<3))

Notificar 'Notificacion' si Estudiante.numero_estudiante>2

Estas entradas son nuevamente agrupadas en sus clases de equivalencia para determinar si el error involucra toda su clase o algunos elementos en específico. Posteriormente son analizadas con un nivel de detalle más profundo para encontrar relaciones entre algunos de los defectos. Terminado este análisis se corrigieron los siguientes defectos.

1. Problemas con los operadores lógicos para modificar uno por otro.
2. Agrupar las reglas dependientes de la que se está modificando.
3. Poder desactivar una regla y sus dependencias si fuera necesario.

## 3.4 Características Generales de la Herramienta Generadora de Reglas de Negocio

En la Figura 3.2 se presenta la ventana principal de la aplicación, esta se viene trabajando desde (Perez Alonso, 2010) hasta su versión anterior en (Ríos Méndez, 2013).

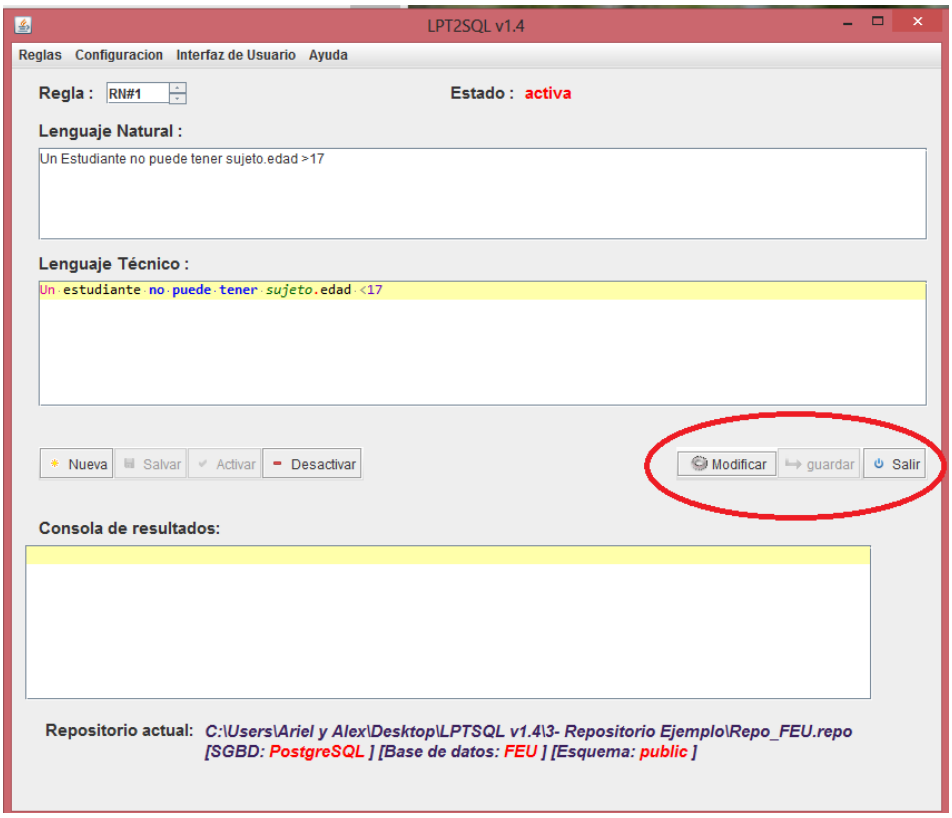



Figura 3.2 Ventana Principal

Como paso inicial para el trabajo con la aplicación es necesario estar conectado a una base de datos ya sea en postgresSql o en SqlServer, para esto la aplicación cuenta con una interfaz como la mostrada en la Figura 3.3. En esta ventana se selecciona con cuál de los dos gestores mencionados se conectará.

