Universidad Central "Marta Abreu" de Las Villas

Facultad Matemática, Física y Computación

Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

"Extensión del Traductor LPT-SQL"

AUTOR: Rolando Pérez Pedraza

TUTOR: M. Sc. Martha Beatriz Boggiano Castillo

SEMINARIO: Base de Datos

"Año 54 de la Revolución"

Junio 2012

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 Laboratorio.

_____

Fecha

Dedicatoria

A mis padres,

A mis Abuelos,

A mi hermana,

A mi novia,

A mis suegros

y a todos mis amigos.

Agradecimientos

A mi tutora Martha Beatriz Boggiano Castillo por guiarme en este trabajo.

A Ariel por asistirme cada vez que me encontraba algún obstáculo durante el trabajo.

A todo el grupo de Base de Datos.

A todos los que me formaron y enseñaron.

A mis padres y abuelos que hicieron todo cuanto estuvo a su alcance para que pudiese llegar aquí.

A mi novia por darme apoyo comprensión y aliento en los momentos difíciles.

A mi primo Dairon y a Antonio por su ayuda incondicional.

A Dunia por sus consejos

A mis amigos, que han estado presentes en las buenas y malas, siempre dispuestos a darme una mano.

Pensamiento

Las ciencias tienen las raíces amargas, pero muy dulces los frutos.

Aristóteles.

Resumen

El enfoque de reglas de negocio es utilizado para modelar e implementar las políticas del negocio. Para la implementación automática de las reglas de negocio en los sistemas de información se utilizan herramientas de software clasificadas como independientes de los gestores de datos. Dichas herramientas permiten la implementación de las reglas en las base de datos usando recursos de los gestores administrados por programas independientes del gestor.

En el laboratorio de bases de datos del Centro de Estudios Informáticos de la UCLV se desarrolla la herramienta LPT-SQL en el marco del tema “Generación automática de reglas de negocio en contextos relacionales”, esta herramienta traduce del Lenguaje de Patrones Técnicos desarrollado en este centro para representar las reglas “Cercanas a los Datos”, a recursos de bases de datos activas y hasta el momento solo brinda la implementación automática de las reglas de restricción, por lo que se hace necesario extender sus funcionalidades a los demás tipos de reglas.

En este trabajo se analizan las gramáticas de cada una de las reglas y se le aplican las modificaciones necesarias, se valoran las propuestas para la implementación de estas reglas como recursos de bases de datos, se modifican las estructuras de los repositorios para adaptarlos a las necesidades de las demás reglas y se adicionan los módulos necesarios para lograr una versión de la herramienta que soporte la implementación automática de las reglas.

Como resultado se obtiene la versión 2.1 del LPT-SQL que brinda la implementación automática de todas las reglas de negocio “Cercanas a los Datos”.

Abstract

The Business Rules Approach is used to model and implement business policies. Software tools classified as independent data managers are used for automatic deployment of business rules in Information Systems. These tools allow the implementation of the rules in the database using resources administered by programs independent of the manager.

In the Database Laboratory of the Center of Studies on Informatics of UCLV the LPT-SQL tool is developed under the theme "Automatic generation of business rules in relational contexts". This tool translates the Technical Pattern Language developed in this center to represent the rules "Near-Data" to active resources of the database and so far only provides automatic implementation of the restriction rules, so it is necessary to extend its functionality to other rule types.

In this paper we analyze the grammars of each one of these rules and we modify them if necessary. Besides, we evaluate some proposals for the implementation of these rules as database resources. We modify the repository structures to suit the needs of other rules and we add the necessary modules for a version of the tool that supports automatic implementation of the rules.

The result is version 2.1 of the LPT-SQL that provides automatic implementation of all RN "Near-Data."

Contenido

Ilustraciones

Tablas

Introducción

Los requisitos del negocio, enfocados como Reglas de Negocio (RN), son convertidos en fragmentos de código, que habitualmente han sido realizados “a mano” por el programador en el programa fuente, en los objetos de bases de datos de la aplicación, o en ambos (Calderón Solís, 2011). El enfoque de RN se entiende como una manera efectiva y aún novedosa de identificar los requerimientos del negocio, de modo que se haga más eficiente implementar los Sistemas de Información (SI) para el control y administración de los negocios (Calderón Solís, 2011).

Es una necesidad creciente en el mundo de la computación de los negocios, que los cambios en las políticas del negocio sean reflejados en sus SI de manera menos costosa y más eficiente, 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).

Se desarrolla un amplio campo de investigación sobre RN, dentro de las Ciencias de la Computación. Estas reglas deben ser formalizadas para poder diseñarlas e implementarlas en un SI (Boggiano Castillo et al., 2007).

Las RN han sido clasificadas por varios autores entre los que podemos citar a Ross, Solivares, Ashwell, Weiden y Morgan.

Uno de los temas de investigación que trabaja el seminario de Bases de Datos (BD) del Centro de Estudios Informáticos (CEI) es la generación automática de RN en BD relacionales. Entre los resultados obtenidos están los trabajos; “Aplicación de Reglas de Negocio” (Pérez Alonso, 2008). “Solución al problema de la cardinalidad en la generación automática de reglas de negocio” (Pereira Toledo, 2009), “Información del catálogo para generar reglas de negocio” y más reciente el trabajo “Traductor LPT-SQL para reglas de negocio en bases de datos relacionales” (Calderón Solís, 2011) que logra una herramienta capaz de generar e insertar RN de restricción para los gestores SQL Server y PostgreSQL escritas en Lenguaje de Patrones Técnicos (LPT) y se sientan las bases para la futura incorporación de otros tipos de reglas.

(Perez Alonso, 2010) aborda estas clasificaciones, propone una nueva clasificación de RN, que parte del diseño conceptual de la BD y formaliza un Lenguaje de Reglas Técnico (LRT) para su representación, a partir del diagrama entidad relación. El LRT fue enriquecido y transformado al LPT en (Calderón Solís, 2011) que

describe, sobre las tablas de la BD relacional, un lenguaje simple que permite la independencia del gestor de bases de datos para representar la regla.

Planteamiento del Problema

A partir de estudios del enfoque de reglas de negocio, como manera de agilizar la implementación de los sistemas de información, se han reconocido un conjunto de tipos o clases de reglas que pueden ser implementadas automáticamente como recursos de los gestores de BD relacionales.

Se ha desarrollado una herramienta LPT-SQL, para insertar las reglas de restricción, y a pesar de que en su diseño se consideraron las estructuras para la generación de las reglas de cómputo, de notificación y de clasificación no brinda la inserción automática de sus implementaciones.

Por esto se hace necesario completar la funcionalidad de la herramienta.

Objetivo general

Extender las funcionalidades del traductor LPT-SQL, para que implemente automáticamente los tipos de reglas de cálculo, clasificación y notificación, además de las reglas de restricción, utilizando los elementos básicos del diseño original del traductor.

Objetivo específicos

- Modificar la gramática de las reglas de cálculo, notificación y clasificación, en los casos que sea necesario.
- Analizar los recursos de BD definidos para las representaciones formales de estos tipos de reglas con vistas a realizar cambios en estas propuestas si es necesario.
- Adicionar los artefactos de las nuevas funcionalidades a los diagramas del diseño actual de la herramienta.
- Adicionar a la versión del traductor actual, los módulos necesarios que permitan reconocer e implementar automáticamente las RN de cálculo, clasificación y notificación.
- Modificar la estructura de los repositorios de acuerdo a las necesidades de las reglas a implementar.

Justificación de la Investigación

Este trabajo tiene alta incidencia en la investigación de generación de reglas de negocio en BD relacionales, pues con esta herramienta LPT-SQL se pretende integrar todas las funcionalidades que se necesitan para lograr la implementación automática de RN en BD. Es viable llevarla a cabo porque se cuenta con la base técnica y las personas necesarias para realizarla.

Preguntas de investigación

1. ¿Qué cambios hacer a la gramática para lograr la eficacia de cada una de las reglas?
2. ¿Qué recursos de BD son necesarios para la implementación formal de cada regla?
3. ¿Qué elementos se necesita añadir al diseño actual de la herramienta?
4. ¿Cómo lograr una versión de la herramienta que trabaje con reglas de clasificación, cálculo y notificación manteniendo la estructura actual de la herramienta?
5. ¿Qué modificaciones hacer a los repositorios en la implementación de estas reglas?

Hipótesis de investigación

Es necesario realizar transformaciones a la gramática del LPT, a la implementación de los recursos de BD y a los repositorios de reglas para extender las funcionalidades actuales del traductor LPT-SQL a los demás tipos de reglas “Cercanas a los Datos”.

Descripción de Capítulos

Este trabajo consta de tres capítulos:

En el Capítulo I se abordan diferentes definiciones y clasificaciones de RN formalizadas por varios autores y los distintos lenguajes para representarlas, así como los recursos de BD que se emplean para la implementación formal de las reglas.

En el Capítulo II se presenta la arquitectura general de la herramienta, se realiza un análisis detallado de la gramática y los patrones de las reglas que se agregan a la

herramienta, se muestran las clases que se agregan al diseño de la aplicación, se revisan y modifican los repositorios de Reglas y de Generación, y se exponen las cosideraciones generales que se tomaron en cuenta para la implementación de las reglas.

El Capítulo III trata la implementación en lenguaje formal de las reglas de cálculo, clasificación y notificación, se presenta el contenido de los repositorios para cada uno de los tipos de reglas que se implementan, también se exponen las características fundamentales de la herramienta.

Capítulo I:

CLASIFICACIONES DE REGLAS DE NEGOCIOS Y LENGUAJES PARA EXPRESARLAS.

CAPÍTULO 1: CLASIFICACIONES DE REGLAS DE NEGOCIOS Y LENGUAJES PARA EXPRESARLAS.

Este capítulo recoge definiciones de RN y clasificaciones de estas que formalizan autores como Ross y Matey, así como los diferentes lenguajes para representarlas. Se presenta además la clasificación de RN “Cercanas a los Datos” que se aborda en este trabajo.

1.1Definiciones

“En cierto modo, todo el mundo sabe qué son "las reglas de negocio", literalmente, lo que se utiliza para hacer funcionar un negocio. En sentido general, por supuesto, todo el mundo tiene razón. Las reglas de negocio son las que orientan un negocio en la gestión de las operaciones del día a día. Sin reglas de negocio, siempre se tienen que tomar decisiones sobre la marcha, elegir entre alternativas, caso por caso, con fines específicos. Hacer las cosas de esa manera sería muy lento. Es probable que se produzcan resultados tremendamente contradictorios. Dudo se pueda ganar la confianza de gran parte de sus clientes. En el mundo de hoy, realmente no se podría operar de esa manera, por lo menos no por mucho tiempo de todos modos. Así que cada proceso de negocio organizado tiene reglas de negocio” (Ross, 2000). Es importante entonces definir las RN formalmente para tener una visión más abarcadora de sus posibles desempeños y utilidades en el manejo de los negocios.

En trabajos precedentes (Pérez Alonso, 2008), (Perez Alonso, 2010) y (Calderón Solís, 2011), se ha descrito ¿qué son las RN? con base en los planteamientos de (Morgan, 2002), quien enuncia: “Básicamente, una regla de negocio es una declaración compacta sobre un aspecto del negocio… Es una restricción en el sentido que una regla de negocio decide lo que debe o no debe hacerse en cada caso. En cualquier punto particular, debe ser posible determinar que la condición que implicó la restricción es verdad en un sentido lógico; si no, tomar la acción necesaria.”

En (Scott W., 2011), se presenta una RN como aquello que define o limita un aspecto del negocio con el objeto de establecer una estructura o un grado de influencia que condiciona el comportamiento de los actores del negocio. A menudo las RN están focalizadas en el control, en la forma de realizar los cálculos, otras permiten establecer las políticas, y así se tienen en cualquier actividad del negocio, que requiera que la gente actúe de una forma pre-establecida.

El BRG (Business Rule Group, 2009) define una RN como “una directiva, destinada a influenciar o guiar el comportamiento del negocio, que apoya una política de negocio que ha sido formulada en respuesta a una oportunidad, amenaza, fortaleza o debilidad”.

En (Business Rules Group, 2003) se plantea que las reglas son restricciones explicitas de comportamiento y/o proporcionan soporte para la dirección de las actividades de negocio.

Otra definición concreta y concisa la da (Ross, 2000) al expresar "Las reglas de negocio" son literalmente el conocimiento codificado de sus operaciones comerciales.

Otras definiciones de autores como Lowenthal, Solivares y Morgan son tratadas en los trabajos (Pérez Alonso, 2008), (Perez Alonso, 2010).

1.2 Clasificaciones de Reglas de Negocio

En un esfuerzo por organizar el trabajo con las RN, cada autor las clasifica de acuerdo a su utilidad y es posible encontrarlas agrupadas en la medida que presenten comportamientos similares. 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, 2008) y las de Weiden, Ashwell y Solivares tratadas en (Perez Alonso, 2010), en la presente investigación se detectaron otras clasificaciones de autores como Ross, IoanaMatei y Coti Colop que se presentan a continuación:

1.2 .1 Clasificación de Ross

Ross define tres categorías de RN; estas son: Restricción, Producción y Proyección.

Estas categorías son intrínsecas, definitivas, y mutuamente excluyentes. Las reglas de producción se dividen a su vez en: regla de cómputo y regla de derivación; y las de proyección se subclasifcan en copia y posibilitador, cada uno de estos tipos contienen subtipos. A continuación se presentan las clasificaciones mencionadas:

Restricción

- **Restricción**
  - Cualquier regla que tiende a desaprobar o rechazar un evento si resultara una violación de la regla. Estas reglas escudan el negocio de datos incorrectos (estados incorrectos), o información que viola las reglas del negocio.

Por ejemplo, una restricción podría especificarse para impedirle a un cliente hacer un pedido a crédito si el cliente tiene una historia de pago pobre.

1. **Producción**

Cualquier regla que no rechaza ni proyecta los eventos, simplemente computa o deriva un valor basado en alguna función matemática.

- **Regla de cómputo (Cómputo)**
  - Cualquier regla tipo producción que compute un valor siguiendo operaciones aritméticas estándar (por ejemplo, suma, multiplicación, promedio entre otros) especificados explícitamente. Una regla de cómputo proporciona una fórmula precisa para cómo un término computado será calculado.

    Por ejemplo, una regla de cómputo podría darse para calcular el volumen del orden anual de un cliente.
- **Regla de derivación (Derivación)**
  - Cualquier regla tipo-producción que deriva un valor verdadero (es decir, TRUE o FALSE) basado en operadores lógicos (por ejemplo, AND, OR, NOT, EQUAL TO, y así sucesivamente) especificados explícitamente. Una regla de derivación mantiene una definición precisa de qué es un término derivado, cuyo valor siempre se establece por los operadores lógicos especificados.

    Por ejemplo, una regla de la derivación podría darse para indicar si un proyecto está en riesgo dependiendo de si está por encima del presupuesto o escaso de personal.

2. **Proyección (Regla de estímulo/repuesta)**

Cualquier regla que tiende a tomar alguna acción ante la ocurrencia de un evento. Un proyector nunca rechaza los eventos (como la restricción); más bien, los proyecta, es decir, causa algún nuevo evento para dar lugar al resultado. Los proyectores generalmente prescriben el comportamiento automático del sistema, mientras estimulan la productividad.

- **Posibilitador (Conmutador)**

Un proyector que puede accionar o detener algo

- **Regla de inferencia**

  Un Posibilitador que infiere algo que es verdad bajo circunstancias apropiadas.

  Por ejemplo, una regla de la inferencia podría darse para indicar que una persona debe ser considerada una mujer si el criterio para la edad y el género está satisfecho.

  - **Regla de conmutación (Regla tipo-excepción)**

    Un posibilitador que es capaz de habilitar o inhabilitar otra regla bajo las circunstancias apropiadas; esto la hace capaz o incapaz de dispararse.

    Por ejemplo, una regla (toggle) podría darse para indicar que alguna regla que opera normal será suspendida bajo circunstancias de emergencia.

  - **Conmutador de proceso**

    Un posibilitador que puede ejecutar o detener operaciones, procesos o procedimientos bajo las circunstancias apropiadas; esto las hace capaces o incapaz de ejecutarse.

    Por ejemplo, conmutador para indicar que un proceso sensible no puede ejecutarse mientras exista una brecha de seguridad sospechosa.

  - **Conmutador de datos**

    Por ejemplo, un conmutador puede indicar que el registro delictivo de un joven debe borrarse cuando él o ella alcanzan 18 años de edad.

- **Copia**

Un proyector que reproduce (copia) valores actuales.

- **Regla impresión**

  Un copiador que pone el valor de algo que persiste (por ejemplo, algo en una BD).

  Por ejemplo, una regla de impresión podría usarse para inicializar la matrícula de un estudiante en un semestre dado al matricular en ese semestre, cuando el estudiante se inscribe.

- **Regla de presentación**

Un copiador que establece un valor o parámetro en relación a cómo serán presentados los datos (por ejemplo, en una pantalla, en un informe, etc.).

Por ejemplo, una regla de presentación podría darse para indicar que una orden será mostrada en la pantalla en la red si la orden se ha retrasado.

- **Administrativa (Trigger)**

  Un proyector que causa la ejecución de una operación, proceso, procedimiento o el disparo de una regla.

  - **Trigger de proceso**

    Un proyector que causa que se ejecute una operación, proceso, o procedimiento.

    Por ejemplo, cuando una orden se envía, un trigger podría darse para ejecutar un proceso que automáticamente le envía una notificación al destinatario intencional.

  - **Trigger de regla**

    Un proyector que causa que se dispare una regla

    Por ejemplo, cuando los datos sobre un embarque se muestran en la pantalla, un trigger de regla podría propiciar que se dispare otra regla para predecir la fecha de la llegada del embarque.

1.2.2 Clasificación de Iona Matei

Hay varias clasificaciones de reglas de negocios, la mayoría de estas dependen de la metodología para la implementación de estas reglas en los sistemas de información (Matei, 2006).

El Grupo de Reglas de Negocio adopta en (Halle, 2002) esta clasificación, que asume Ioana Matei para acomodar el alcance de la implementación de un agente de software. Esta clasificación de regla es lo suficientemente simple, no obstante capaz para estandarizar todos los tipos de reglas. En la Figura 1 se muestra esta clasificación.

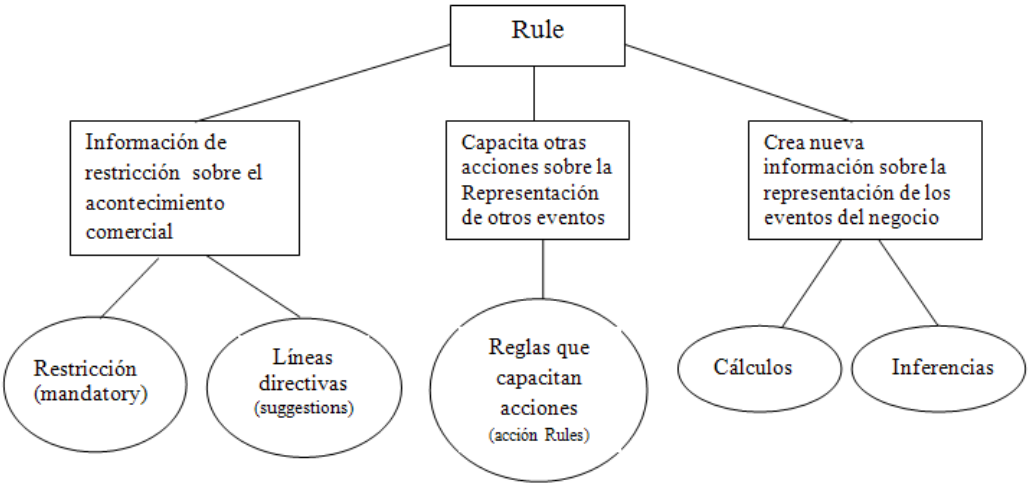


Figura 1: Esquema de clasificación de reglas (Halle, 2002).