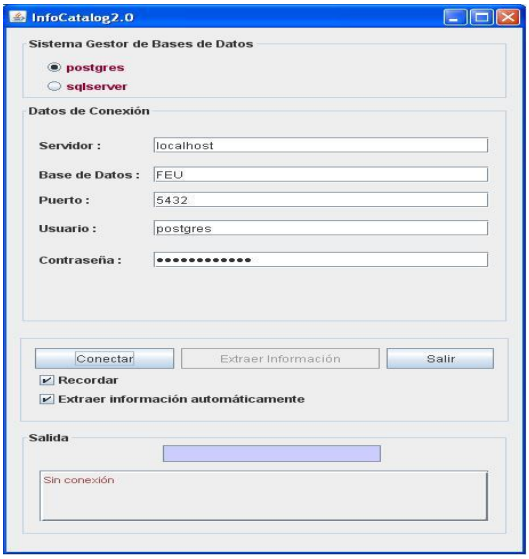



Figura 3.3 Interfaz para obtener la información de la base de datos en PostgreSQL

En el caso de postgres se insertan los datos referentes a la conexión como el nombre del servidor, la base de datos, el usuario permitido para la conexión y su respectiva contraseña. Luego de registrados los datos se provee la posibilidad de que se extraigan los metadatos de manera automática y de que recuerden estos datos permitiendo tener estos datos guardados para otra futura conexión.

Si la BD está implementada en SQL Server es necesario crear y conectarse al origen de datos de la BD física. Para crear un nuevo origen de datos marcamos el enlace y seguimos los siguientes pasos:

1. Seleccionar Orígenes de datos ODBC.
2. Seleccionar la opción Agregar (Para crear un nuevo origen de datos).
3. Seleccionar el driver ODBC para SQL Server.
4. Especificar el nombre para referirse al nuevo origen de datos.
5. Especificar el nombre del Servidor.
6. Especificar cómo desea que SQL Server compruebe la autenticidad del Identificador inicio de sesión.

7. Establecer la base de datos en cuestión como predeterminada.
8. Comprobar orígenes de datos.

La Figura 3.4 y La Figura 3.5 muestran las ventanas para crear un nuevo origen de datos y la comprobación de este al terminar de crearlo.

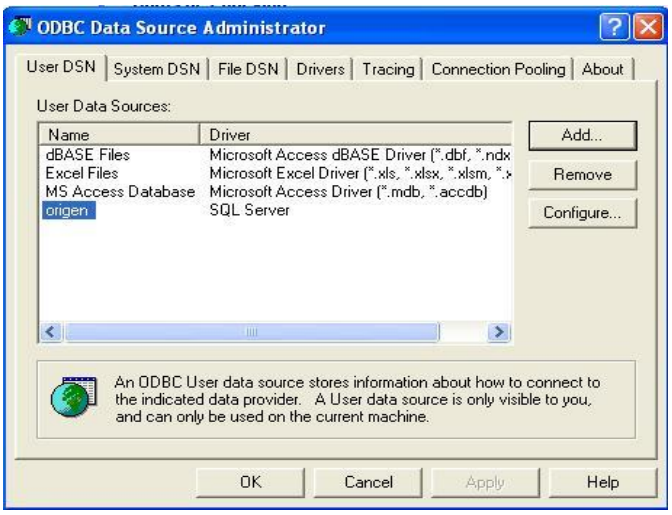



Figura 3.4 : Crear un nuevo origen de datos ODBC

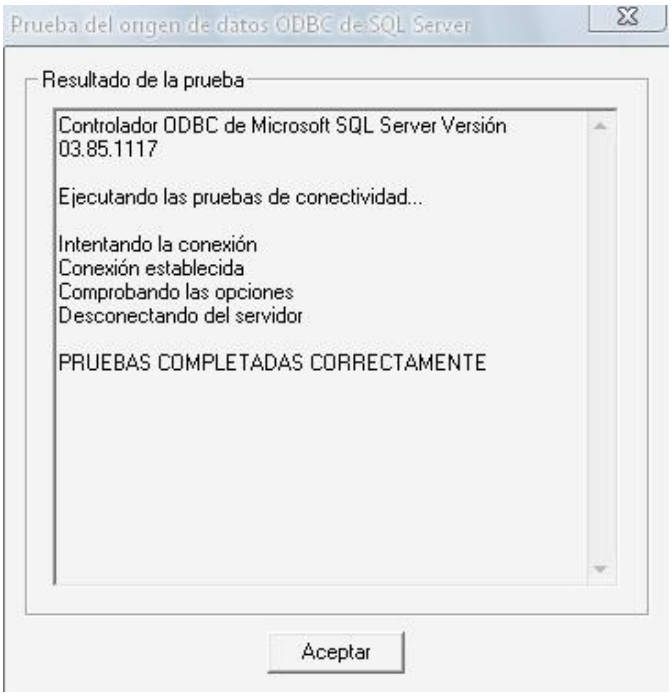



Figura 3.5 Resultado de la Conexión

Una vez creado el origen de datos de la BD en MS SQL Server para obtener sus Metadatos se selecciona el gestor SQL Server y se le provee el nombre del origen como se muestra en la Figura 3.6.


Figura 3.6 Extracción de la información de la base de datos en SQL Server

Para proseguir es necesario cargar el archivo (.repo) que contiene el repositorio de reglas, ver Figura 3.7. Este archivo debe contener el repositorio con la información del catálogo. En caso de no existir se puede extraer nuevamente esta información. De igual manera es posible la creación de un nuevo archivo (. repo).

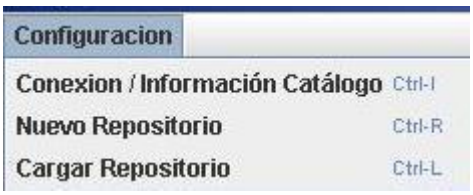



Figura 3.7 Menú de configuración

Una vez que se logre la conexión y se obtenga la información de la base de datos, no es necesario repetir estas operaciones mientras la ventana principal se encuentre activa.

## 3.5 Conclusiones parciales del capítulo.

En este capítulo se conformaron 10 clases de clases de equivalencias para la prueba de caja negra del software LPT-SQL, de modo que se probaron las funcionalidades de la herramienta para las modificaciones de las reglas de cada tipo de categoría. Se mostró el proceso de elaboración de las clases de equivalencia y la obtención del conjunto de reglas a probar. Fueron analizados los resultados de la ejecución de la prueba. Se hizo un análisis del proceso de ejecución del software, dirigido

fundamentalmente a futuros programadores.

# Conclusiones

- ✓ Se analizaron las funciones técnicas de modificación de reglas y se definieron las que son factibles a partir de los componentes de los diferentes tipos de patrón, sea de restricción, cálculo, notificación y clasificación.
- ✓ Se rediseñó la herramienta para incorporarle las funcionalidades de modificación para las categorías de reglas desde la perspectiva de los datos, las posibles modificaciones fueron justificadas.
- ✓ Se implementaron las modificaciones de las reglas basadas en el nuevo diseño y se validó el software a partir de una estrategia de prueba de técnicas de caja negra, las clases de equivalencia.

# Recomendaciones

- ✓ Extender las funcionalidades del LPT mediante la incorporación de nuevas funciones y operadores.
- ✓ Expandir la generación de reglas a otros SGBD.

# Referencias Bibliográficas

BOGGIANO CASTILLO, M. B., MARTÍNEZ DEL BUSTO, M. E., PÉREZ VÁZQUEZ , R. & GONZÁLEZ GONZÁLEZ , L. Aplicando Reglas de Negocio mediante triggers CIE 2007, 2007 Cuba.

CIENTEC. 2011. BRMS: Reglas Más Sintonizadas con el Negocio [Online]. Santiago de Chile. Available: www.cientec.com/management/management-brms.html [Accessed 11-12-2011 2011].

RÍOS MÉNDEZ, J. F. 2013. LPT-SQL v1.3: Herramienta para la generación automática de reglas de negocio en bases de datos relacionales.

ROSS, R. G. 2009. Business Rule Concepts ~ Getting to the Point of Knowledge.

TRILLES, J. J. 2006. Reglas de Negocio y Gestión por Procesos de Negocio.

BAJEC, M. & KRISPER, M. 2001. Managing business rules in enterprises. Electrotechnical Review, Ljubljana, Slovenija, Elektrotehniˇski vestnik 68(4) 236-241.