Restricciones

Una restricción obligatoria es una sentencia completa que expresa una circunstancia incondicional que debe ser verdadera o no para los eventos del negocio para completar la integridad (Halle, 2002).

Ejemplos:

- Una máquina debe correr sólo un trabajo a la vez.
- Un cliente debe tener 10 % del valor de la propiedad para pedir un préstamo de la casa.

Líneas directivas

Una línea directiva es una restricción suave. Es una declaración completa que expresa un aviso acerca de una circunstancia que no debería ser verdadera o falsa (Halle, 2002) porque la línea directiva es sólo una advertencia, deja al humano hacer la decisión de denegarlo.

Ejemplos:

- Una máquina debería maximizar el número de trabajos para los que se sirvió.
- Un cliente debería tener sólo un préstamo de la casa a la vez.

Posibilitadores de Acción

Una regla posibilitadora de acción es una declaración completa que prueba condiciones al encontrar iniciado en verdadero otro acontecimiento comercial,

mensaje u otras actividades. Un posibilitador de acción inicia una nueva acción externa (Halle, 2002).

Ejemplos:

- Si un trabajo está próximo a comenzar, entonces inicie negociación con máquina.
- Si una petición de préstamo del cliente es válida entonces se adopta un método de préstamo.

Las computaciones o los cálculos

Una computación es una declaración completa que provee un algoritmo para lograr el valor de un término donde tales algoritmos pueden incluir suma, diferencia, producto, cociente, cuenta, mínimo, máximos y promedios (Halle, 2002). El resultado de la computación es un valor nuevo para un atributo.

Ejemplos:

- La tasa mensual de préstamo es computada como (la tasa anual de préstamo / 12months).

Inferencias

Una inferencia es una declaración completa que el *staf* de prueba pone en forma y al encontrarlos verdadero establece la verdad de un hecho nuevo (Halle, 2002). El resultado de una inferencia puede ser una instancia nueva de una entidad o una instancia nueva de un atributo.

Ejemplos:

- Si el cliente no tiene tardanza en los pagos entonces el estatus del cliente es preferido.

1.2.3 Clasificación de Benito Manolo Coti Colop

Según (Coti Colop, 2003), cada regla del negocio debe ser uno de los siguientes:

Structural Assertion: Es una definición conceptual o un argumento de un hecho que exprese algunos aspectos de la estructura de una empresa. Este abarca términos y los hechos de estos términos.

Action Assertion: Un argumento de una restricción o condición que limite o controle las acciones de la empresa.

Derivation: Un argumento de conocimiento que es derivado en otro conocimiento en el negocio.

La siguiente figura muestra las partes del modelo de las reglas del negocio reflejando estas ideas.

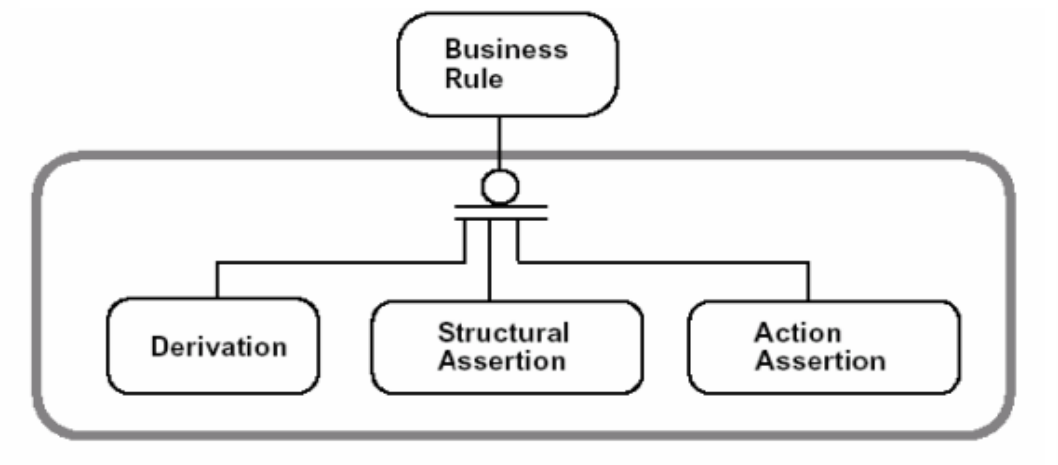


Figura 2: Tipos de RN.

Luego establece una clasificación de los tipos en varios grupos.

1. **Reglas del modelo de datos**

   Aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto es válida.

   Ejemplo:

   No hay precios de artículos negativos, el sexo de una persona solo puede ser masculino o femenino.

2. **Reglas de relación**

   Aquellas reglas que controlan las relaciones entre los datos.

   Ejemplo:

   Todo pedido debe ser realizado por un cliente, y el mismo debe estar dado de alta en el sistema: además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos.

3. **Reglas de derivación**

   Es el conjunto de reglas que especifican y controlan la obtención de información que se puede calcular a partir de la ya existente.

   Ejemplo:

El total de un pedido se puede calcular a partir de las distintas líneas que lo componen, mientras que el total de cada línea se puede calcular a partir del número de unidades vendidas y el precio por unidad.

4. Reglas de restricción

Reglas que restringen los datos que el sistema puede contener. Este grupo de reglas se solapa en cierto modo con las reglas del modelo de datos, dado que aquellas también impiden la introducción de datos erróneos, como se vio anteriormente.

Ejemplo:

El saldo nunca puede ser menor que cierta cantidad tope establecida para cierto tipo de clientes.

5. Reglas de flujo

Aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema.

Ejemplo:

Un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado: hecho esto, se genera una solicitud para uno o más analistas, estos realizan las mediciones correspondientes y devuelven los reportes con la información pertinente, a partir de la cual se genera un informe de análisis, que será un análisis válido solo cuando sea firmado por los responsables de garantizar su corrección.

1.2.4 Clasificación en que se fundamenta esta investigación: reglas "Cercanas a los Datos"

Esta clasificación es el fundamento de las investigación "Generación automática de reglas de negocio en bases de datos relacionales", tiene como precedente el trabajo "Reglas de Negocio en Bases de Datos Relacionales" (Alonso, 2010) que están inspiradas en la clasificación de RN que da Tony Morgan en (Morgan, 2002), donde expresa que la forma más conveniente de crear sentencias de regla es seleccionar patrones adecuados desde una pequeña lista disponible. Esta clasificación fue modificada posteriormente en el trabajo de (Solís, 2011), en este trabajo se formaliza el concepto de RN "Cercanas a los Datos", como aquellas reglas que están involucradas en las operaciones sobre la base de datos del negocio, y presenta la siguiente clasificación en forma de patrones de reglas:

Patrón de Restricción

Obliga a que se cumplan los requisitos del negocio, contribuyendo a preservar la integridad del mismo.

Ejemplo:

Un estudiante no puede tener una nota mayor que 5 en un Trabajo de Control.

Patrón de cómputo

Su objetivo es calcular un valor determinado en el negocio, y su resultado es numérico. Este patrón comparte grandes semejanzas con el patrón de clasificación (Ross, 2010).

Ejemplo:

El promedio de un estudiante es calculado como el promedio de las notas de todas sus asignaturas.

Patrón de clasificación

Organiza el conocimiento básico del negocio contribuyendo claramente al significado de conceptos, centrándose en la esencia del mismo (Ross, 2010), este patrón es conocido también como una regla de definición.

Ejemplo:

Un estudiante con índice académico mayor que 4 es definido como estudiante de alto aprovechamiento.

Patrón de notificación

Este patrón informa a los usuarios autorizados del negocio sobre algún conocimiento básico en tiempo real, no restringe estados del negocio, solo lo informa para que se tomen las medidas pertinentes.

Ejemplo:

Notificar “Estudiante sin nota” si no tiene nota en alguna asignatura.

En (Calderón Solís, 2011) se estudian otras clasificaciones y se hallan puntos en común entre ellas tomando como base la de Morgan, de forma similar, en las clasificaciones presentadas en este trabajo se hallan las características que son comunes a la clasificación de Reglas cercanas a los datos que se utiliza en la versión precedente del Traductor LPT-SQL y que se sigue empleando para esta versión:

- **Restricción:** es común en todas estas clasificaciones.

En todos los casos esta regla restringe los datos contenidos en el sistema.

- **Cómputo:** Se da en todas las clasificaciones que se presentaron, en la de Coti Colop con el nombre Reglas de Derivación e igual funcionalidad aunque un poco más general.

En esencia la función de esta regla es el cálculo de un resultado mediante un determinado algoritmo.

- **Notificación:** Es común en las clasificaciones de Ross, en la Cercana a los Datos y Matei con el nombre de líneas directivas.

Su objetivo es informar de la ocurrencia de algún hecho a los usuarios pertinentes.

- **Clasificación:** Es común en la de Ross y en la cercana a los datos. La inferencia que trata Matei concuerda con la clasificación, aunque en la clasificación cercana a los datos no se ha trabajado hasta el momento con la adición de instancias o valores de atributos.

Este tipo de regla permite definir términos del negocio en función de otros que cumplen determinadas características, en función del valor de algunos de sus atributos.

De acuerdo con Morgan (Morgan, 2002) las reglas de negocio pueden expresarse de diversas formas, principalmente, según su especificación en el desarrollo de un sistema de información o de acuerdo a la manera en que se introduzcan a este.

Según expone (Calderón Solís, 2011), se destacan tres niveles de expresión de las RN basadas (Morgan, 2002):

- **Informal:** este nivel proporciona una sentencia en lenguaje natural, sin un rango limitado de parámetros, tal y como el cliente del negocio desee.
- **Técnico:** este nivel combina referencias a datos estructurados, operadores y restricciones con el lenguaje natural, nivel intermedio entre la entrada de la regla y su implementación.
- **Formal:** este nivel proporciona sentencias conforme a una sintaxis definida y proporciona la funcionalidad automática de la regla.

Durante la confección de una regla es necesario pasar por cada uno de estos niveles; desde el informal en el momento en que se detecta una posible regla por parte de algún analista del negocio, se pasa al técnico, nivel donde se logra la

representación matemática de la regla y se facilita la traducción al formal, en nuestro caso a recursos de bases de datos.

1.3 Lenguajes de especificación de reglas de negocio

Con el propósito de representar las RN se han creado diversos lenguajes. Estos tratan de establecer un punto medio entre el lenguaje formal y el natural, de modo que se facilite la comprensión a todos los involucrados en el negocio y la implementación de la regla para el diseñador; además de establecer un estándar para la comunicación entre estas dos partes.

1.3.1 RuleSpeak

RuleSpeak® es un conjunto de pautas para expresar las reglas comerciales de negocio de forma concisa que facilitan la redacción y comprensión de la regla para cada una de las partes involucradas en el negocio; es desarrollado por Ross, considerado el padre de las RN. RuleSpeak y es una de tres notaciones de referencia usada en la creación del *Semantics for Business Vocabulary and Business Rules* (SBVR) publicado por el *Object Management Group* (OMG) en 2007. Es totalmente consistente con ese estándar.

- Los símbolos <> indican que el artículo sintáctico dentro es obligatorio.
- Los símbolos [] indique el artículo sintáctico dentro es opcional.
- El símbolo / indica que sólo un artículo sintáctico de lista puede ser seleccionado.
- La condición siempre involucra una expresión lógica (algo que siempre debe ser verdad o falso y puede incluir a los operadores lógicos como AND, OR y NOT.
- El hecho se refiere al resto de la declaración después del sujeto.

  Se considera que puede catalogarse como un lenguaje informal o natural en forma de patrones.
- El <sujeto> es el primer término (o hecho) incluido en la declaración de una regla, indica sobre que es la regla.

1. Restricción

<sujeto> [no] tiene/debe/puede <hecho> [si/mientras <condición>].

<sujeto> no necesita <hecho> [si/mientras <condición>].

<sujeto> puede/debe <hecho> solo si/mientras <condición>.

<sujeto> puede/debe <hecho> solo <preposición><condición>.

2. Cómputo

<sujeto> [no] tiene/debe ser calculado como <fórmula matemática> [si/mientras <condición>].

<sujeto> = <fórmula matemática> [si/mientras <condición>].

3. Derivación

<sujeto> [no] tiene/debe tomar para significar <expresión lógica> [si/mientras <condición>].

<sujeto> [no] significa <expresión lógica> [si/mientras < condición >].

4. Inferencia

<sujeto> [no] tiene/debe ser considerado [un] <término> si/mientras <condición>.

<sujeto> [no] es [un] <término> si/mientras <condición>.

5. Conmutación

<sentencia de la regla>, a menos que/excepto <condición>.

<nombre de la regla> [no] tiene/debe cumplir si/mientras <condición>.

6. Conmutación de procesos

<sujeto> [no] tiene/debe ser habilitado/deshabilitado si/mientras <condición>.

7. **Conmutación de datos**

<objeto> [no] tiene/debe ser creado/eliminado si/mientras <condición>.

8. Impresión

<término> [no] tiene/debe ser colocado en <término/valor> [cuando/si <condición>].

9. Presentación

<sujeto> [no] tiene/debe ser mostrado [hacia/sobre/dentro <medios>] <forma de mostrar> [si/mientras <condición>].

10. Trigger de proceso

<sujeto> tiene/debe ser ejecutado cuando <condición>.

11. Trigger de regla

<nombre de la regla> tiene/debe ser disparado cuando <condición>.

1.3.2 LPT

El LPT no es más que una expresión matemática de los elementos que conforman los patrones de reglas cercanas a los datos vistos anteriormente, para ello consta de una notación específica, operadores aritméticos, lógicos y funciones que en conjunto conforman un formalismo capaz de representar reglas de negocio. Su

núcleo es la notación punto para la navegación entre las entidades y el acceso a sus atributos, lo cual según se refiere en (Zimbrão et al., 2002), le brinda consistencia. Este lenguaje ha sido conformado en el marco del tema de investigación "Generación automática de reglas de negocio en contexto relacional" que se desarrolla en el CEI de nuestra universidad.

Las características fundamentales del LPT son:

1. Es un lenguaje para expresar reglas de negocio, en una BD relacional siendo los elementos de las reglas tablas, atributos o interrelaciones entre las tablas.
2. Para referirse a un camino de navegación entre tablas interrelacionadas directamente o a atributos de una tabla se usa la navegación punto.
3. Es un lenguaje sencillo que define un conjunto de operadores aritméticos y lógicos para establecer comparaciones y operaciones con las tablas y atributos de la base de datos, involucrados en la regla de negocio enunciada.

Convenios utilizados para precisar elementos variables de los patrones:

- Paréntesis: ( ) Encierran un grupo de ítems.
- Corchetes: [ ] Encierran elementos opcionales.
- Barra Vertical: | Separa términos alternativos.
- Corchetes Angulares:<> encierran términos especiales como los mostrados en la siguiente tabla.

Elemento	**Significado**
<determinante>	Es el determinante para cada sujeto, por ejemplo: Una, Uno, El, La, Cada, Todos. Según el mejor sentido en la redacción.
<sujeto>	Es una entidad en la BD del negocio o una clasificación de la misma.
<hecho>	Hechos relativos al estado o comportamiento de la BD del negocio, incluyendo o no al sujeto.
<característica>	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>	Cualquier valor, no necesariamente numérico, que tiene algún significado en el negocio. El resultado es usualmente el valor del atributo de un objeto del negocio.
<algoritmo>	Definición de una expresión matemática para obtener el valor de un resultado; normalmente expresada utilizando combinaciones de términos del negocio junto a constantes disponibles.
<clasificación>	Definición de un término del negocio. Típicamente define el valor de un atributo o un subconjunto de objetos en una clase existente.
<mensaje>	Mensaje de información entre comillas para usuarios autorizados del negocio.

Tabla 1: Elementos de variables de los patrones.

Patrones de las reglas:

1. **Restricción**

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

2. **Cómputo**

<determinante><resultado> [en <sujeto>] es calculado como <algoritmo>.

3. **Clasificación**

<determinante><sujeto> [no] es definido como <clasificación>

[(si | a menos que)<característica>].

4. **Notificación**

Notificar <mensaje> si <hecho>.

Para la sintaxis y semántica del LPT ver Anexos 1 y 2.

1.3.3 Otros lenguajes de especificación de reglas de negocio

En esta investigación se revisaron otros lenguajes como RuleML, *Semantic Web Rules Language* (SWRL), *Semantic Business Vocabulary and Rules* (SBVR) y *Rules Interchange Format* (RIF) de los cuales se hablará de forma general, otro de los lenguajes altamente reconocidos es el *Object Constraint Language* (OCL), del cual se trata en (Calderón Solís, 2011) y se establece una comparación con LPT.

1.3.3.1 RuleML

RuleML es un lenguaje de marcado para la publicación y acceso compartido a bases de reglas en web. Se ha convertido en un estándar de facto para el intercambio de reglas de negocio, por lo que ha servido de base para la definición de los nuevos estándares desarrollados por el W3C, como RIF. El lenguaje RuleML es en realidad un árbol de sublenguajes (que tienen como base XML, RDF, XSLT y OWL), cuya raíz permite utilizar el lenguaje como un todo y cuyos nodos permiten identificar subconjuntos adaptados del lenguaje. El sublenguaje fundamental en RuleML se llama Kernel Datalog (Martínez Fernández, 2010).

1.3.3.2 Semantic Web Rules Language

Semantic Web Rule Language (SWRL) fue definido por *World Wide Web Consortium* (W3C) para definir reglas sobre la Web Semántica, está basado en una combinación de *Web Ontology Language* (OWL) DL y OWL Lite, sublenguajes de OWL con Datalog y RuleML sublanguages de *Rule Markup Language*. SWRL incluye una sintaxis abstracta de alto nivel para reglas *Horn-like* en OWL DL y OWL Lite sublenguages de OWL, además de un modelo teórico semántico para mantener el significado formal de las ontologías de OWL, incluso para las reglas escritas en esta sintaxis abstracta. Una sintaxis de XML basada en RuleML y OWL XML *Presentation Syntax* así como sintaxis concreta RDF basada en OWL RDF/XML (Ian Horrocks et al., 2004)

1.3.3.3 Semantic Business Vocabulary and Rules (SBVR)

SBVR es una especificación disponible públicamente en el *Object Management Group* (OMG), pretende ser la base de una descripción formal y declarativa de un lenguaje natural detallado en una entidad compleja, tal como un negocio. SBVR tiene la intención de formalizar la complejidad de las normas de cumplimiento, tales como normas de funcionamiento para una empresa, la política de seguridad, el cumplimiento de normas o reglas de cumplimiento normativo. Estos vocabularios formales y las reglas pueden ser interpretados y utilizados por los sistemas informáticos. SBVR permite la producción de vocabulario y reglas de negocio, el vocabulario más reglas constituyen un modelo de dominio compartido con la misma potencia expresiva de las lenguas estándar ontológicas. SBVR permite hacer accesibles las reglas de negocio a las herramientas de software, incluyendo herramientas que apoyan los expertos en negocios en su creación, la búsqueda, validación y gestión de reglas de negocio, y herramientas que apoyan los expertos

en tecnología de la información en la conversión de las reglas de negocio en las normas de aplicación para los sistemas automatizados (Analytics, 2004).