BAJEC, M., KRISPER, M. & RUPNIK, R. 2000. Using Business Rules Technologies To Bridge The Gap Between Business and Business Applications. In: RECHNU, G. E. (ed.) Proc. of the IFIP 16th World Computer Congress 2000, Information Technology for Business Management. Peking, China.

BAJEC, M., KRISPER, M. & RUPNIK, R. 2005. A methodology and tool support for managing business rules in organisations. Information Systems, Sep 2005, 30, no. 6**,** 423-443.

BOGGIANO CASTILLO, M. B. 2014. GENERACIÓN AUTOMÁTICA DE CÓDIGO PARA LA IMPLEMENTACIÓN DE REGLAS DE NEGOCIO EN BASES DE DATOS RELACIONALES.

BOGGIANO CASTILLO, M. B., MARTÍNEZ DEL BUSTO, M. E., PÉREZ VÁZQUEZ , R. & GONZÁLEZ GONZÁLEZ , L. Year. Aplicando Reglas de Negocio mediante triggers In: CIE 2007, 2007 Cuba.

BOGGIANO CASTILLO, M. B., PÉREZ ALONSO, A., CALDERÓN SOLÍS, A., M.E., M. B., DÍAZ DE LA PAZ, L., PÉREZ VÁZQUEZ, R. & GONZÁLEZ GONZÁLEZ, L. M. 2012. Enfoque de reglas de negocio y sus implementaciones en bases de datos relacionales. Universidad Central "Marta Abreu" de Las Villas. Cuba CITMA-VC, Reconocimiento a nivel provincial 1060/2012.

CALDERÓN SOLÍS, A. 2011. Traductor LPT-SQL para reglas de negocio en bases de datos relacionales.

CIENTEC. 2011. BRMS: Reglas Más Sintonizadas con el Negocio [Online]. Santiago de Chile. Available: www.cientec.com/management/management-brms.html [Accessed 11-12-2011 2011].

DEMUTH, B. Year. The Dresden OCL Toolkit and the Business Rules Approach. In, 2005 Technische Universität Dresden.

DÍAZ DE LA PAZ, L. 2011. Modificación de reglas de negocio creadas automáticamente en bases de datos relacionales.

GARCÍA PÉREZ, A. M. 1997. Un modelo de versiones para la construcción de software de ayuda al diseño. Departamento de Ciencias de la Computación. Universidad Central "Marta Abreu" de Las Villas. Santa Clara. . 116p.

HALLE VON, B. 2001. Business rules applied: building better systems using the business rules

approach. John Wiley & Sons [Online].

HEIDENREICH, F., WENDE, C. & DEMUTH, B. 2005. A Framework for Generating Query Language Code from OCL Invariants. Available: http://opus.kobv.de/tuberlin/volltexte/2008/1803/pdf/ECEASST_Vol_9_2008_05.pdf.

LOWENTHAL 2005. Rule Enabling Applications with Oracle Business Rules.

MARRERO LORENZO, I. 2009. Modificación de las reglas de negocio tipo restricción y su implementación.

MORGAN, T. 2002. Business Rules and Information Systems: Aligning IT with Business Goals, Indianapolis, USA, Addison Wesley.

PEREIRA TOLEDO, A. 2009. Solución al problema de la cardinalidad en la generación automática de reglas de negocio en bases de datos relacionales.

PEREZ ALONSO, A. 2010. Reglas de Negocio en Bases de Datos Relacionales. Maestria, Universida Central "Marta Abreu" de Las Villas.

PÉREZ ALONSO, A. 2008a. Aplicación para reglas de restricción en negocios. Licenciado Trabajo de Diploma, Universidad Central "Marta Abreu" de Las Villas.

PÉREZ ALONSO, A. 2008b. Aplicación para Reglas de Restricción en Negocios.

PÉREZ ALONSO, A. 2010. Reglas de Negocio en Bases de Datos Relacionales. Master en Ciencia de la Computación, Universidad Central "Marta Abreu" de Las Villas.

PÉREZ PEDRAZA, R. 2012. Extensión del Traductor LPT-SQL.

PERKINS, A. 2002. Business Rules Are Meta Data. . Business Rules Journal, 3**,** http://www.BRCommunity.com/a2002/b097.html.

PLOTKIN, D. 1999. David. Business Rules Everywhere. Intelligent Enterprise Magazine.

RÍOS MÉNDEZ, J. F. 2013. LPT-SQL v1.3: Herramienta para la generación automática de reglas de negocio en bases de datos relacionales.

ROSS, R. G. 2003. Principles of the Business Rule Approach, Addison-Wesley Professional.

ROSS, R. G. 2009. Business Rule Concepts ~ Getting to the Point of Knowledge.

ROSS, R. G. 2010. Decision Analysis Using Decision Tables and Business Rules. In: HEALY, K. A. (ed.).

SOMMERVILLE, I. 2002. Ingeniería de softtware.

SPREEUWENBERG, S. 2007a. Rule History and Versioning (Part 1). Business Rules Journal, Vol. 8, No. 11

SPREEUWENBERG, S. 2007b. Rule History and Versioning (Part 2). Business Rules Journal, Vol. 8, No. 12

SPREEUWENBERG, S. 2008. Rule History and Versioning (Part 3). Business Rules Journal, Vol. 9, No. 1.

ZIMBRÃO, G., MIRANDA, R., SOUZA, J. M. D., ESTOLANO, M. H. & NETO, F. P. 2002. Enforcement of Business Rules in Relational Databases Using Constraints. Available: http://www.mii.lt/informatica/pdf/INFO764.pdf.

# ANEXOS

# Anexo 1 Sintaxis del LPT

**Notación**

La siguiente descripción sintáctica utiliza los convenios de la XBNF, incluyendo:

1. (_):=Cualquier elemento del lenguaje.

2. **O** (_):=un (_) opcional.

3. **#** (_):=cualquier número de (_) incluyendo nulo.

4. **N** (_):=uno o más (_).

5. **L**(x, (_)):= cualquier número de (_) separados por x.

6. **List** (_):= **L**( "," , (_) ).

**Gramática.**

**regla**::=restricción | cómputo | clasificación | notificación

**restricción** ::=determinante sujeto ‘no puede tener’ características |determinante sujeto ‘puede tener’ características ‘solo si’ hechos

**cómputo** ::=determinante resultado ‘es calculado como’ expresión_matemática| determinante resultado ‘en’ sujeto ‘es calculado como’ expresión_matemática | determinante resultado ‘en’ sujeto ‘para’ atributo ‘es calculado como’ expresión_matemática

**clasificación** ::= determinante sujeto ‘es definido como’ clasif ‘si’ características | determinante sujeto ‘no es definido como’ clasif ‘a menos que’ características

**notificación** ::= ‘notificar’ mensaje ‘si’ hechos

**clasif** ::= identificador

**sujeto** ::= identificador

**mensaje**::= string_dec

**expresión_matemática**::= exp_valor

**resultado**::=string

**exp** ::= L(‘OR’ , terminoXOR)

**terminoXOR**::= L(‘XOR’ , terminoAND)

**terminoAND**::= L(‘AND’ , termino)

**termino**::= exp_logica |exp_conjunto | ‘(’termino‘)’

**exp_valor**::=L(operador_aritmetico , atomo)

**atomo**::= exp_elemental| ‘(’exp_valor‘)’

**exp_conjunto**::=L(operador_comp ,exp_valor)

**exp_logica**: :=‘exists’ ‘(’exp_elemental ‘,’ exp_conjunto ‘)’ | ‘empty’ ‘(’exp_conjunto‘)’

**exp_elemental**::=operador_conjunto ‘(’exp_conjunto ‘)’ | numero | real | booleano | string_dec | calculo ‘(’sujeto‘)’ |camino