1.3.3.4 Rules Interchange Format (RIF)

La *World Wide Web Consortium* (W3C) ha publicado este estándar para la construcción de sistemas de reglas en la Web. Reglas declarativas permiten la integración y transformación de datos de múltiples fuentes en una forma distribuida, transparente y escalable. La nueva norma, denominada RIF, se desarrolló con la participación de las Reglas de Negocio, la programación lógica y las comunidades de la Web Semántica para proporcionar la interoperabilidad y la portabilidad entre diferentes lenguajes de reglas y motores de reglas.

RIF cierra la brecha entre las reglas de la lógica Web (semántica), apoya a la integración de datos y reglas de negocio reactivos, para que las organizaciones sean más ágiles. La nueva norma incluye un formato concreto de serialización XML para el modelo basado en lenguajes como OMG PRR, SBVR OMG, y la subfamilia RIF RuleML de la RuleML.

1.4 Recursos de bases de datos para la implementación de reglas de negocio

El estándar de SQL ofrece disímiles recursos para el manejo de datos, en la implementación para SQL Server 2005 (MSDN, 2008). En (Perez Alonso, 2010) se presentan los posibles recursos a emplear en la implementación de las reglas, de los cuales, para la implementación de las reglas en esta investigación se tomaron los siguientes:

Disparadores (Triggers)

Un disparador es una pieza de código ejecutable, que consiste en instrucciones declarativas y procedurales y que se almacenan en el catálogo del Sistema de gestión de bases de datos (SGBD)(González, 2010). Son tipos especiales de procedimientos almacenados para la ejecución automática al emitirse una instrucción UPDATE, INSERT o DELETE en una tabla o una vista. Constituyen herramientas eficaces que pueden utilizar los sitios para exigir automáticamente las reglas comerciales cuando se modifican los datos. Amplían la lógica de comprobación de integridad, valores predeterminados y reglas del estándar SQL, aunque se deben utilizar las restricciones y los valores predeterminados siempre que estos aporten toda la funcionalidad necesaria (Melton and Simon, 2002)

Los desencadenadores pueden consultar las tablas y usar instrucciones SQL complejas. Son muy útiles para exigir reglas o requisitos complejos; también para exigir la integridad referencial, que conserva las relaciones definidas entre tablas (MSDN, 2008)

Según se plantea en (Healy, 2000) utilizar desencadenadores en un manejador de base de datos puede traer como resultado:

- *Rapidez de desarrollo de las aplicaciones :*

  Porque al estar los disparadores almacenados en las bases de datos relacionales, las acciones llevadas a cabo por ellos no tienen que ser codificadas en cada aplicación.

- *Ejecución global de las reglas de negocio:*

  Un disparador sólo tiene que ser definido una vez, y entonces puede ser usado por cualquier aplicación que modifique la tabla.

- *Facilidad de mantenimiento*

  Si una política de negocio cambia, sólo el disparador correspondiente necesita cambiar en lugar de cada programa de la aplicación.

Vistas (View)

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Suelen utilizarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario (Melton and Simon, 2002).

Las vistas se pueden utilizar para realizar particiones de datos y para mejorar el rendimiento cuando estos se copian. Además, permiten a los usuarios centrarse en datos de su interés y en tareas específicas de las que son responsables. Los datos innecesarios pueden quedar fuera de la vista; de ese modo, también es mayor su seguridad, dado que los usuarios solo pueden ver los definidos en la vista y no los que hay en la tabla subyacente (MSDN, 2008).

Algunas de las de las ventajas de utilizar las vistas que se encuentran en (Date, 2000) son:

- Las vistas proporcionan seguridad automática para los datos ocultos.

  “Datos ocultos” se refiere a los datos que no son visibles a través de alguna vista determinada. Existe la seguridad de que estos datos no serán accedidos

(por lo menos, en un acceso de recuperación) a través de esa vista en particular. Por lo tanto, obligar a los usuarios a acceder a la BD a través de una vista constituye un mecanismo de seguridad simple pero efectivo.

- Las vistas ofrecen una posibilidad de forma abreviada o “macro”.

Las vistas en un sistema de BD juegan un papel en cierta forma similar al de las macros en un sistema de lenguaje de programación; las ventajas y beneficios de las macros también se aplican, haciendo los cambios necesarios, directamente a las vistas. El uso de las vistas no implica ninguna sobrecarga adicional del rendimiento en tiempo de ejecución; solo hay una pequeña sobrecarga en tiempo de procesamiento de la vista.

- Las vistan pueden ofrecer la independencia lógica de los datos.

La independencia lógica de los datos puede definirse como la inmunidad de los usuarios y los programas de usuario ante los cambios en la estructura lógica de la BD, y las vistas son el medio por el cual se logra la independencia lógica de los datos en un sistema relacional.

Funciones definidas por el usuario

Al igual que las funciones en los lenguajes de programación, las funciones definidas por el usuario de Microsoft SQL Server 2005 (MSDN, 2008) son rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor. Son múltiples las ventajas de las funciones definidas por el usuario entre las cuales se tienen:

- Permiten una programación modular.
- Permiten una ejecución más rápida.
- Pueden reducir el tráfico de red.

Procedimientos Almacenados

Los procedimientos almacenados pueden facilitar en gran medida la administración de la base de datos y la visualización de información sobre esta y sus usuarios. Son una colección precompilada de instrucciones SQL e instrucciones de control de flujo opcionales, almacenadas bajo un solo nombre y procesadas como una unidad. Estos se guardan en una base de datos, permitiendo ser ejecutados desde una aplicación.

Los procedimientos almacenados pueden contener flujo de programas, lógica y consultas a la base de datos; aceptar parámetros y proporcionar sus resultados, devolver conjuntos individuales o múltiples y retornar valores.

Se pueden utilizar procedimientos almacenados para cualquier finalidad que requiera la utilización de instrucciones SQL, con estas ventajas:

- Ejecución de una serie de instrucciones SQL en un único procedimiento almacenado.
- Referenciar a otros procedimientos almacenados desde el propio procedimiento almacenado, con lo que se puede simplificar una serie de instrucciones complejas.
- El procedimiento almacenado se compila en el servidor cuando se crea, por tanto, se ejecuta con mayor rapidez que las instrucciones SQL individuales.(MSDN, 2008).

1.5 Conclusiones Parciales

En este capítulo se caracterizaron otras clasificaciones de RN que no se habían abordado en trabajos anteriores, se presentaron los diversos lenguajes que se utilizan en la actualidad para la representación de los Requisitos del Negocio y los recursos de BD para la implementarlas. Se profundizó en la clasificación de Reglas cercanas a los datos, producto de la investigación de este tema en nuestro centro y en el LPT, lenguaje que se define para su representación matemática y punto de partida para lograr el objetivo principal de este trabajo.

Capítulo II:

EXTENSIÓN DEL LPT.

CAPÍTULO 2: EXTENSIÓN DEL LPT.

El LPT, presentado con descripción sintáctica y semántica en (Calderón Solís, 2011) ha sido utilizado para expresar patrones en el nivel técnico, lográndose una especificación analizada en detalle para el patrón de restricción.

Para expresar los demás patrones de reglas cercanas a los datos, con LPT, se trabaja en analizar cada tipo de regla aún no implementada con el propósito de mejorar el nivel de expresividad de este y en caso necesario extender el LPT. Se logran así las extensiones que se necesiten para que este leguaje exprese las especificaciones de todos los patrones de reglas, presentados aquí, así como operadores y funciones necesarios.

En este capítulo, después de presentar de forma general el proceso de generación del Traductor, se analizan los patrones y las gramáticas de las reglas que se implementan, se presentan las clases que se añaden a la aplicación, se analiza la necesidad de modificar los repositorios y se exponen las consideraciones generales que se toman en cuenta para la implementación de las reglas de cómputo, clasificación y notificación.

2.1 Arquitectura general de la herramienta

Un esquema general del proceso de generación de las reglas se muestra en la Figura 3, donde la entrada del traductor es la regla en LPT que se toma del repositorio de reglas. Durante la traducción se consulta la información del catálogo que está almacenada en el repositorio del catálogo. Este repositorio contiene la información de las tablas, atributos, disparadores, 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 (Calderón Solís, 2011).

Para las reglas tipo restricción, esta información consiste en la consulta base en SQL y la lista de eventos, los eventos que posibilitan un cambio de estado relacionado con cada regla (Calderón Solís, 2011).

En este trabajo se trata la regla de notificación de forma similar a la de restricción, por tanto se guarda consulta base en SQL y la lista de eventos.

Para la regla de tipo cálculo se almacena una consulta base en SQL y la Lista de Tablas en las que es necesario implementar los disparadores.

Para la regla de clasificación se guarda la consulta SQL y la <clasificación> a manera de función.

Finalmente se genera la regla a partir de la información extraída del repositorio de generación 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 (Calderón Solís, 2011).

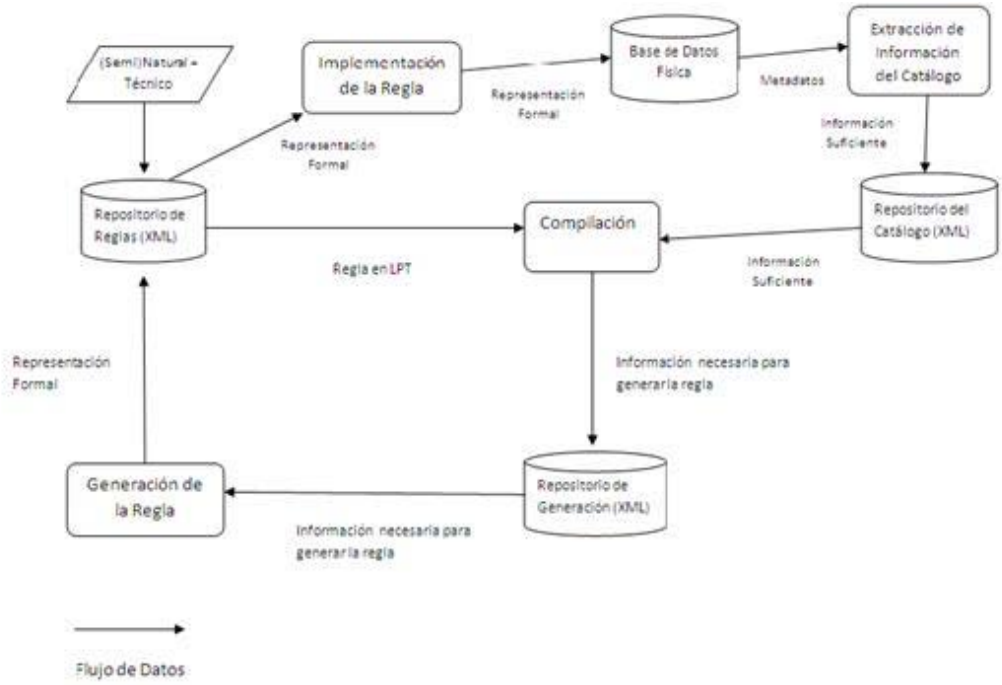


Figura 3: Diseño general del proceso de traducción.

2.2 Análisis de la gramática y los patrones de reglas

De acuerdo a (Hernández, 2007), 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 esa componente (Calderón Solís, 2011).

En la implementación del LPT-SQL, (Calderón Solís, 2011) utiliza la construcción de un ASA como salida intermedia del proceso de traducción. De esta forma se reconocen y generan sentencias navegando por una estructura de árbol, desde el concepto más abstracto (raíz del árbol) hasta los símbolos del vocabulario (hojas del árbol). Cada subárbol representa una fase de una sentencia y se hace corresponder directamente con una regla de la gramática. A cada regla que

contenga terminales que aporten algún valor semántico se le hace corresponder un nodo del ASA.

Para la construcción del traductor fue utilizada la herramienta ANTLR v3, un generador de analizadores sintácticos que automatiza la construcción de reconocedores de lenguajes.

Al definir la gramática en el ANTLR, los elementos a la derecha del operador -> indican la estructura del árbol que se desea construir. El primer elemento dentro de la especificación ^ (…) representa la raíz del árbol. Los demás elementos representan hijos de esa raíz (ver Figura 4).

```
stat:   expr NEWLINE         -> expr
    |   ID '=' expr NEWLINE -> ^('=' ID expr)
    |   NEWLINE              ->
    ;
```

Figura 4: Definición de una regla y de la estructura del ASA resultante.

2.2.1 Análisis del patrón de regla de cómputo

Este patrón es común en la mayoría de las clasificaciones estudiadas. En LPT se concibió como se muestra a continuación:

<determinante><resultado> [en <sujeto>] es calculado como <algoritmo>.

A continuación se muestra la gramática de esta regla, donde se puede apreciar los nodos que le aportan valor semántico. Para esta regla es necesario definir el nodo raíz TCOMPUTO, ya que los demás nodos del ASA que conforman el cuerpo de la regla están implementados y son comunes para cualquier otra regla del LPT.

```
computo
     : TDETERMINANTE resultado TES_CALCULADO algoritmo->
^(TCOMPUTO resultado algoritmo)
             | TDETERMINANTE resultado TEN sujeto
TES_CALCULADO algoritmo->^(TCOMPUTO resultado algoritmo sujeto
);
```

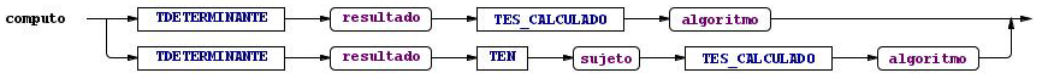


Figura 5: Diagrama de sintaxis del patrón de cálculo original.

En la Figura 5 se muestra su diagrama de sintaxis. En esta investigación se hace necesario hacer transformaciones a este patrón para lograr abarcar las posibles variantes que se pudieran formular para la regla, puesto que no se consideró la posibilidad de que el resultado del cálculo fuese un atributo de la BD, se le agrega entonces al patrón la cláusula [para <atributo>] que indica que el <resultado> se calcula para un <atributo> de la tabla <sujeto>; a continuación se presenta el patrón de la regla modificado.

<determinante><resultado> [en <sujeto> [para <atributo>]] es calculado como <algoritmo>.

A la gramática se le agrega esta variante incluyendo el nodo <atributo> que aporta semántica al nodo TCOMPUTO y es definido por los nodos que posee la versión anterior. A continuación se presenta la gramática de la regla modificada y su diagrama de sintaxis en la Figura 6.

```
computo
      :     TDETERMINANTE resultado TES_CALCULADO algoritmo->
^(TCOMPUTO resultado algoritmo)
              | TDETERMINANTE resultado TEN sujeto
TES_CALCULADO algoritmo->^(TCOMPUTO resultado algoritmo sujeto
)
              | TDETERMINANTE resultado TEN sujeto TPARA
atributo TES_CALCULADO algoritmo->^(TCOMPUTO resultado
atributo sujeto algoritmo);
```

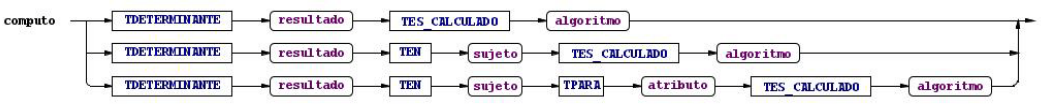


Figura 6: Diagrama de sintaxis del patrón de cálculo modificado.

La propuesta inicial de la implementación formal de esta regla consistía en una función que realizara el cómputo de un resultado, al incorporar la posibilidad de que el resultado se almacenara en la BD se detectaron dos variantes para la generación de esta regla:

1. El resultado no es un atributo de la BD.
    1.1 Las expresiones del algoritmo involucran a todas las instancias de una tabla.

Ejemplo:

La matrícula es calculado como sizeof(estudiante.idEstudiante)

1.2 El resultado no es un atributo de la BD pero las expresiones del algoritmo involucra una instancia específica de la tabla sujeto.

Ejemplo:

El promedio en estudiante es calculado como avg(sujeto.cursa.promedioTCP)+avg(sujeto.cursa.promedioES)+ max(sujeto.EF.nota)

2. El resultado es un atributo de la BD y en específico de la tabla sujeto.

Ejemplo:

El promedio_TCP en cursa para promedioTCP es calculado como avg(sujeto.asignatura.tcp.nota)

Para estas posibilidades, tras hacer los pertinentes planes de ejecución que se muestran más adelante, y tomar la solución óptima, se asume lo siguiente:

1.1 Generar una función sin parámetros que calcula el resultado.

1.2 Generar una función que calcula el resultado, a la que se le pasan como parámetros los atributos llaves del sujeto.

2. Generar una función que calcula el resultado, a la que se le pasan como parámetros los atributos llaves del sujeto, un procedimiento almacenado con los atributos llave del sujeto como parámetros y un disparador para cada tabla involucrada en el cálculo.

2.2.2 Análisis del patrón de regla de clasificación

Al analizar el patrón de esta regla se consideran las posibilidades de que la clasificación estuviera en función de un valor del atributo, o que fuera ya una tabla de la base de datos, en estos casos se hizo complejo abordar al detalle la diversidad de variantes que dependían de la filosofía para diseñar la base de datos, además en caso de que se considerara como una tabla, se tendría información para completar los atributos de la llave quizás, pero no necesariamente los demás atributos, en otro caso que la clasificación se hiciese a través del valor de un atributo o conjunto de estos, no es necesario hacer cambios al patrón; para este último caso que es el que se implementa en este trabajo se mantiene el patrón de la versión anterior.

<determinante> <sujeto> [no] es definido como <clasificación>

[(si | a menos que) <característica>]

Al igual que el patrón la gramática no sufrió cambios, a continuación se puede apreciar los nodos que aportan valor semántico; de estos solo es necesario implementar el nodo raíz TCLASIF, para los nodos de los demás niveles del ASA de la regla son reutilizados los definidos en la versión anterior de la herramienta.

```
clasificacion
      :     TDETERMINANTE sujeto TES_DEF clasif TSI
caracteristicas -> ^(TCLASIF sujeto clasif caracteristicas)
            | TDETERMINANTE sujeto TNO_ES_DEF clasif
TA_MENOS_QUE caracteristicas->^(TCLASIF sujeto clasif
caracteristicas);
```

El diagrama de sintaxis se presenta a continuación.

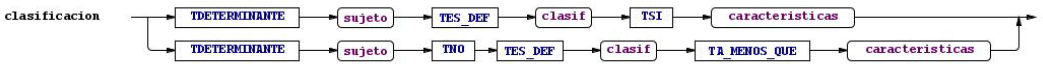


Figura 7: Diagrama de sintaxis del patrón de clasificación.

Para la implementación formal de esta regla se mantiene la propuesta de usar una función lógica que retorne verdadero en caso de que el sujeto cumpla con ciertas características y falso en caso contrario.

2.2.3 Análisis del patrón de regla de notificación

Se realizó el análisis de este patrón, concluyendo que no es necesario hacerle transformaciones, de manera que conserva la estructura con que fue definido en el LPT.

Notificar <mensaje> si <hechos>.

La gramática no se transforma y al igual que en las reglas anteriores es necesario implementar el nodo raíz del ASA TNOTIF. La gramática y su diagrama de sintaxis son presentados a continuación.

```
notificacion
      :     TNOTIFICAR mensaje TSI hechos->^(TNOTIF mensaje
hechos);
```

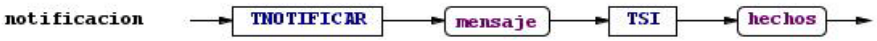


Figura 8: Diagrama de sintaxis del patrón de notificación.

En el trabajo de (Perez Alonso, 2010), se propone para la implementación formal de esta regla, el uso de un procedimiento almacenado en el que se verifica la ocurrencia de los hechos para cada evento en las tablas correspondientes. Al apreciar la semejanza de esta regla con la de restricción implementada por (Calderón Solís, 2011), se decidió implementarla de forma similar, a través de una vista que contiene los elementos en que ocurren determinados hechos y disparadores en las tablas pertinentes.

2.3 Diseño de las clases

El diseño de clases para la herramienta es transformado en el sentido de que se le añaden los componentes necesarios a la aplicación para lograr la implementación automática de todos los patrones cercanos a los datos; manteniéndose el diseño original de la herramienta. En la Figura 9 se muestra una sección de su diagrama de clases.

La modularidad con que fue concebido el traductor cuyas funcionalidades se amplían en este trabajo, hizo posible la incorporación y adaptación de los artefactos necesarios conservando sin modificaciones el código precedente.

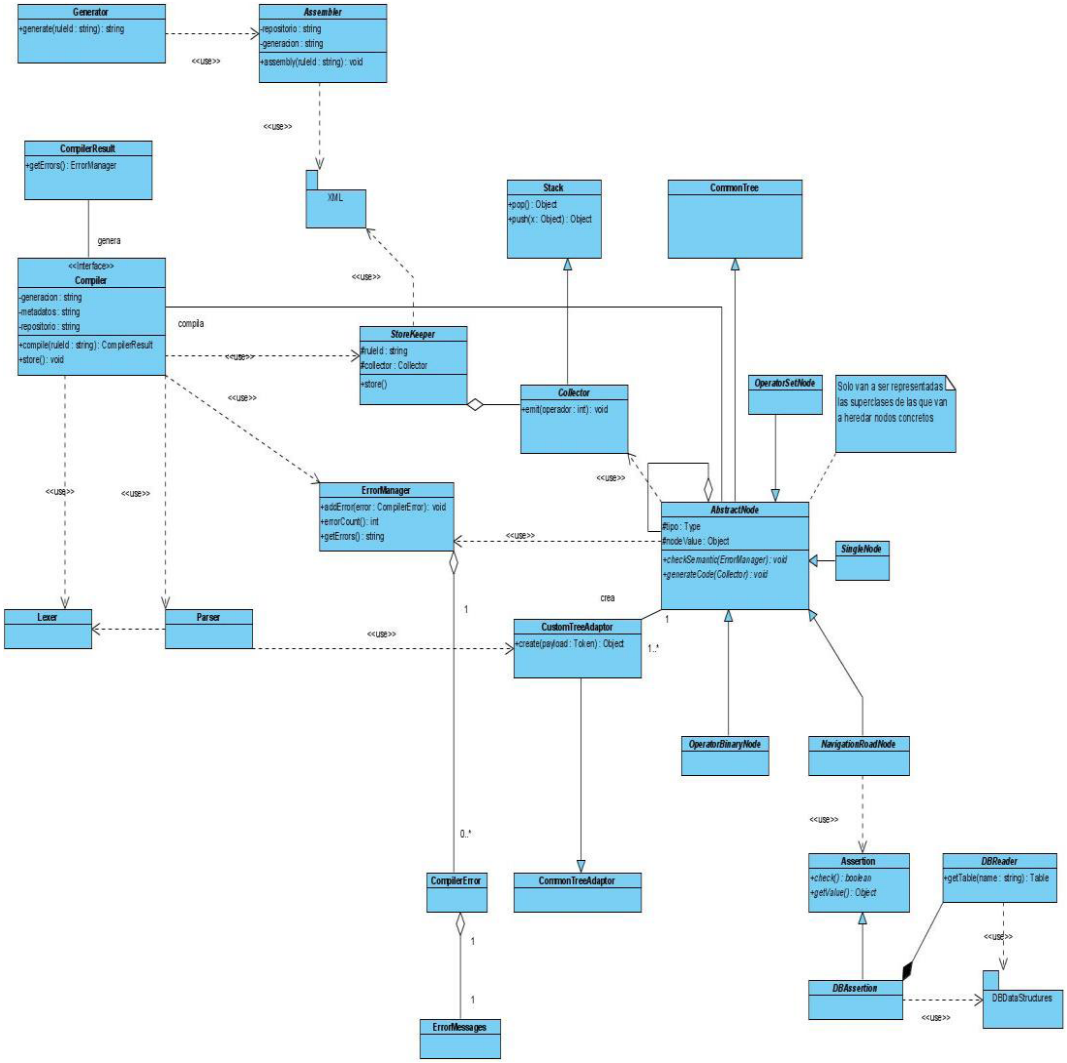


Figura 9: Parte del Diagrama de Clases.

Solis, en (Calderón Solís, 2011) sugiere los pasos a seguir para la incorporación de otras reglas a la aplicación. Para cada regla es necesario implementar las clases que derivan del ASA como subclase de AbstractNode, las partes o nodos del ASA que pueden ser comunes a cada regla ya están implementadas en la aplicación, para cada regla es necesario agregar la clase correspondiente a su nodo raíz, de modo que a la regla de cálculo se le hizo corresponder ComputoNode, a la de clasificación ClasificationNode y a la de notificación NotificationNode (ver Figura 10), cada una de estas clases posee los métodos checkSemantic() y gererateCode(), los cuales han de ser redefinidos en cada caso.

(A partir de este momento, las clases de color azul en los diagramas fueron las que se incorporaron como resultado de este trabajo).

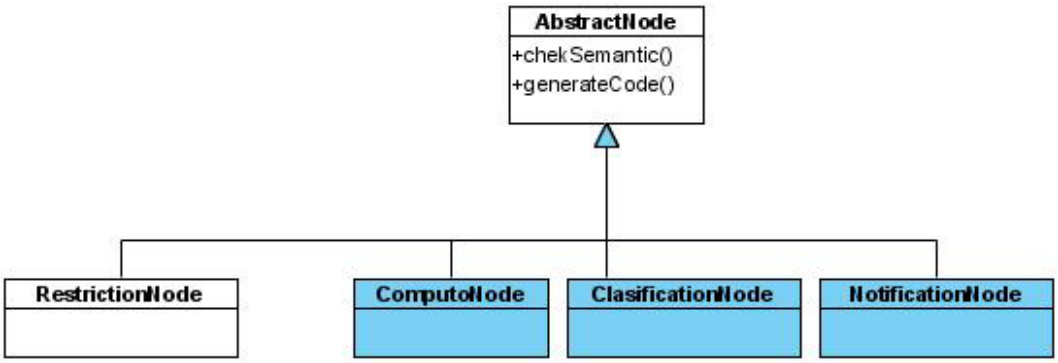


Figura 10: Clases correspondientes a los nodos del ASA de cada regla.

Para cada una de las reglas fue necesario implementar una clase que hereda de StoreKeeper y redefiniera el método store() correspondiente (ver Figura 11). Estas clases se encargan de almacenar en el repositorio de generación, la información necesaria para la generación de la regla obtenida en el proceso de compilación.

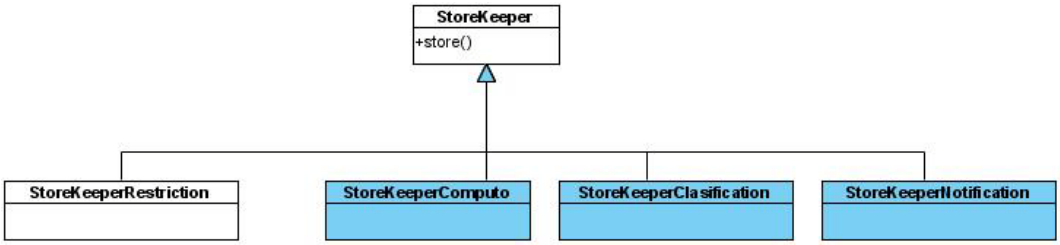


Figura 11: Clases que almacenan la información necesaria para generar la regla.

Al paquete Assembler se le agregan las clases correspondientes a cada regla y gestor para el que se va a generar.

Atendiendo a los recursos de BD propuestos para la implementación de la regla de cálculo, en la clase AssemblerCalculus se implementan los métodos createFunction(), createProcedure() y createTrigger() que se redefine en las clases AssemblerCalculusSQLServer y AssemblerCalculusPostgres de acuerdo a las características de estos recursos en los gestores para los que se crearon respectivamente. Dado que los recursos que se proponen para la formalización de la regla de notificación son los disparadores y las vistas, la clase AssemblerNotification posee los métodos createTrigger() y createView() que se redefinen para cada gestor en las subclases correspondientes. De igual forma, en la regla de clasificación que se implementa por medio de una función, se tiene el método createFunction() y se redefine en las subclases de acuerdo al gestor. Las clases añadidas al diseño se muestran en la Figura 12.

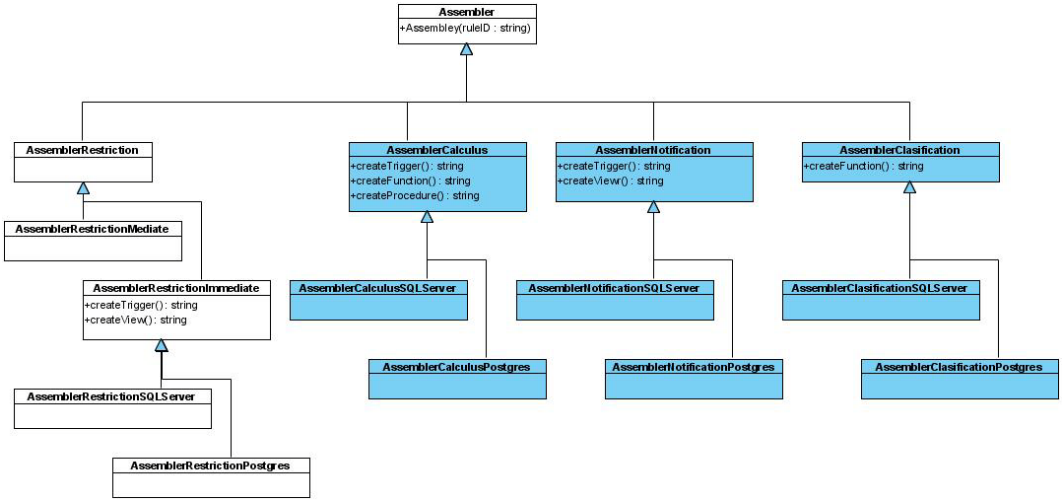


Figura 12: Clases que se agregan al paquete Assembler para cada regla y gestor.

2.4 Revisión y modificaciones del repositorio de regla

Para este trabajo se mantienen los repositorios de Metadatos (ver Anexo 3), de Generación y de Reglas utilizados por (Calderón Solís, 2011) en la versión anterior de la herramienta. La estructura general de estos repositorios se mantiene, se realizan modificaciones en los repositorios de generación y el de regla, las transformaciones en el de generación se hacen para incluir los datos necesarios en el proceso de generación de las reglas a implementar en esta versión y el de Reglas, se hacen adiciones para el caso de que la regla sea implementada por una función, su tipo de retorno.

El repositorio de Generación contiene la información requerida para la generación de las reglas, estos datos se obtienen al compilar la regla en LPT que se lee del repositorio de Reglas. En la Figura 13 se muestra la estructura del repositorio original y en la Figura 14 los cambios hechos a este repositorio.

Inicialmente para la regla de restricción solo se almacenaban los atributos tipo, sujeto, complejidad y gestor; para la implementación de la regla de cálculo se usan otros recursos de BD por lo que se hace necesario agregarle:

1. A la etiqueta regla los atributos
    - *Resultado*: el resultado del cálculo que representa el nombre de la función que se crea en la implementación de la regla.
    - *Parámetros*: parámetros que deben ser pasados, entradas a la función.
    - *Atributo llave*: llave primaria de la tabla sujeto.

- *Atributo*: atributo de la BD en que debe ser almacenado el resultado del cálculo.
- *Tipo_retorno*: tipo de retorno de la función que se crea.
- *Tipo_atributo*: tipo de dato del atributo.

2. La etiqueta ListaTablas, que por cada tabla en el cálculo contiene una etiqueta *tabla* con los atributos:

- *Nombre:* nombre de la tabla a la que pertenece un operador en el algoritmo.
- *Llave_Macheo*: Este atributo guarda la llave foránea de esa tabla, a que llave primaria hace referencia y de que tabla proviene separados por ",", en caso de que la llave foránea se compuesta, se obtiene una lista con estos datos, separados por "/". Esta información facilita la correcta correspondencia de las llaves y los atributos en cada tabla.

Para la regla de clasificación se agregó una etiqueta *Clasificación* que guardara ese valor.

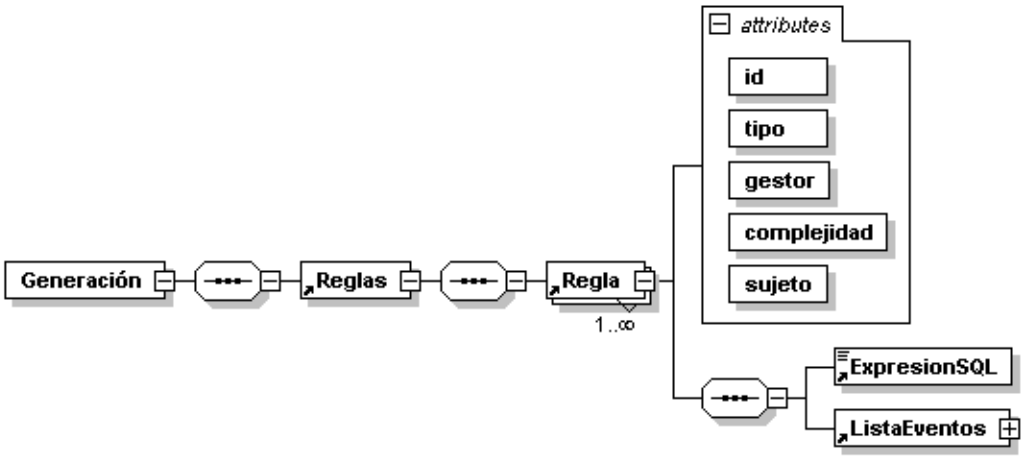


Figura 13: Esquema XSD del repositorio de Generación original.

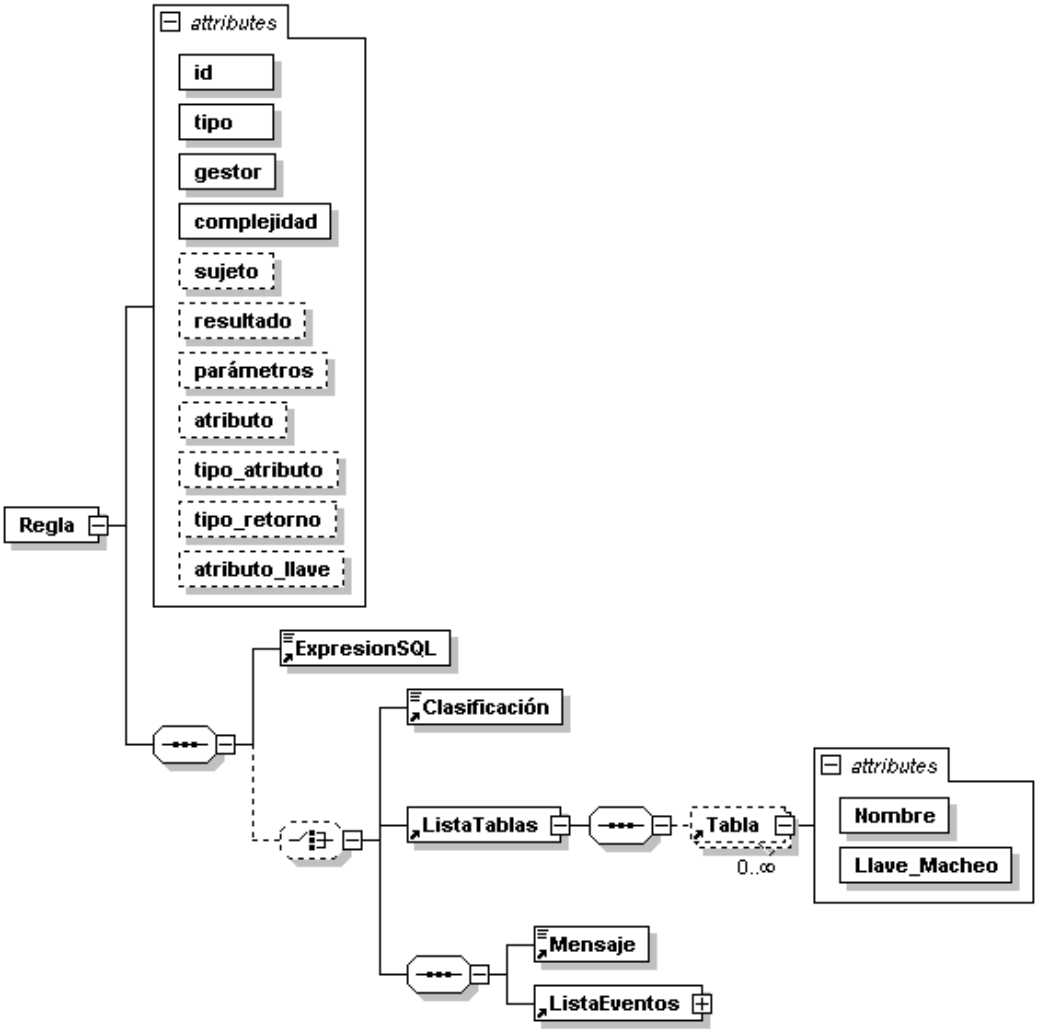


Figura 14: Esquema XSD del repositorio de Generación modificado.

En el repositorio de Reglas se almacena la regla en sus tres niveles de expresión, de cada una se tienen los atributos id, tipo, gestor y estado obligatorios además de la versión última fecha y enfoque opcionales. En el momento en que es añadida una regla en el repositorio se tiene el id y el estado, además de su expresión en los lenguajes Informal y técnico. El lenguaje formal se obtiene al ser generada la regla, para cada regla se tiene el recurso de BD correspondiente con nombre y tipo como atributos, además, en el caso de las funciones también el tipo de retorno y en el caso de los disparadores, la tabla donde se implementa (Figuras 15 y 16).