**calculo**::= identificador

**camino**::= ‘sujeto’ O(punto camino_navegacion) | camino_navegacion

**camino_navegacion** ::=L(punto, elemento_nav)

**elemento_navegacion**::= nombre_tabla O( ‘(’ macheo ‘)’)

**macheo**::=O(atributo op_comp )exp_valor

**atributo**::=identificador

**nombre_tabla**:= identificador

**Léxico**

**op_comp**::= ‘>’|‘<’ |‘>=’|‘<=’|‘=’|‘<>’

**operador_logico** :=‘and’ |‘or’ |‘xor’

**operador_conjunto**:= ‘sizeof’|‘sizeofdif’|‘sum’|‘sumdif’|‘avg’|‘avgdif’|‘min’|‘max’

**determinante**::= 'El' |'La' |'Los' |'Las' | 'Un' | 'Uno' | 'Una' | 'Cada' | 'Todos';

**punto**::=‘.’

**dpuntos**::= ‘:’

**esp**::= ‘ ’

**div**::= ‘/’

**esc**::= ‘-’

**delim**::= '\t'

**meol** ::= '\n'

**letra**: := 'a'..'z'|'A'..'Z'|'ñ'|'Ñ'|'á'|'Á'|'é'|'É'|'í'|'Í'|'ó'|'Ó'|'ú'|'Ú'

**esc_char**::='\\' ('n' |'t'|'"'|'\\'|'^' '@'..'