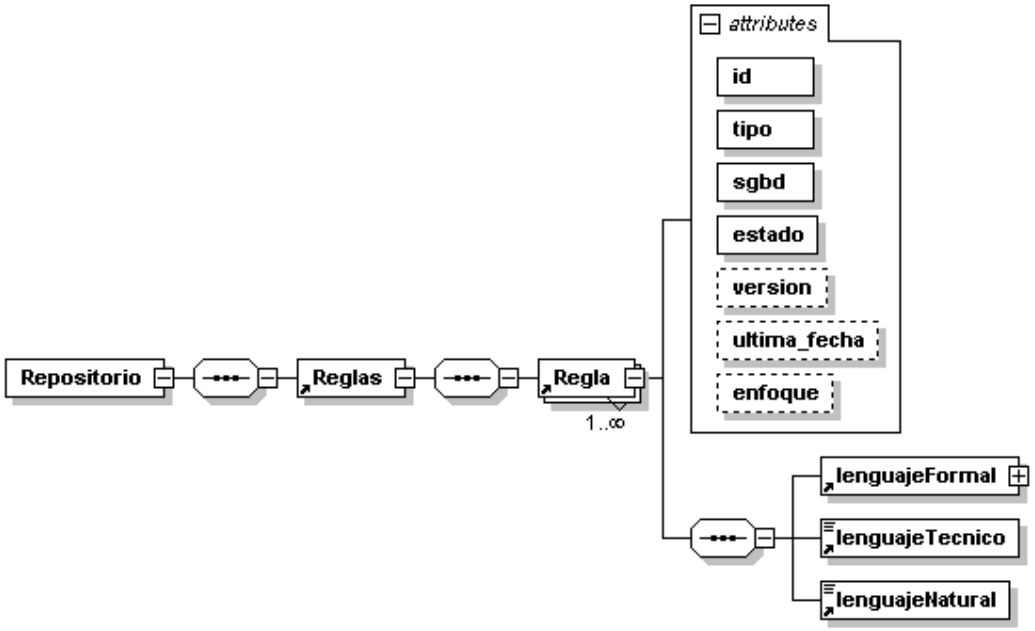


Figura 15: Esquema XSD del repositorio de Reglas.

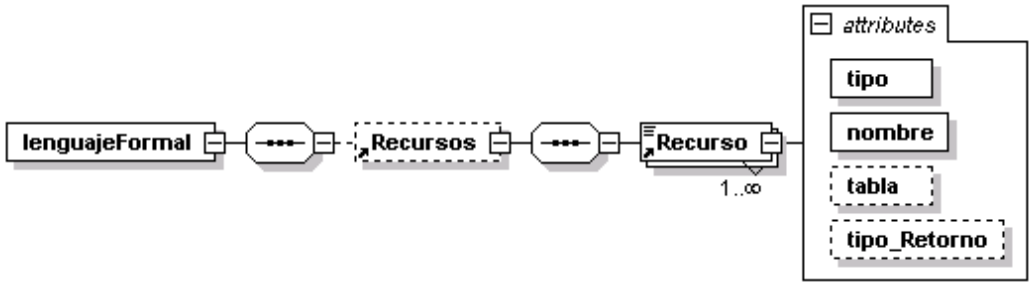


Figura 16: Sección correspondiente al lenguaje formal del esquema XSD del repositorio de Reglas.

2.5 Consideraciones para la implementación de las reglas

Para la implementación de las reglas cercanas a los datos que se agregan a la versión actual de la herramienta, fue necesario analizar las propuestas que presentó (Perez Alonso, 2010) sobre su formulación como recursos de BD, también se tomó en cuenta la implementación de la regla de restricción por parte de (Calderón Solís, 2011) en la versión precedente de esta herramienta y se probaron otras posibles soluciones que se presentaron en los epígrafe 2.1 y 2.3.

La regla de cálculo implementada de la forma propuesta por (Perez Alonso, 2010), no considera la posibilidad de tener un atributo calculable en la BD que recibiera precisamente el resultado de aplicar esta regla. En el epígrafe 2.1 se enuncian las soluciones que se presentaron a cada una de las posibilidades de esta regla.

La variante 1.1 en la generación de la regla, es de carácter general, o sea que las operaciones en el cálculo no involucran a una instancia específica de una tabla,

sino a todos sus registros; para esta se crea una función sin parámetros con el nombre del <resultado> que devuelve el valor de las operaciones definidas en el algoritmo, el tipo de retorno es float, el más abarcador de los tipos de datos para el resultado; esta función puede ser llamada directamente por una aplicación o desde otra regla.

Ejemplo:

RN#1:

La matricula es calculado como sizeof(estudiante.codigo)

```
CREATE FUNCTION matricula () RETURNS float AS BEGIN DECLARE
@y float SET @y = (SELECT COUNT( a.codigo ) FROM estudiante a)
RETURN @y END
```

La variante 1.2, está ligada directamente con una instancia en específico de la tabla sujeto, se implementó por medio de una función que lleva por nombre el <resultado>, a la cual se le pasan como parámetros los atributos llave del <sujeto> y computa el <resultado>, al igual que la anterior, el tipo de retorno es float y puede ser ejecutada por el usuario o dentro del cuerpo de otra regla.

Ejemplo:

RN#2

El promedo_Asignatura en cursa es calculado como sujeto.promediotcp+sujeto.promedioes+max(sujeto.ASIGNATURA.EF.notaef)

```
CREATE FUNCTION promedo_Asignatura (@anombreasig
text,@agradoasig integer,@ecodigo integer) RETURNS float AS
BEGIN DECLARE @y float SET @y = (((SELECT a.promediotcp FROM
cursa a) + (SELECT a.promedioes FROM cursa a)) + (SELECT MAX(
c.notaef ) FROM cursa a , asignatura b , ef c WHERE
(a.anombreasig=b.nombreasig) AND (a.agradoasig=b.gradoasig)
AND (b.nombreasig=c.anombreasig) AND
(b.gradoasig=c.agradoasig)))  RETURN @y END
```

En la segunda posibilidad se soluciona la problemática de almacenar el resultado en una tabla de la BD, este resultado debe ser actualizado ante la ocurrencia de

algún evento de inserción, actualización o eliminación, para lograr esto se analizaron tres posibles variantes:

- Implementar una función que compute el resultado y lo inserte en la tabla correspondiente, y un disparador en cada tabla involucrada en el cálculo que ejecuta la función pasándole como parámetros los atributos llave de la tabla <sujeto>.
- Un procedimiento almacenado que calcule el resultado del algoritmo y lo almacena en la tabla correspondiente y un disparador en cada tabla involucrada en el cálculo llama al procedimiento almacenado pasándole los atributos llave del <sujeto> como parámetros.
- Un procedimiento almacenado que se encarga de almacenar el <resultado> en la tabla destino, una función que calcula el <algoritmo> y un disparador en cada tabla involucrada en el cálculo. Al ocurrir una inserción, actualización u eliminación en alguna de las tablas donde se implementaron los disparador, este toma los atributos llave del <sujeto> modificado y se invoca el procedimiento almacenado con estos atributos como parámetros, este procedimiento ejecuta la función con los parámetros que recibe y actualiza la tabla correspondiente con el <resultado>.

En todas estas variantes se le da un mensaje de error al usuario en caso de que no sea posible terminar la operación.

Tras hacer los planes de ejecución para cada una de esas posibilidades enfrentando una misma operación de inserción (ver Figuras 17, 18 y 19), se pudo apreciar que no hay diferencias significativas entre ellas, entonces, tratando de modularizar el trabajo se decide utilizar la tercera variante, quedando así cada recurso implementado con una tarea específica a realizar y en caso de que sea necesario hacerle modificaciones a alguno de estos recursos en el futuro, solo es necesario modificar una parte en específico sin necesidad de interactuar con el código del resto de la regla. El nombre de la función que se crea es el del <resultado> lo que da la posibilidad de ser llamada por el usuario o dentro de otra regla como las anteriores. El tipo de retorno de la función coincide con el del <atributo> donde se va a insertar el <resultado>. El procedimiento almacenado se le llama PROCRN#? según la regla, los disparadores son nombrados TIRN#? para el evento de inserción, TURN#? para la actualización y TDRN#? para la eliminación. Estos nombres no son significativos para el usuario, pues solo se utilizan en las llamadas entre ellos para completar las operaciones del algoritmo.

Ejemplo:

El promedioEvalSist en cursa para promedioes es calculado como avg(sujeto.ASIGNATURA.ES.notaes)

```
CREATE FUNCTION promedioES (@anombreasig text,@agradoasig
integer,@ecodigo integer) RETURNS float …
--ver código de la implementación en Anexo 5

CREATE PROCEDURE PROCRN#4 (@anombreasig text,@agradoasig
integer,@ecodigo integer) AS UPDATE cursa  SET PROMEDIOES =
DBO.promedioES ( @anombreasig, @agradoasig, @ecodigo )  …
--ver código de la implementación en Anexo 5

CREATE TRIGGER TIRN#4_1 ON es AFTER INSERT  AS DECLARE …
--ver código de la implementación en Anexo 5

CREATE TRIGGER TURN#4_2 ON es AFTER UPDATE  AS IF (  UPDATE
--ver código de la implementación en Anexo 5

CREATE TRIGGER TDRN#4_3 ON es AFTER DELETE  AS DECLARE
--ver código de la implementación en Anexo 5
```

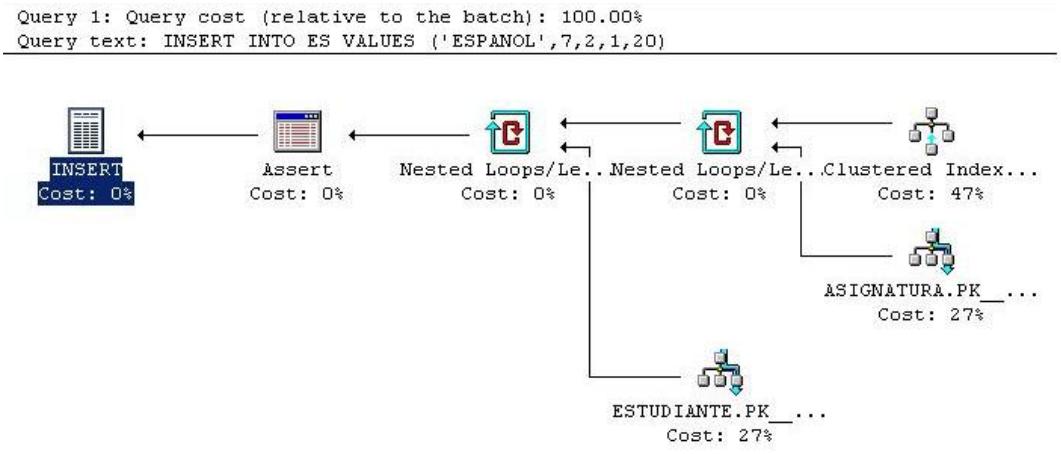


Figura 17: Plan de ejecución con disparador y función.

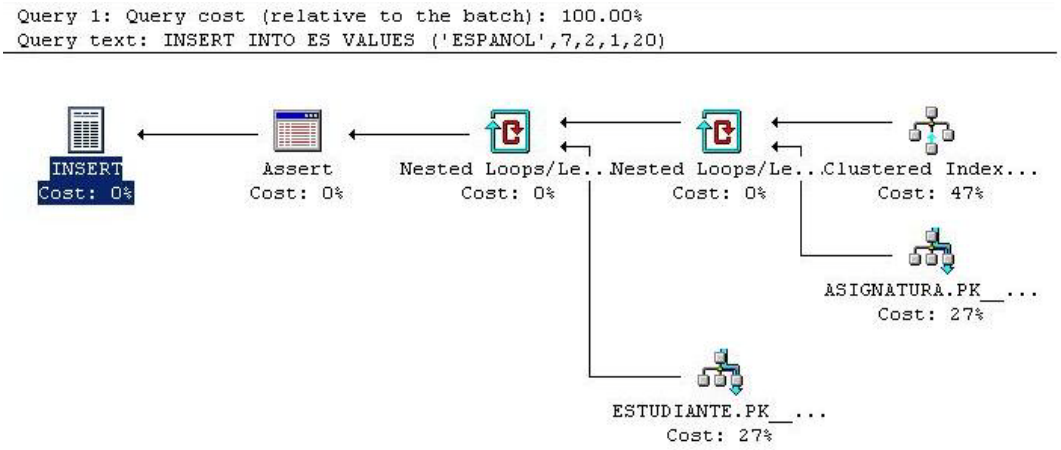


Figura 18: Plan de ejecución con procedimiento almacenado y función.

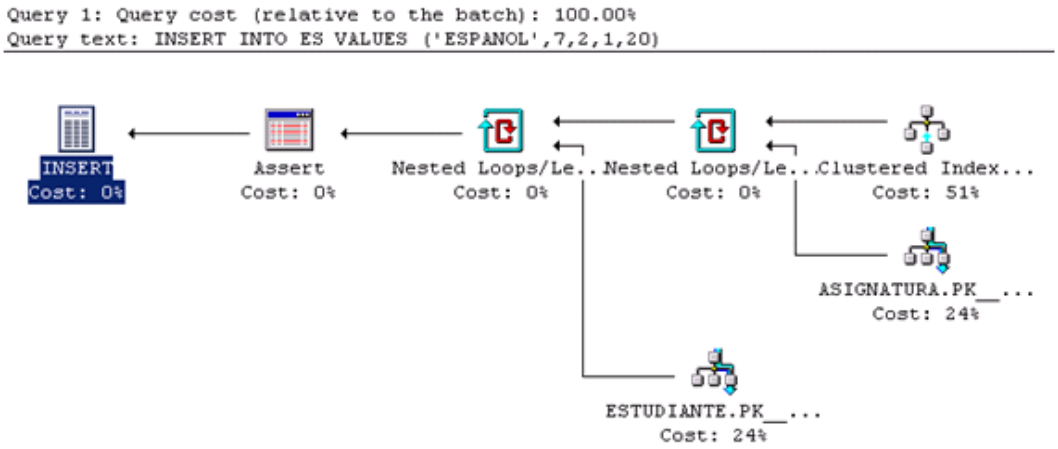


Figura 19: Plan de ejecución con disparador, procedimiento almacenado y función.

La regla de notificación es similar a la regla de Restricción, en el sentido de que descubre una contradicción con una política del negocio ante la ocurrencia de un evento sobre los datos involucrados, si se llega a un estado no deseado, este se rechaza; la notificación no llega a rechazar este estado, sino que emite un mensaje informando la ocurrencia de ese hecho. Para la implementación de la regla de Restricción, (Calderón Solís, 2011), utilizó una vista que es realizada sobre el <sujeto> mediante una consulta externa que contiene subconsultas generadas por la notación punto, junto a la implementación de los operadores. Estas subconsultas se obtienen como resultado de la implementación de las <características> y los <hechos>, y disparadores en las tablas contenidas en la lista de eventos que determinan si existe algún <sujeto> infringiendo la regla. El contenido de la vista son los elementos que violan la regla, así que la función del disparador es determinar si existe algún elemento en la vista y en ese caso deshacer la

transacción. Con la regla de notificación se hizo de forma similar; se creó una vista que contiene los elementos que se necesitan notificar y los disparadores en las tablas correspondientes que determinan si existe algún elemento en la vista, en caso afirmativo se notifica pero no se rechaza o deshace la transacción. El convenio utilizado para los nombres de estos recursos es el mismo utilizado en la implementación de la regla de Restricción T[I | U | D]RN#? para los disparadores y la vista VRN#?.

Ejemplo:

Notificar 'Estudiante desaprobado' si estudiante.promedio<60

```
CREATE TRIGGER TIRN#1_1 ON estudiante FOR INSERT AS
if(EXISTS(SELECT * FROM VRN#1 ) ) BEGIN raiserror('Estudiante
desaprobado',10,1); END
CREATE TRIGGER TURN#1_1 ON estudiante FOR UPDATE AS
if(EXISTS(SELECT * FROM VRN#1 ) ) BEGIN raiserror('Estudiante
desaprobado',10,1); END
 CREATE VIEW VRN#1 AS SELECT * FROM estudiante WHERE ( (SELECT
COUNT( a.promedio ) FROM estudiante a WHERE (a.promedio < 3
))>0 );
```

La regla de clasificación se implementó de la forma que plantea (Perez Alonso, 2010) por medio de una función cuyo nombre es la <clasificación> a la que se le pasan como parámetros la llave del <sujeto>, esta retorna verdadero o falso en dependencia de que el <sujeto> pertenezca o no al la clasificación que se define.

Ejemplo:

RN#5

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

```
CREATE FUNCTION desaprobado ( @codigo integer )RETURNS int
AS BEGIN DECLARE @x int; IF (( SELECT COUNT(*) FROM estudiante
sujeto WHERE ( (SELECT a.promedio FROM estudiante a WHERE
(sujeto.codigo=a.codigo) AND (sujeto.codigo=@codigo)) < 3))
!=0 ) SET @x=1; ELSE SET @x=0; RETURN @x; END
```

Dado que SQL Server no permite crear funciones booleanas, se retorna un entero, 1 si cumple con las <características> o 0 en otro caso. En PostgreSQL la función si puede ser booleana.

2.5.1 Interdependencia entre reglas

Alain Pérez en (Perez Alonso, 2010), presenta las interdependencias entre las reglas de negocio cercanas a los datos (ver la tabla 2), dado el hecho de que en la versión precedente del LPT solo se implementó la regla de restricción, estas interdependencias no se pudieron probar en la práctica, en esta investigación se ilustrarán ejemplos donde en el cuerpo de una regla se hacen llamadas a otras reglas.

En este trabajo no se trató el tema de la eliminación de estas reglas interrelacionadas, que se recomienda terminar en futuras versiones del traductor. Las interdependencias entre las reglas de notificación y clasificación no se analizaron porque en la definición de la regla de notificación no se hace referencia al <sujeto>, de igual forma sucede en el caso de las reglas de notificación y de cómputo donde se haga referencia al sujeto.

Tipo de Regla	Posibles Reglas a depender
CLASIFICACIÓN	CLASIFICACIÓN CÓMPUTO
CÓMPUTO	CÓMPUTO
NOTIFICACIÓN	CLASIFICACIÓN CÓMPUTO
RESTRICCIÓN	CLASIFICACIÓN CÓMPUTO

Tabla 2: Interdependencia de los Patrones para Reglas de Negocio cercanas a los datos.

2.6 Conclusiones Parciales

En este capítulo se presentó la arquitectura general de la herramienta, dando las nociones del proceso de traducción de la regla. Se analizaron los patrones y las

gramáticas correspondientes a las reglas de cómputo, clasificación y notificación que se implementan en esta versión de la herramienta. Se exponen además las modificaciones hechas al diseño de las clases de la herramienta a raíz de la incorporación de estas reglas, las transformaciones a los repositorios y las consideraciones que se tuvieron en cuenta para la implementación de cada tipo de regla.

Capítulo III:

IMPLEMENTACIÓN AUTOMÁTICA DE LAS REGLAS.