_'|digito|(' ' | '\r' | '\n' | '\t' | '\f')+ '\\')

**string**::= ( letra | digito| simbolo | esc_char | ' ' | '\t')

**símbolo**::='!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' |')' | '_' | '-' | '+' | '=' | '{' | 'r' | '[' | ']' |':' | ';' | '<' | '>' | ',' | '.' | '?' | '~' | '/'

**digito**::= '0'..'9'

**nueva_línea** ::= delim O(meol);

**ws:**:=# (' ' | '\n' | '\r' | '\t')

**booleano**::= 'true' | 'false';

**identificador**::= letra #( letra | digito | '_');

**numero**::=:N(digito);

**date**::= digito O (digito) div digito O (digito) div O(digito digito) digito digito

| digito O (digito) punto digito O (digito) punto O(digito digito) digito digito

| digito O (digito) esc digito O (digito) esc O(digito digito) digito digito

**time**::= digito O(digito) dpuntos O(digito) digito O(dpuntos O(digito) digito)

**datetime**::= '\"date O(N(esp) time) '\"

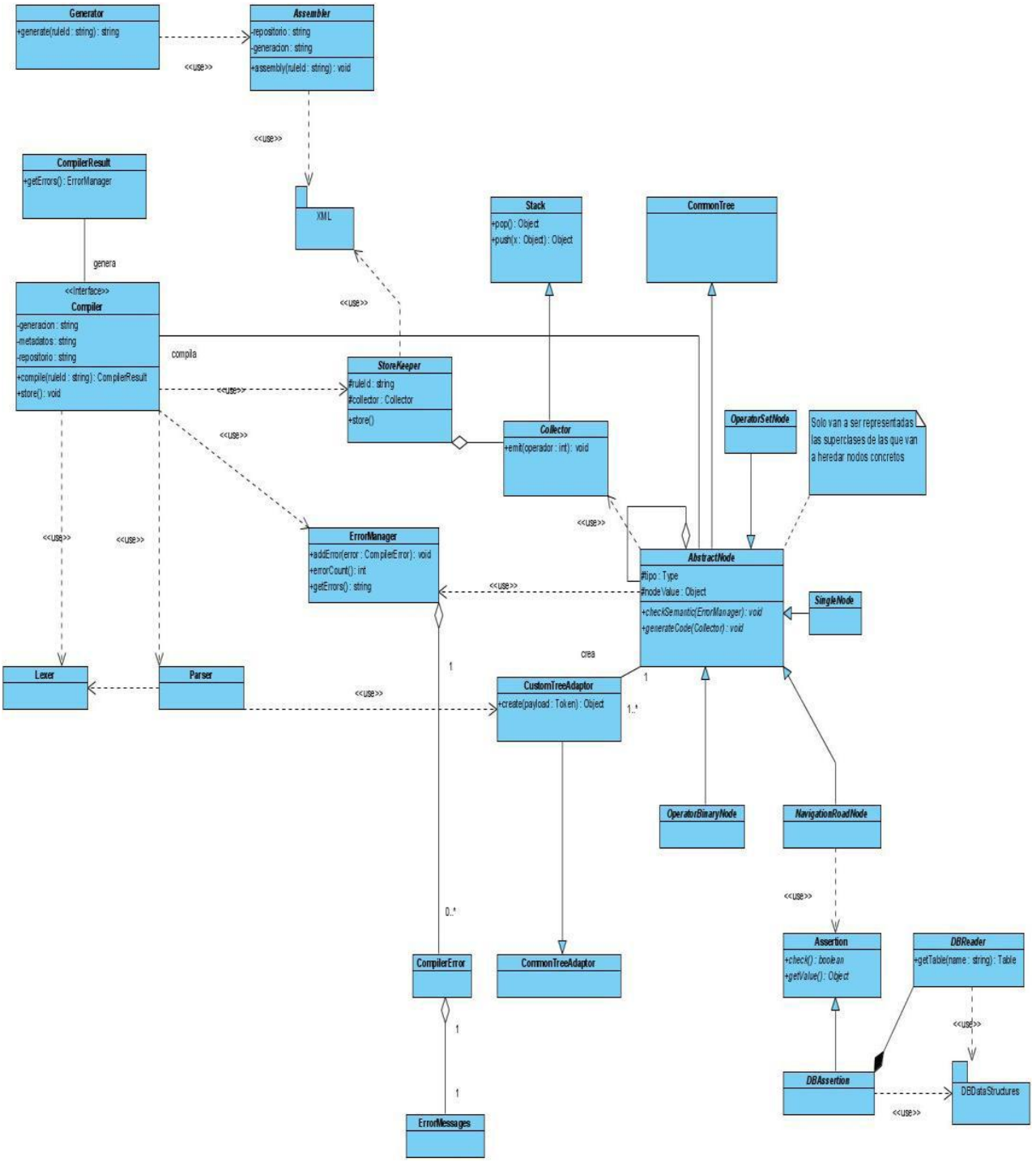
|'\"time '\";