CAPÍTULO 3: IMPLEMENTACIÓN AUTOMÁTICA DE LAS REGLAS.

En este capítulo se aborda la implementación automática de las reglas incorporadas a la herramienta para los gestores de BD PostgreSQL y SQL Server. Se documentan detalles relevantes en la implementación formal y uso de las reglas y se toma como caso de estudio la BD que se emplea en la Escuela Secundaria Básica Urbana Héctor Martínez Valladares para el control de la evaluación estudiantil, a la cual se le aplican las reglas de negocio que se detectan. Se muestra el contenido de los repositorios de Generación y de Regla para cada tipo de regla y gestor correspondiente. Además se presenta la herramienta con los nuevos operadores añadidos para la edición de las reglas en LPT.

3.1 Implementación de las reglas en lenguaje formal

El usuario de esta herramienta debe conocer las posibilidades de la implementación de estas reglas como recursos de BD, así como las variantes en la confección de las reglas teniendo en cuenta las que ya existen en el repositorio y se pueden reutilizar conociendo las posibles interdependencias entre ellas. También es importante que el usuario del traductor conozca los posibles usos de cada regla para enfocarlas debidamente. A continuación se presentan las características fundamentales de las reglas que se añaden en este trabajo y su implementación formal.

3.1.1 Regla de cálculo

Esta regla, como se explicó en el capítulo anterior, puede tener variantes, la primera analizada es útil para cálculos generales como porcientos y totales, por ejemplo:

RN#1

El porcientoAprobados es calculado como 100 * sizeof (estudiante.promedio > 60) / sizeof(estudiante.codigo)

Cuya implementación formal se traduce a:

```
CREATE FUNCTION porcientoAprobados () RETURNS float AS BEGIN
DECLARE @y float SET @y = ((100 * (SELECT COUNT( a.PROMEDIO )
FROM ESTUDIANTE a WHERE (a.PROMEDIO < 60 ))) / (SELECT COUNT(
a.CODIGO ) FROM ESTUDIANTE a)) RETURN @y END
```

La regla de cómputo en su implementación formal contiene una función, al ser activada en la BD introduce un nuevo término en el negocio con ese nombre, teniendo en cuenta que las interdependencias de esta regla se dan con reglas del mismo tipo, puede ser utilizada directamente por el usuario o dentro de otra. Por ejemplo; si se tiene una regla que calcula la matrícula como la cantidad de estudiantes en la tabla estudiantes, la regla anterior se puede formular de la siguiente manera:

RN#1

El porcientoAprobados es calculado como 100 * sizeof (estudiante.promedio > 60) / matricula

Esta opción se traduce a la función porcientoAprobados que contiene la llamada a la función matricula como se muestra a continuación:

```
CREATE FUNCTION porcientoAprobados () RETURNS float AS BEGIN
DECLARE @y float SET @y = ((100 * (SELECT COUNT( a.PROMEDIO )
FROM ESTUDIANTE a WHERE (a.PROMEDIO < 60 ))) /
dbo.matricula()) RETURN @y END
```

En la otra alternativa de la primera variante se involucra un <sujeto> en específico de la BD, es útil para el cálculo de un resultado que se desea obtener de un elemento determinado. Por ejemplo; el promedio de un estudiante, el salario de un trabajador, la función que se genera en su implementación formal contiene los atributos llave del <sujeto> como parámetros, de esta forma el algoritmo se ejecuta solo para aquella tupla que contenga estas llaves. Para utilizar el término del negocio que se agrega en la activación de la regla en el cuerpo de otra, es necesario pasarle como parámetro la palabra reservada sujeto, por ejemplo:

Si se tiene una regla que calcula el promedio de las Evaluaciones Sistemáticas, otra que calcula el promedio de los Trabajos de Control Parciales y otra la Nota en Evaluación Final:

RN#3

El promedioES en estudiante es calculado como avg(sujeto.es.notaes)

RN#4

El promedioTCP en estudiante es calculado como avg(sujeto.tcp.nota)

RN#6

La EvalFinal en estudiante es calculado como max(sujeto.ef.notaef)

Una posible regla para calcular el promedio del estudiante sería:

RN#7

El promedio en estudiante es calculado como promedioES(sujeto) + promedioTCP(sujeto) + evalFinal(sujeto)

En la implementación formal de esta regla se genera la llamada a esta función y se sustituye sujeto por sus atributos llave:

```
CREATE FUNCTION promedio (@CODIGO int) RETURNS float AS BEGIN DECLARE @y float SET @y = ((dbo.promedioES(@CODIGO) + dbo.promedioTCP(@CODIGO)) + dbo.EvalFinal(@CODIGO)) RETURN @y END
```

La tercera alternativa para la regla de cálculo es útil para automatizar el almacenamiento de los valores en campos calculables de la BD como por ejemplo el promedio y subtotales, la opción [para <atributo>], como se explicó en el capítulo anterior se traduce en disparadores que toman la llave del elemento con que se va a calcular y el procedimiento que se encarga de llamar a la función y de almacenar el valor en el <atributo> de la tabla <sujeto>. Para poner un ejemplo se tomará la siguiente regla que calcula el promedio de los TCP en la tabla <sujeto>, CURSA en este caso, para el <atributo> PROMEDIOTCP:

RN#12

El promedioTCP en cursa para PROMEDIOTCP es calculado como avg(sujeto.estudiante.cursa.asignatura.tcp.nota)

Que se traduce a:

```
CREATE FUNCTION promedioTCP (@anombreasig text,@agradoasig integer,@ecodigo integer) RETURNS float AS

--Código de la implementación en anexo 5

… END
```

```
CREATE PROCEDURE PROCRN#2 (@anombreasig text,@agradoasig
integer,@ecodigo integer) AS UPDATE cursa  SET PROMEDIOTCP =
DBO.promedioTCP ( @anombreasig, @agradoasig, @ecodigo )  WHERE
(cursa.anombreasig=@anombreasig) AND
(cursa.agradoasig=@agradoasig) AND (cursa.ecodigo=@ecodigo)

CREATE TRIGGER TIRN#2_1 ON tcp AFTER INSERT  AS DECLARE
@anombreasig text,@agradoasig integer,@ecodigo integer SET
@anombreasig=(SELECT anombreasig FROM INSERTED) SET
@agradoasig=(SELECT agradoasig FROM INSERTED) SET
@ecodigo=(SELECT ecodigo FROM INSERTED)   EXEC PROCRN#2
@anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN
RAISERROR ( 'No se pudo completar el cálculo', 16, 1 )
ROLLBACK TRANSACTION END

CREATE TRIGGER TURN#2_2 ON tcp AFTER UPDATE  AS IF (  UPDATE
(nota)  )

--Código de la implementación en anexo 5

…END

GO

CREATE TRIGGER TDRN#2_3 ON tcp AFTER DELETE  AS DECLARE

--Código de la implementación en anexo 5

END
```

Del camino de navegación del algoritmo se toman las tablas donde se deben implementar los disparadores, tcp en este caso.

3.1.2 Regla de clasificación

La utilidad de esta regla es determinar si un sujeto pertenece o no a una clasificación. Por ejemplo, si se desea saber si un estudiante x esta desaprobado y existe la regla con esta clasificación, entonces al ejecutarla pasándole los atributos identificadores del sujeto esta responde 't' si es verdadero o 'f' en caso contrario. Al ser activada, esta regla se inserta como término del negocio al igual que las reglas de cálculo, que también pueden ser utilizadas dentro del cuerpo de otra regla con la

que esté interrelacionada, en este caso pudiera ser dentro de una regla de restricción u otra clasificación, por ejemplo:

Si existe la regla que clasifica a un estudiante como desaprobado, ésta puede ser usada por otra del mismo tipo que define a un estudiante como crítico si tiene menos de 60 en las Evaluaciones Finales y esta desaprobado, o sea el promedio que acumula es menor que 60, la cual se expresa a continuación:

RN#18

Un estudiante es definido como crítico si max(sujeto.EF.notaef) < 60 AND desaprobado(sujeto)

Que se traduce en:

```
CREATE FUNCTION critico ( @CODIGO int )RETURNS CHAR AS BEGIN
DECLARE @x int; IF (( SELECT COUNT(*) FROM ESTUDIANTE sujeto
WHERE ( (SELECT MAX( b.NOTAEF ) FROM ESTUDIANTE a , EF b WHERE
(a.CODIGO=b.ECODIGO)) < 60 AND dbo.desaprobado(@CODIGO))) !=0
) SET @x=1; ELSE SET @x=0; RETURN @x; END
```

(En el siguiente ejemplo se utilizará una clasificación dentro de una Restricción para probar esta interdependencia, esta regla es hipotética y carece de sentido, se utilizará solo con el fin de ilustrar este caso)

RN#21

Un estudiante no puede tener sujeto.EF.notaef > 60 and desaprobado

Que se traduce a:

```
CREATE TRIGGER TIRN#21_1 ON EF FOR  INSERT  AS
if(EXISTS(SELECT * FROM VRN#21 ) ) BEGIN
raiserror('RN#21',16,1); rollback transaction; END

CREATE TRIGGER TURN#21_1 ON EF FOR  UPDATE  AS
if(EXISTS(SELECT * FROM VRN#21 ) ) BEGIN
raiserror('RN#21',16,1); rollback transaction; END

CREATE VIEW VRN#21 AS SELECT * FROM ESTUDIANTE sujeto WHERE (
(SELECT COUNT( b.NOTAEF ) FROM ESTUDIANTE a , EF b WHERE
```

```
(sujeto.CODIGO=a.CODIGO) AND (a.CODIGO=b.ECODIGO) AND
(b.NOTAEF > 60 ))>0 AND dbo.desaprobado(sujeto.CODIGO) )
```

3.1.3 Regla de notificación

Esta regla mantiene al usuario alerta ante la ocurrencia de un hecho que no significa un estado incorrecto pero se necesita tomar alguna decisión ante su ocurrencia, por ejemplo, notificar *"Porciento de aprobados muy bajo"* si el porciento de aprobados es menor que 75, ante esta situación se debe trazar una estrategia para superar estos resultados desfavorables. En LPT esta regla quedaría de la siguiente forma:

RN#19

Notificar 'porciento de aprobados muy bajo' si (100 * sizeof(estudiante.promedio> 60) / sizeof(estudiante.codigo))<65

Que se traduce a:

```
CREATE TRIGGER TIRN#19_1 ON ESTUDIANTE FOR  INSERT  AS
if(EXISTS(SELECT * FROM VRN#19 ) ) BEGIN raiserror('porciento
de aprobados muy bajo',10,1); END

CREATE TRIGGER TURN#19_1 ON ESTUDIANTE FOR  UPDATE  AS
if(EXISTS(SELECT * FROM VRN#19 ) ) BEGIN raiserror('porciento
de aprobados muy bajo',10,1); END

CREATE VIEW VRN#19 AS SELECT * FROM ESTUDIANTE WHERE ( ((100 *
(SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE
(a.PROMEDIO > 60 ))>0) / (SELECT COUNT( a.CODIGO ) FROM
ESTUDIANTE a)) < 65 );\
```

Las posibles interdependencias de esta regla son con las reglas de clasificación y cómputo. Para ilustrar el caso con la regla de cómputo se utiliza la regla anterior haciendo una llamada a la regla *matrIcula:*

RN#22

Notificar 'porciento de aprobados muy bajo' si (100 * sizeof(estudiante.promedio> 60) / matricula)<65

Quedando formalmente de la siguiente forma:

```
CREATE TRIGGER TIRN#22_1 ON ESTUDIANTE FOR  INSERT  AS
if(EXISTS(SELECT * FROM VRN#22 ) ) BEGIN raiserror('porciento
de aprobados muy bajo',10,1); END

CREATE TRIGGER TURN#22_1 ON ESTUDIANTE FOR  UPDATE  AS
if(EXISTS(SELECT * FROM VRN#22 ) ) BEGIN raiserror('porciento
de aprobados muy bajo',10,1); END

CREATE VIEW VRN#22 AS SELECT * FROM ESTUDIANTE WHERE ( ((100 *
(SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE
(a.PROMEDIO > 60 ))>0) / dbo.matricula()) < 65 );
```

3.2 Reglas utilizadas como caso de estudio

Para probar las reglas que se adicionan a esta versión de la herramienta se toma la BD que se utiliza en la ESBU Héctor Martínez de Villa Clara (su diagrama Relacional se muestra en el Anexo 8).

Esta BD presenta un ambiente ideal para probar la regla de cálculo, pues existen varios atributos calculables como es caso de PROMEDIOTCP, PROMEDIOES en la relación cursa y PROMEDIO en la tabla estudiante. Las reglas que se detectan para probar este tipo de regla se presentan a continuación en lenguaje natural:

1. El promedio de los TCP en cursa para PROMEDIOTCP se calcula como el promedio de las notas de los TCP.
2. El promedio de ES en cursa para PROMEDIOES se calcula como el promedio de las notas de las ES.
3. El promedioPorAsignatura se calcula como la suma del promedio de las ES más el promedio de los TCP más el máximo de las EF.
4. El promedio en estudiante para promedio se calcula como la suma del promedio de TCP más el promedio de ES más el máximo de las EF.
5. La matrícula se calcula como la cantidad de estudiantes.
6. El porciento de aprobados de la escuela es calculado como el porciento de estudiantes con promedio mayor que 60.

Para probar la regla de clasificación se implementan las siguientes reglas:

1. Un estudiante es definido como desaprobado si tiene un promedio < 60.

2. Un estudiante no es de alto aprovechamiento si su promedio no es mayor que 95.
3. Un estudiante es definido como crítico si el máximo de sus evaluaciones finales es menor que 60 y está desaprobado.

Para la regla de notificación se detectan las siguientes reglas:

1. Notificar estudiante desaprobado si tiene un promedio menor que 60.
2. Notificar estudiante mal en ES si el promedio de sus ES es menor que 20.
3. Notificar estudiante mal en TCP si el promedio de sus TCP es menor que 30.
4. Notificar 'porciento de aprobados muy bajo' si el porciento de aprobados es menor que 65.

La implementación formal de algunas de estas reglas en PostgreSQL se puede ver en el anexo 5.

3.3 Repositorios de Regla y Generación

El repositorio de Reglas, como se menciona en el capítulo anterior, contiene las reglas en sus tres niveles de expresión, a continuación se presentan ejemplos de los datos que almacena para cada tipo de regla que se implementa en este trabajo. También se muestra el contenido del repositorio de Generación con la información necesaria para la generación de las reglas que se presentan en el repositorio de Regla.

La siguiente figura muestra el contenido del repositorio para una regla de cálculo en la variante 1.1:

```
- <Regla id="RN#1" estado="activa" ultima_fecha="2012-6-12" version="1.0" tipo="calculus"
    sgbd="sqlserver" enfoque="">
  - <lenguajeFormal>
    - <Recursos>
        <Recurso tipo="FUNCTION" nombre="porcientoAprobados">CREATE FUNCTION
          porcientoAprobados ( ) RETURNS float AS BEGIN DECLARE @y float SET @y =
          (( 100 * (SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE
          (a.PROMEDIO < 60 ))) / (SELECT COUNT( a.CODIGO ) FROM ESTUDIANTE a))
          RETURN @y END</Recurso>
      </Recursos>
    </lenguajeFormal>
    <lenguajeTecnico>El porcientoAprobados es calculado como 100 * sizeof
      (estudiante.promedio< 60 ) / sizeof(estudiante.codigo)</lenguajeTecnico>
    <lenguajeNatural>El porciento de aprobados se calcula como el porciento de estudiantes
      con promedio mayor que 60</lenguajeNatural>
  </Regla>
```

Figura 20: Sección del repositorio de reglas para la regla de cálculo variante 1.1.

El contenido del repositorio de Generación para esta regla es el siguiente:

```
- <Regla id="RN#1" gestor="sqlserver" resultado="porcientoAprobados" complejidad="1"
    Tipo_retorno="float" tipo="calculus">
    <ExpresionSQL>((100 * (SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE
      (a.PROMEDIO < 60 ))) / (SELECT COUNT( a.CODIGO ) FROM ESTUDIANTE a))
      </ExpresionSQL>
    <ListaTablas />
  </Regla>
```

Figura 21: Sección del repositorio de generación para la regla de cálculo variante 1.1.

Un ejemplo para la variante 1.2 es el siguiente:

```
- <Regla id="RN#22" estado="inactiva" tipo="calculus" sgbd="sqlserver" enfoque="">
  - <lenguajeFormal>
    - <Recursos>
        <Recurso nombre="promedio" tipo="FUNCTION" tipo_Retorno="float">CREATE
          FUNCTION promedio (@CODIGO int) RETURNS float AS BEGIN DECLARE @y float
          SET @y = (((SELECT AVG( b.NOTAES ) FROM ESTUDIANTE a , ES b WHERE
          (a.CODIGO=b.ECODIGO)) + (SELECT AVG( b.NOTA ) FROM ESTUDIANTE a , TCP
          b WHERE (a.CODIGO=b.ECODIGO))) + (SELECT MAX( b.NOTAEF ) FROM
          ESTUDIANTE a , EF b WHERE (a.CODIGO=b.ECODIGO))) RETURN @y
          END</Recurso>
      </Recursos>
    </lenguajeFormal>
    <lenguajeTecnico>El promedio en estudiante es calculado como avg(sujeto.es.notaes)+avg
      (sujeto.tcp.nota)+max(sujeto.ef.notaef)</lenguajeTecnico>
    <lenguajeNatural />
  </Regla>
```

Figura 22: Sección del repositorio de reglas para la regla de cálculo variante 1.2

La información que contiene el repositorio de Generación para esta regla es:

```
- <Regla id="RN#22" tipo="calculus" gestor="sqlserver" resultado="promedio" complejidad="2" Tipo_retorno="float" sujeto="ESTUDIANTE"
    atributo_llave="CODIGO" parámetros="_CODIGO int">
    <ExpresionSQL>(((SELECT AVG( b.NOTAES ) FROM ESTUDIANTE a , ES b WHERE (a.CODIGO=b.ECODIGO)) + (SELECT AVG
      ( b.NOTA ) FROM ESTUDIANTE a , TCP b WHERE (a.CODIGO=b.ECODIGO))) + (SELECT MAX( b.NOTAEF ) FROM ESTUDIANTE a ,
      EF b WHERE (a.CODIGO=b.ECODIGO)))</ExpresionSQL>
  - <ListaTablas>
      <Tabla Nombre="ES"
        Llave="ANOMBREASIG,NOMBREASIG,ASIGNATURA/AGRADOASIG,GRADOASIG,ASIGNATURA/ECODIGO,CODIGO,ESTUDIANTE/" />
      <Tabla Nombre="TCP"
        Llave="ANOMBREASIG,NOMBREASIG,ASIGNATURA/AGRADOASIG,GRADOASIG,ASIGNATURA/ECODIGO,CODIGO,ESTUDIANTE/" />
      <Tabla Nombre="EF"
        Llave="ANOMBREASIG,NOMBREASIG,ASIGNATURA/AGRADOASIG,GRADOASIG,ASIGNATURA/ECODIGO,CODIGO,ESTUDIANTE/" />
    </ListaTablas>
  </Regla>
```

Figura 23: Sección del repositorio de generación para la regla de cálculo variante 1.2.