**string_dec**::= "'string"'|'\"string'\";

**real**::= N(digito) O('.'N(digito)) O( ('e' |'e')('+'|'-') N(digito) );

**operador_aritmetico** ::=‟+‟ | „-„ | „*„ | „/ „

# Anexo 2 Diagrama de Clases

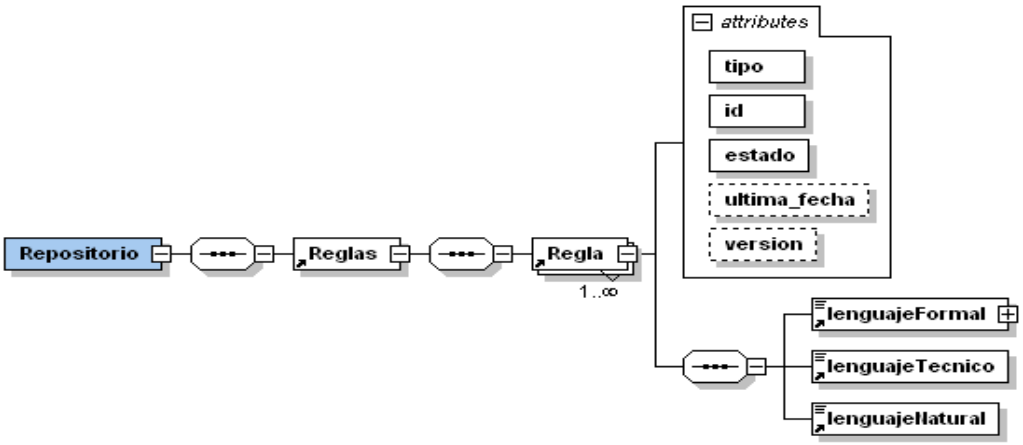


# Anexo 3 Métodos de Generación en Sql Server para regla de restricción

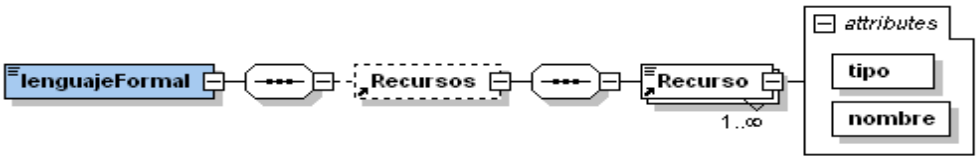
```
public String createTrigger(String triggerName, String event,String statement, String table, String viewName, String id) {

String baseStatement="CREATE TRIGGER %s ON %s FOR %s AS if(EXISTS("+statement+") ) BEGIN raiserror('%s',16,1); rollback transaction; END";

return String.format(baseStatement,triggerName,table, event,viewName,message);

}

  public String createView(String nombre, String sujeto, String exp) {

return String.format("CREATE VIEW %s AS SELECT * FROM %s sujeto WHERE ( %s ) ", nombre,sujeto, exp);

}

public String assembly(String id) {

while (eit.hasNext()) {

        Element curr = eit.next();

        String event = curr.getText();//evento

        String triggerName = String.format("T%s%s_%d", event.substring(0, 2).trim(), id, counter);//nombre

        Element trigger = new Element("Recurso");

        trigger.setAttribute("tipo", "TRIGGER");

        trigger.setAttribute("nombre", triggerName);

        recursos.addContent(trigger);


        String sqlCode = createTrigger(triggerName, event,temp.getChild("CaminoAlSujeto").getText(), table, viewName, id);
```

```
            trigger.setText(sqlCode);
            formal += sqlCode + sep;
        }
    }
```

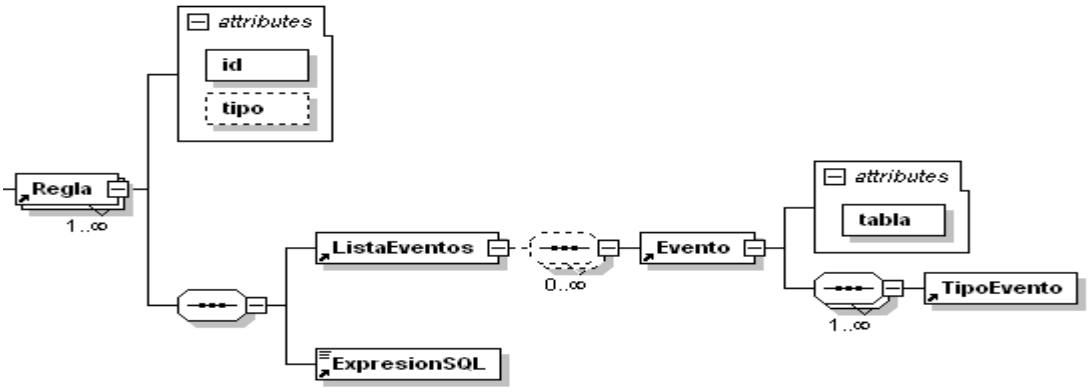
# Anexo 4 Esquema del repositorio primario de reglas




Esquema XSD del repositorio de reglas.




Parte del esquema XSD del repositorio de reglas referente a la representación formal de las reglas.




Parte del esquema XSD del repositorio de generación.

```
- <Regla id="RN#1" estado="inactiva">
    <lenguajeFormal />
    <lenguajeTecnico>Un Estudiante no puede tener
      sujeto.edad>35</lenguajeTecnico>
    <lenguajeNatural>Un estudiante no puede tener más de 35
      años.</lenguajeNatural>
  </Regla>
</Reglas>
```

Fragmento o sección del repositorio de reglas en XML.

```
- <Reglas schema="public">
  - <Regla id="RN#1" tipo="restriction" estado="inactiva" sujeto="estudiante"
      complejidad="compleja" gestor="postgres">
      <ExpresionSQL>(SELECT a."edad" FROM public."estudiante" a WHERE
        (sujeto."id_estudiante"=a."id_estudiante")) > 35</ExpresionSQL>
    - <ListaEventos>
      - <Evento tabla="estudiante">
        - <TipoEvento>
            <Tipo>INSERT</Tipo>
            <Tipo>UPDATE</Tipo>
          </TipoEvento>
        </Evento>
      </ListaEventos>
    </Regla>
  </Reglas>
```

Parte del repositorio de generación en XML.