De la segunda variante de la regla de cálculo se puede poner el siguiente ejemplo:

```
- <Regla id="RN#2" estado="inactiva" tipo="calculus" sgbd="sqlserver" enfoque="" ultima_fecha="2012-6-11"
    version="1.0">
  - <lenguajeFormal>
    - <Recursos>
        <Recurso nombre="promedioTCP" tipo="FUNCTION" tipo_Retorno="double precision">CREATE
          FUNCTION promedioTCP (@anombreasig text,@agradoasig integer,@ecodigo integer) RETURNS
          float AS BEGIN DECLARE @y float SET @y = (SELECT AVG( e.nota ) FROM cursa a , estudiante b ,
          cursa c , asignatura d , tcp e WHERE (a.ecodigo=b.codigo) AND (b.codigo=c.ecodigo) AND
          (c.anombreasig=d.nombreasig) AND (c.agradoasig=d.gradoasig) AND
          (d.nombreasig=e.anombreasig) AND (d.gradoasig=e.agradoasig)) RETURN @y END</Recurso>
        <Recurso nombre="PROCRN#2" tipo="PROCEDURE">CREATE PROCEDURE PROCRN#2 (@anombreasig
          text,@agradoasig integer,@ecodigo integer) AS UPDATE cursa SET PROMEDIOTCP =
          DBO.promedioTCP ( @anombreasig, @agradoasig, @ecodigo ) WHERE
          (cursa.anombreasig=@anombreasig) AND (cursa.agradoasig=@agradoasig) AND
          (cursa.ecodigo=@ecodigo)</Recurso>
        <Recurso tipo="TRIGGER" nombre="TIRN#2_1" tabla="tcp">CREATE TRIGGER TIRN#2_1 ON tcp
          AFTER INSERT AS DECLARE @anombreasig text,@agradoasig integer,@ecodigo integer SET
          @anombreasig=(SELECT anombreasig FROM INSERTED) SET @agradoasig=(SELECT agradoasig
          FROM INSERTED) SET @ecodigo=(SELECT ecodigo FROM INSERTED) EXEC PROCRN#2
          @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN RAISERROR ( 'No se pudo
          completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END</Recurso>
        <Recurso tipo="TRIGGER" nombre="TURN#2_2" tabla="tcp">CREATE TRIGGER TURN#2_2 ON tcp
          AFTER UPDATE AS IF ( UPDATE (nota) ) BEGIN DECLARE @anombreasig text,@agradoasig
          integer,@ecodigo integer SET @anombreasig=(SELECT anombreasig FROM INSERTED) SET
          @agradoasig=(SELECT agradoasig FROM INSERTED) SET @ecodigo=(SELECT ecodigo FROM
          INSERTED) EXEC PROCRN#2 @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN
          RAISERROR ( 'No se pudo completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END
          END</Recurso>
        <Recurso tipo="TRIGGER" nombre="TDRN#2_3" tabla="tcp">CREATE TRIGGER TDRN#2_3 ON tcp
          AFTER DELETE AS DECLARE @anombreasig text,@agradoasig integer,@ecodigo integer SET
          @anombreasig=(SELECT anombreasig FROM DELETED) SET @agradoasig=(SELECT agradoasig
          FROM DELETED) SET @ecodigo=(SELECT ecodigo FROM DELETED) EXEC PROCRN#2
          @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN RAISERROR ( 'No se pudo
          completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END</Recurso>
      </Recursos>
    </lenguajeFormal>
    <lenguajeTecnico>El promedioTCP en cursa para PROMEDIOTCP es calculado como avg
      (sujeto.estudiante.cursa.asignatura.tcp.nota)</lenguajeTecnico>
    <lenguajeNatural>El promedio de los TCP en cursa para PROMEDIOTCP se calcula como el promedio de
      las notas de los TCP.</lenguajeNatural>
  </Regla>
```

Figura 24: Sección del repositorio de reglas para la regla de cálculo variante 2.

Para la cual el repositorio de generación es el siguiente:

```
- <Regla id="RN#2" gestor="sqlserver" complejidad="3" Tipo_retorno="double precision" sujeto="cursa"
    parámetros="_anombreasig text,_agradoasig integer,_ecodigo integer" atributo_llave="anombreasig,
    agradoasig, ecodigo" Tipo_atributo="double precision" atributo="PROMEDIOTCP" resultado="promedioTCP"
    tipo="calculus">
    <ExpresionSQL>(SELECT AVG( e.nota ) FROM cursa a , estudiante b , cursa c , asignatura d , tcp e WHERE
      (a.ecodigo=b.codigo) AND (b.codigo=c.ecodigo) AND (c.anombreasig=d.nombreasig) AND
      (c.agradoasig=d.gradoasig) AND (d.nombreasig=e.anombreasig) AND (d.gradoasig=e.agradoasig))
      </ExpresionSQL>
  - <ListaTablas>
      <Tabla Nombre="cursa"
        Llave_Macheo="anombreasig,nombreasig,asignatura/agradoasig,gradoasig,asignatura/ecodigo,codigo,estudiante/" />
      <Tabla Nombre="tcp"
        Llave_Macheo="anombreasig,nombreasig,asignatura/agradoasig,gradoasig,asignatura/ecodigo,codigo,estudiante/" />
    </ListaTablas>
  </Regla>
```

Figura 25: Sección del repositorio de Generación para la regla de cálculo variante 2.

Para la regla de clasificación se tiene este ejemplo:

```
- <Regla id="RN#8" estado="activa" tipo="clasification" sgbd="sqlserver" enfoque="inmediato"
    ultima_fecha="2012-6-12" version="1.0">
  - <lenguajeFormal>
    - <Recursos>
        <Recurso nombre="desaprobado" tipo="FUNCTION" tipo_Retorno="int">CREATE FUNCTION
          desaprobado ( @CODIGO int )RETURNS CHAR AS BEGIN DECLARE @x char; IF (( SELECT
          COUNT(*) FROM ESTUDIANTE sujeto WHERE ( (SELECT a.PROMEDIO FROM ESTUDIANTE a
          WHERE (sujeto.CODIGO=@CODIGO)) < 60)) !=0 ) SET @x='t'; ELSE SET @x='f'; RETURN @x;
          END</Recurso>
      </Recursos>
    </lenguajeFormal>
    <lenguajeTecnico>Un estudiante es definido como desaprobado si
      sujeto.promedio<60</lenguajeTecnico>
    <lenguajeNatural>Un estudiante es definido como desaprobado si su promedio es menor que
      60</lenguajeNatural>
  </Regla>
```

Figura 26: Sección del repositorio de reglas para la regla de clasificación.

Donde el repositorio de Generación contiene:

```
- <Regla id="RN#8" sujeto="ESTUDIANTE" parámetros="_CODIGO int" atributo_llave="CODIGO"
    complejidad="compleja" gestor="sqlserver" tipo="clasification">
    <ExpresionSQL>(SELECT a.PROMEDIO FROM ESTUDIANTE a) < 60</ExpresionSQL>
    <Clasificación>desaprobado</Clasificación>
  </Regla>
```

Figura 27: Sección del repositorio de Generación para la regla de clasificación.

La siguiente imagen muestra un ejemplo de la información del repositorio de regla para la regla de notificación.

```
- <Regla id="RN#19" estado="inactiva" ultima_fecha="2012-6-12" version="1.0" tipo="notification"
    enfoque="inmediato" sgbd="sqlserver">
  - <lenguajeFormal>
    - <Recursos>
        <Recurso tipo="TRIGGER" nombre="TIRN#19_1">CREATE TRIGGER TIRN#19_1 ON ESTUDIANTE
          FOR INSERT AS if(EXISTS(SELECT * FROM VRN#19 ) ) BEGIN raiserror('porciento de
          aprobados muy bajo',10,1); END</Recurso>
        <Recurso tipo="TRIGGER" nombre="TURN#19_1">CREATE TRIGGER TURN#19_1 ON
          ESTUDIANTE FOR UPDATE AS if(EXISTS(SELECT * FROM VRN#19 ) ) BEGIN raiserror
          ('porciento de aprobados muy bajo',10,1); END</Recurso>
        <Recurso tipo="VIEW" nombre="VRN#19">CREATE VIEW VRN#19 AS SELECT * FROM
          ESTUDIANTE WHERE ( ((100 * (SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE
          (a.PROMEDIO > 60 ))>0) / dbo.matricula()) < 65 );</Recurso>
      </Recursos>
    </lenguajeFormal>
    <lenguajeTecnico>Notificar 'porciento de aprobados muy bajo' si (100 * sizeof
      (estudiante.promedio> 60 ) / matricula)<65</lenguajeTecnico>
    <lenguajeNatural />
  </Regla>
```

Figura 28: Sección del repositorio de Reglas para la regla de notificación.

Para la cual el repositorio de Generación tiene lo siguiente:

```
- <Regla id="RN#19" complejidad="compleja" sujeto="ESTUDIANTE" gestor="sqlserver" tipo="notification">
    <ExpresionSQL>((100 * (SELECT COUNT( a.PROMEDIO ) FROM ESTUDIANTE a WHERE (a.PROMEDIO
      > 60 ))>0) / dbo.matricula()) < 65</ExpresionSQL>
    <Mensaje>'porciento de aprobados muy bajo'</Mensaje>
  - <ListaEventos>
    - <Evento tabla="ESTUDIANTE">
      - <TipoEvento>
          <Tipo>INSERT</Tipo>
          <Tipo>UPDATE</Tipo>
        </TipoEvento>
      </Evento>
    </ListaEventos>
  </Regla>
```

Figura 29: Sección del repositorio de Generación para la regla de notificación.

3.4 Características de la herramienta

En la versión 1.1 del traductor que se logra en este trabajo se mantiene la interfaz de usuario que se tenía en la versión anterior, agregándole los operadores del LTP para las reglas que se extienden. A continuación se describen las características fundamentales de la herramienta.

En la siguiente figura se muestra la ventana principal de la aplicación y se señalan los operadores que se añaden.

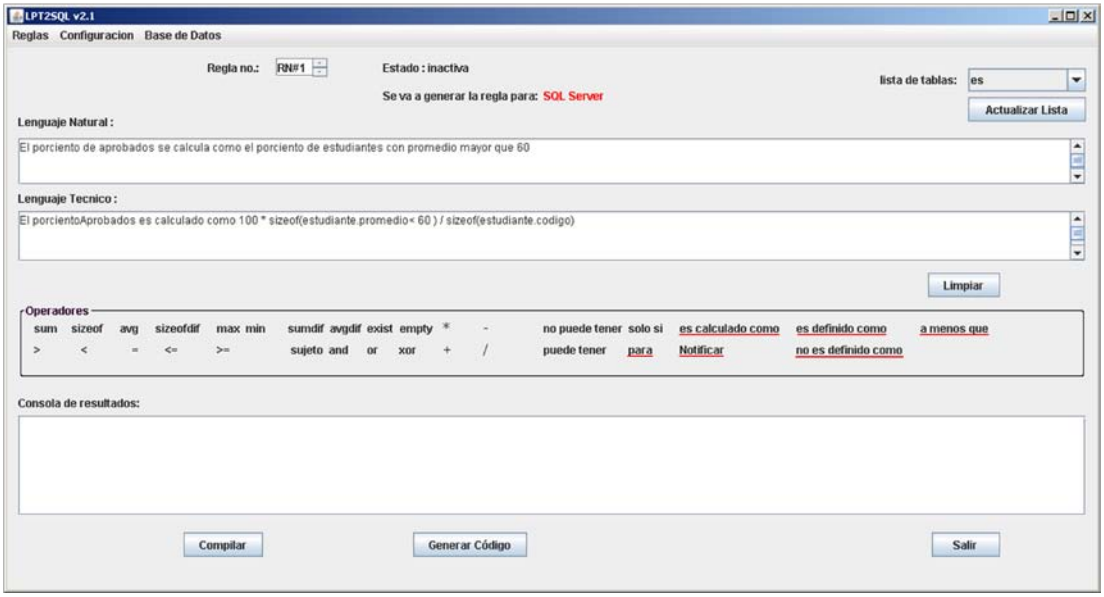


Figura 30: Ventana principal de la aplicación (subrayados los operadores que se agregan).

En esta herramienta se utiliza la aplicación para la extracción de información del catálogo, creada en (Díaz, 2010) y mejorada en (Calderón Solís, 2011). Para que la aplicación pueda ejecutarse correctamente es necesario contar con una base de datos física en la cual se van a implementar las reglas. En el caso de que la BD esté implementada en PostgreSQL, es necesario proveerle el servidor, nombre de

la BD, usuario y contraseña como se muestra en la Figura 31, si la BD está implementada en SQL Server es necesario crear y conectarse a origen de datos de la BD física. Para crear un origen de datos seguimos los siguientes pasos:

1. Acceder al Panel de Control, en Herramientas Administrativas

2. Seleccionar Orígenes de datos ODBC.

3. Seleccionar la opción Agregar (Para crear un nuevo origen de datos).

4. Seleccionar el driver ODBC para SQL Server.

5. Especificar el nombre para referirse al nuevo origen de datos.

6. Especificar el nombre del Servidor.

7. Especificar cómo desea que SQL Server compruebe la autenticidad del Identificador inicio de sesión.

8. Establecer la base de datos en cuestión como predeterminada.

9. Comprobar orígenes de datos.

Las Figuras 32 y 33 muestran las ventanas para crear un nuevo origen de datos y la comprobación de este al terminar de crearlo.

Figura 31: Interfaz de la herramienta InfoCatálogo (información necesaria para obtener los Metadatos en PostgreSQL).

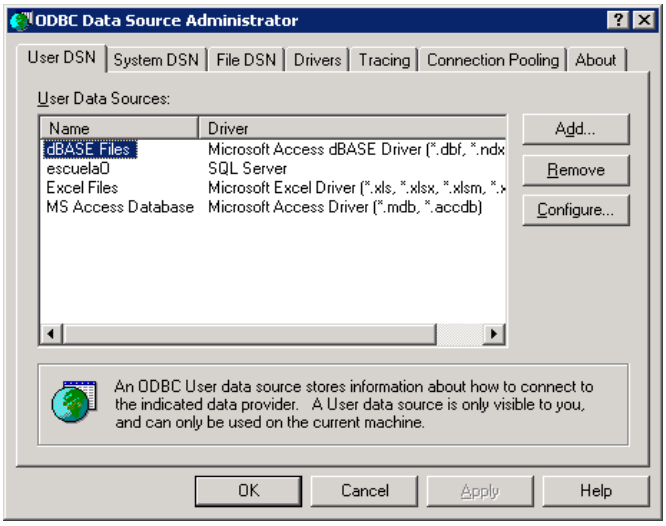


Figura 32: Crear un nuevo origen de datos ODBC.

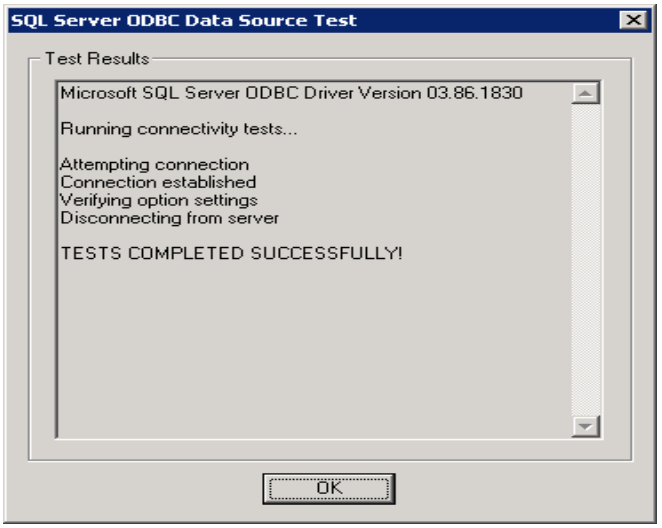


Figura 33: Comprobación del origen de datos.

Una vez creado el origen de datos de la BD, para obtener sus Metadatos se selecciona el gestor SQL Server y se le provee el nombre del origen como se muestra en la Figura 34:

Figura 34: Interfaz de la herramienta InfoCatálogo (información necesaria para obtener los Metadatos en SQL Server).

Al iniciar la aplicación se carga el archivo repositorios/Repositorio.xml que representa el repositorio de reglas; si no existe, crea un repositorio vacío. Igualmente se carga el archivo repositorios/Metadatos.xml. En caso que no exista se muestra la ventana para la extracción de información del catálogo que corresponde a la herramienta InfoCatálogo que se comentó anteriormente. Una vez que se logre la conexión y se obtenga la información del catálogo, no es necesario repetir estas operaciones mientras la ventana principal se encuentre activa.

La aplicación, cuya interfaz se muestra en la Figura 30, posee dos cajas de texto editables: una para la entrada de la regla en lenguaje natural (opcional) y otra para la entrada de la regla en LPT (obligatorio). En la caja de texto no editable situada en la parte inferior de la ventana se muestra el resultado de la compilación o generación de la regla. Para la edición de las reglas, se consta de una lista con todas las tablas de acuerdo a la información extraída del catálogo que se puede actualizar por el botó *Actualizar Lista*, y un marco con todos los operadores. Al seleccionar algún elemento de la lista desplegable o algún operador, el elemento seleccionado es situado en la caja de texto de representación de la regla en LPT. También posee los menús *Reglas*, *Configuración y Base de Datos.*

En el Menú de reglas que se muestra en la Figura 35 se tienen las opciones:

Figura 35: Menú de Reglas

- Nueva Regla: Prepara la aplicación para la adición de una nueva regla y le asigna un identificador a la regla.
- Adicionar Regla: Adiciona una nueva regla al repositorio de reglas.
- Actualizar Regla: Actualiza en el repositorio de reglas una regla inactiva. Si la regla está activa, esta opción se deshabilita.
- Desactivar Regla: Desactiva una regla en el repositorio de reglas y elimina los recursos que fueron implementados en la base de datos
- Activar Regla: Compila, genera e implementa en la base de datos los recursos obtenidos.

El menú de configuración que se muestra en la Figura 36 consta de:

- Información Catálogo: Muestra la ventana para la extracción de información del catálogo que se muestra en la Figura 31.
- Conexión: Ofrece la opción de conectarse a una base de datos
- Nuevo repositorio: Permite crear un nuevo repositorio vacío.

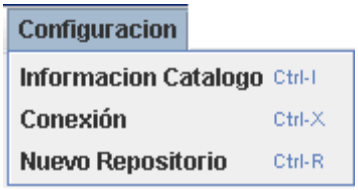


Figura 36: Menú de Configuración.

El menú de Base de datos permite seleccionar para cual gestor se va a generar las regla; SQL Server o PostgreSQL. Por defecto siempre esta seleccionado SQL Server.

También se tiene en la parte superior de la ventana el número de la regla en cuestión, el estado de la regla (activa o inactiva) y el gestor para el cual se va a generar.

El botón Limpiar borra el contenido de la caja de texto editable para la regla en LPT. El botón Compilar compila la regla y guarda en el repositorio de Generación, la información necesaria para generar la regla. El botón Generar, permite generar la regla y almacenarla en el repositorio.

3.5 Conclusiones parciales

En este capítulo se trataron las características principales de las reglas que se extienden en esta versión del Traductor, se presentaron los repositorios de reglas y generación con su correspondiente contenido para la implementación de cada una

de las reglas extendidas, se presentan las reglas que se detectan y se aplican para en la DB que se toma como caso de estudio, en lenguaje natural y en LPT y se describen los aspectos fundamentales de la aplicación y su uso.

Conclusiones

- Se revisaron las gramáticas de las reglas de cálculo, notificación y clasificación, y se modificó la correspondiente a la regla de cálculo.
- Se analizaron los recursos de BD propuestos para la representación formal de estas reglas, de los que se modificaron las implementaciones formales de las reglas de cálculo y la de notificación.
- Se adicionaron los artefactos de las nuevas reglas a los diagramas del diseño de la aplicación.
- Se añadieron los módulos necesarios a la herramienta logrando la versión 2.1 capaz de implementar las reglas de cálculo, clasificación y notificación, en adición a la de restricción que implementaba en su versión anterior.
- Se modificaron los repositorios de Regla y de Generación adaptándolos a las necesidades de las reglas que se implementan en esta versión de la herramienta.

Recomendaciones

- Aplicar la herramienta en una BD poblada sin afectar la integridad del negocio.
- Implementar algoritmos de validación de las reglas.
- Resolver las interdependencias entre las reglas para su eliminación.
- Resolver las interdependencias de las reglas de notificación con las de clasificación y de cálculo.
- Implementar la regla de Clasificación en función del valor de un atributo, o de una tabla de la base de datos.

Bibliografía

ANALYTICS, K. 2004. *Semantics of Business Vocabulary and Business Rules (SBVR)* [Online]. [Accessed http://www.kdmanalytics.com/sbvr/ 2011].

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.

BUSINESS RULE GROUP. 2009. *Business Rule Group, a non-commercial peer group of IT professionals.* [Online]. Available: http://www.businessrulesgroup.org/home-brg.shtml [Accessed 2011].

BUSINESS RULES GROUP 2003. Manifiesto de Reglas de Negocio. *In:* ROSS, R. G. (ed.). Business Rules Group.

CALDERÓN SOLÍS, A. 2011. *Traductor LPT-SQL para reglas de negocio en bases de datos relacionales.* Diploma, Universidad Central "Marta Abreu" de Las Villas.

COTI COLOP, B. M. 2003. *Reglas de negocio en arquitectura de tres capas.* Diploma, Universida de San Carlos de Guatemala.

DATE, C. J. 2000. *Introducción a los Sistemas de Bases de Datos, Séptima edición,* México, Addison-Wesley.

DÍAZ, R. 2010. *Información del catálogo para generar reglas de negocio.* Universidad Central de Las Villas.

GONZÁLEZ, L. 2010. Sistemas de Bases de Datos postrelacionales.Triggers

HALLE, B. V. 2002. *Business Rules Applied.*

HEALY, K. A. 2000. Defining business rules using triggers. Available: http://publib.boulder.ibm.com/infocenter/db2luw/v8//topic/com.ibm.db2.udb.doc/ad/t0006686.htm.

HERNÁNDEZ, E. H. 2007. Compilación II.

IAN HORROCKS, PETER F. PATEL-SCHNEIDER, HAROLD BOLEY, SAID TABET, BENJAMIN GROSOF & DEAN, M. 2004. *SWRL: A Semantic Web Rule Language*

Combining OWL and RuleML [Online]. [Accessed http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/ 2011].

MARTÍNEZ FERNÁNDEZ, J. L. 2010. *Introduciendo Semántica en un Proceso de Desarrollo Software a través de Reglas de Negocio.* Doctorado, Universidad Politécnica de Madrid.

MATEI, I. 2006. *Implementing Business Rules with Software Agents.* University of Tampere.

MELTON, J. & SIMON, A. R. 2002. *SQL1999: underestanding relational languaje components*, Morgan Kaufmann.

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

MSDN 2008. MSDN LIBRARY VisualStudio 2008. Microsoft Corporation.

PEREIRA TOLEDO, A. 2009. *Solución al problema de la cardinalidad en la generación automática de reglas de negocio en bases de datos relacionales.* Licenciado, Universidad Central "Marta Abreu" de Las Villas.

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

PÉREZ ALONSO, A. 2008. *Aplicación para reglas de restriccion en negocios.* Diploma, Universidad Central "Marta Abreu" de Las Villas.

ROSS, R. G. 2000. *WHAT IS A 'BUSINESS RULE'?* [Online]. Available: www.brcommunity.com/b005.php.htm [Accessed 2011].

ROSS, R. G. 2010. What Is a Business Rule? *Business Rules Journal,* Vol. 11, No. 3.

SCOTT W., A. 2011. *Business Rules* [Online]. Available: http://www.agilemodeling.com/artifacts/businessRule.htm [Accessed 2011].

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' algoritmo| determinante resultado 'en' sujeto 'es calculado como' algoritmo | determinante resultado 'en' sujeto 'para' atributo 'es calculado como' algoritmo

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

algoritmo::= exp_valor

resultado::=string

exp ::= L(operador_logico , 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::=‘.’
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);
string_dec::= '"'string'"'|'\"string'\";
real::= N(digito) O('.'N(digito)) O(('e' |'e')('+'|'-') N(digito));
operador_aritmetico ::="+" | " -" | " *" | " /"

Anexo 2: Semántica del LPT.

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

Semántica:

Establece una restricción sobre el sujeto de la regla.

Ejemplo:

Un estudiante no puede tener sujeto.promedio>100

Semántica de los elementos:

determinante: Es el determinante para cada sujeto, por ejemplo: Una, Uno, El, La, Cada, Todos. Según el mejor sentido en la redacción.

sujeto: Es una tabla en la Base de Datos del negocio o una clasificación de la misma.

Ejemplo:

Estudiante

características : Describe las características del sujeto en el negocio, tanto internas como relacionadas con otras tablas. Pueden incluir hechos con el fin de caracterizar al *sujeto*.

Ejemplo:

sujeto.promedio>100

hechos: Hechos relativos al estado o comportamiento de la Base de Datos del negocio, incluyendo o no al sujeto.

Ejemplo:

sizeof(sujeto.ef.notaef) ***> 2***

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

Semántica:

Clasifica a un sujeto según determinadas características del mismo.

Ejemplos:

Un estudiante es definido como desaprobado si sujeto.promedio<60

Semántica de los elementos:

clasif : Definición de un término del negocio. Típicamente define el valor de un atributo o un subconjunto de objetos en una clase existente

Ejemplo:

Estudiante

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

Semántica:

Informa a los usuarios autorizados del negocio sobre algún conocimiento básico en tiempo real.

Ejemplos:

Notificar ‘porciento de aprobados muy bajo’ si

sizeof(estudiante.promedio)*100/sizeof(estudiante.codigo)<65

Semántica de los elementos:

mensaje: Mensaje de información entre comillas para usuarios autorizados del negocio.

Ejemplo:

‘Porciento de aprobados muy bajo’

computo ::=determinante resultado ‘es calculado como’ algoritmo| determinante resultado ‘en’ sujeto ‘es calculado como’ algoritmo| determinante resultado ‘en’ sujeto ‘para’ atributo ‘es calculado como’ algoritmo

Semántica:

Calcula un valor determinado en el negocio, asociado o no al sujeto y su resultado es numérico.

Ejemplos:

El promedio es calculado como

avg(estudiante.es.notaes)+avg(estudiante.tcp.notatcp)+

max(estudiante.ef.notaef)

Semántica de los elementos:

algoritmo:

Definición de una expresión matemática para obtener el valor de un resultado; normalmente expresada utilizando combinaciones de términos del negocio junto a constantes disponibles.

atributo: atributo de la tabla sujeto.

Ejemplo:

avg(estudiante.es.notaes)+avg(estudiante.tcp.notatcp)+

max(estudiante.ef.notaef)

resultado: Cualquier valor, no necesariamente numérico, que tiene algún significado en el negocio. El resultado es usualmente el valor del atributo de un objeto del negocio.

Ejemplo:

promedio

exp ::= L(operador_logico , termino)

Semántica:

Expresa una proposición lógica entre términos booleanos.

Operador	**Significado**	**Argumentos/Resultado**
OR	Verdadero si alguna expresión booleana es verdadera.	Booleano
AND	Verdadero si ambas expresiones booleanas son verdaderas.	Booleano
NOT	Negación de cualquier valor booleano devuelto por otro operador booleano.	Un único argumento Booleano
XOR	(a or not b) and (not a or b)	Booleano

Ejemplo:

sujeto.edad < 18 or sujeto.edad > 55;

exp_valor::=L(operador_aritmetico , atomo)

Semántica:

Los operadores aritméticos son muy útiles para las reglas de tipo cómputo, aunque pueden ser ampliamente utilizados en el contexto de otras reglas.

Operador	**Significado**	**Argumentos/Resultado**
+	Adición	Numéricos
-	Resta	Numéricos
*	Multiplicación	Numéricos
/	División	Numéricos

Ejemplos:

sujeto.edad + 30

(sujeto.Pulso / sujeto.Edad)+10

sizeof(sujeto.promedio > 60) * 100

exp_conjunto:=L (operador_comp, exp_valor)

Semántica:

Compara entre sí los valores de dos expresiones de acuerdo a un operador. Ambas expresiones deben tener el mismo tipo. Pueden utilizarse indistintamente para comparar elementos individuales y múltiples.

Operador	**Significado**	**Argumentos**	**Resultado**
=	Igual a	Numéricos/Booleanos/Cadenas	Booleano
>	Mayor que	Numéricos	Booleano
<	Menor que	Numéricos	Booleano
>=	Mayor igual a	Numéricos	Booleano
<=	Menor igual a	Numéricos	Booleano
<>	Diferente a	Numéricos/Booleanos/Cadenas	Booleano

Ejemplos:

sujeto.cursa.asignatura.nombreAsig <> 'historia'

sujeto.nota >20

exp_logica ::='exists' '(' exp_elemental' ,' exp_conjunto ')' | "empty " „(" exp_conjunto „)"

Semántica (exist):

Retorna verdadero si un elemento existe en una colección; en otro caso devuelve falso. La expresión debe representar un valor y debe ser del mismo tipo de los elementos del conjunto.

Semántica (empty):

Retorna verdadero si la colección no contiene elementos.

Ejemplos:

exist(sujeto.CI.Paciente(ciudad="Santa Clara").CI)

empty(sujeto.DonantePotencial)

exp_elemental::=operador_conjunto'('exp_conjunto')' | numero | real | booleano | string_dec | calculo '('sujeto')' |camino

Semántica (operador conjunto):

Operan sobre una colección de elementos y devuelven un valor.

Operadores	**Significado**
SIZEOF <colección>	Retorna cuántos elementos contiene la colección de elementos.
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.
AVGDIF <colección>	Retorna el promedio de los elementos diferentes de una colección numérica.
SIZEOFDIF <colección>	Retorna cuántos elementos diferentes contiene la colección de elementos.
SUMDIF <colección>	Retorna la suma de los elementos diferentes de una colección numérica.

Ejemplos:
sizeof(sujeto.promedio) > 5
max(sujeto.ef.notaEF) > 20
camino::=’sujeto’ O(punto camino_navegacion) | camino_navegacion
Semántica:
Establece el medio de acceso a los atributos de tablas y posibilita la navegación. Puede terminar en una tabla o un atributo. Es admisible emplear como tabla la palabra reservada *sujeto* al inicio del camino, la cual no pertenece realmente a ninguna tabla específica, su utilización referencia al sujeto de la regla.
Ejemplos:
sujeto
sujeto .grado
estudiante.TCP.nota
elemento_navegacion:= nombre_tabla O(‘(‘ macheo ‘)’)
Semántica:
Representa una tabla de la base de datos. Si se especifica algún *macheo*, se restringe el valor de algún atributo perteneciente a una tabla que pertenezca al camino de navegación. En el caso que se quiera restringir a partir de la llave primaria, puede omitirse el atributo. Esto no es posible si la llave primaria es compuesta.
Estudiante

Estudiante(nombre=”José Luis”)

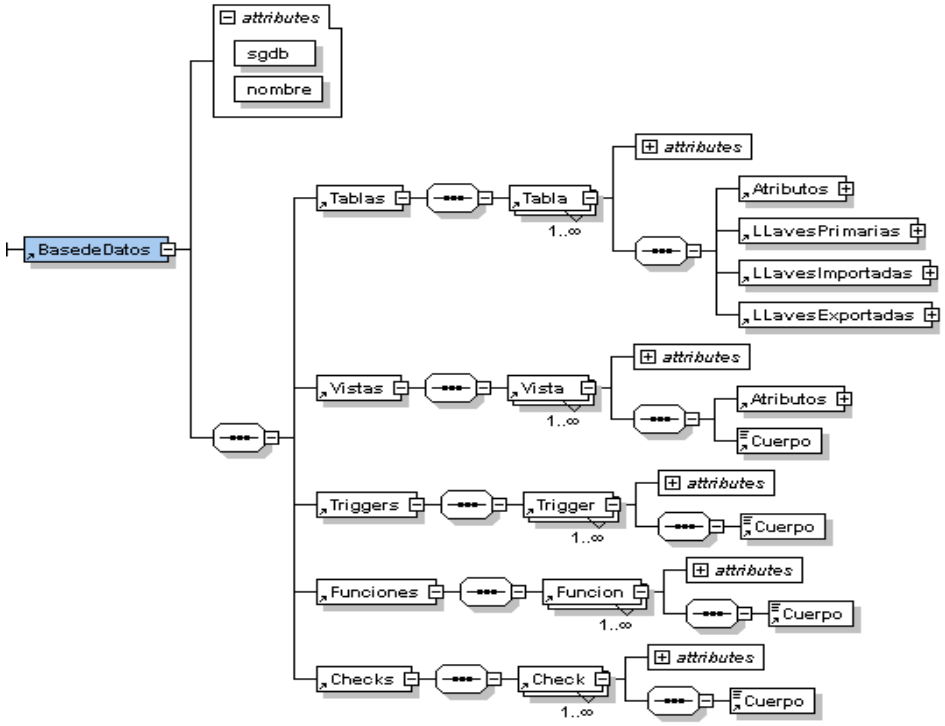
Hospital(2).

Tipos de datos:

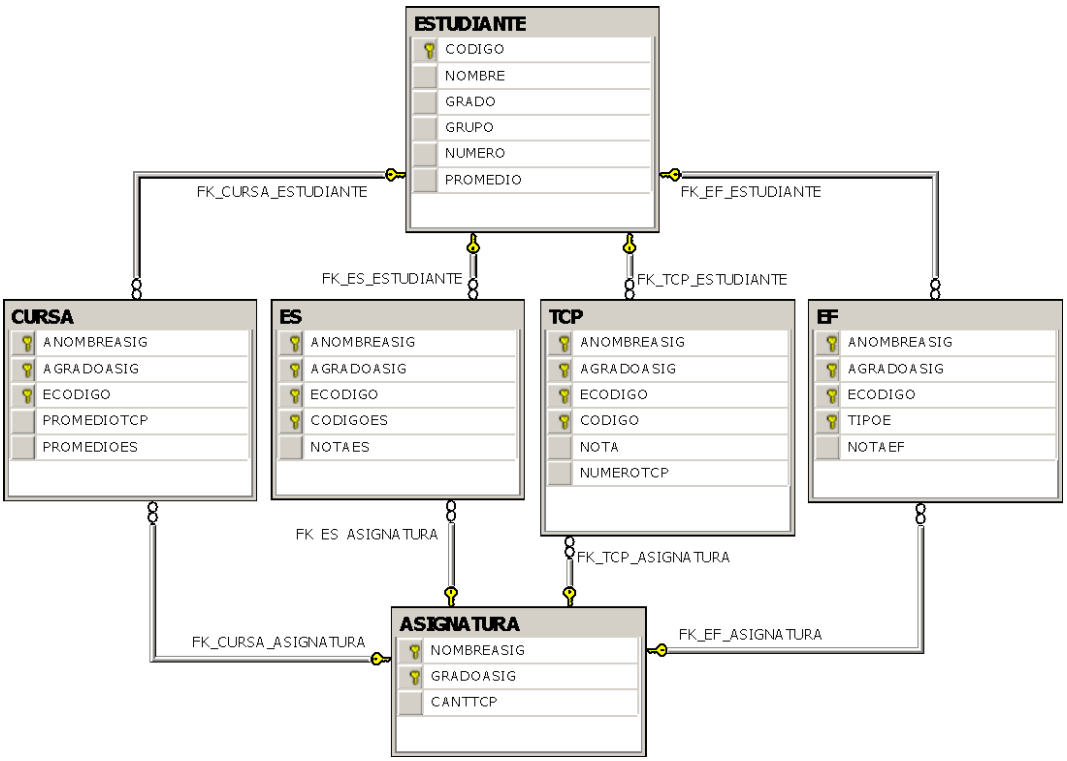
Para el LPT se han definido cuatro tipos de datos fundamentales:

Tipo	**Valores**
Booleano	True, False
Cadena	“Esto es una cadena”
Entero	-2, -1, 0, 1, 2, 3…
Real	-0.5, 0.25, 0.5, 0.75, 1.25

Anexo 3: Esquema XSD del Repositorio de Metadatos.



Anexo 4: Diagrama Relacional de la BD para el control de la evaluación estudiantil en la ESBU Héctor Martínez Valladares:



Anexo 5: Implementación formal algunas de las reglas que se tomaron como caso de estudio.

RN#11

El promedioEvalSist en cursa para promedioes es calculado como avg(sujeto.ASIGNATURA.ES.notaes)

```
CREATE FUNCTION promedioES (@anombreasig text,@agradoasig
integer,@ecodigo integer) RETURNS float AS BEGIN DECLARE @y
float SET @y = (SELECT AVG( e.notaes ) FROM cursa a ,
estudiante b , cursa c , asignatura d , es e WHERE
(a.ecodigo=b.codigo) AND (b.codigo=c.ecodigo) AND
(c.anombreasig=d.nombreasig) AND (c.agradoasig=d.gradoasig)
AND (d.nombreasig=e.anombreasig) AND
(d.gradoasig=e.agradoasig))  RETURN @y END


CREATE PROCEDURE PROCRN#4 (@anombreasig text,@agradoasig
integer,@ecodigo integer) AS UPDATE cursa  SET PROMEDIOES =
DBO.promedioES ( @anombreasig, @agradoasig, @ecodigo )  WHERE
(cursa.anombreasig=@anombreasig) AND
(cursa.agradoasig=@agradoasig) AND (cursa.ecodigo=@ecodigo)


CREATE TRIGGER TIRN#4_1 ON es AFTER INSERT  AS DECLARE
@anombreasig text,@agradoasig integer,@ecodigo integer SET
@anombreasig=(SELECT anombreasig FROM INSERTED) SET
@agradoasig=(SELECT agradoasig FROM INSERTED) SET
@ecodigo=(SELECT ecodigo FROM INSERTED)     EXEC PROCRN#4
@anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN
RAISERROR ( 'No se pudo completar el cálculo', 16, 1 )
ROLLBACK TRANSACTION END

CREATE TRIGGER TURN#4_2 ON es AFTER UPDATE  AS IF (  UPDATE
(notaes)  ) BEGIN DECLARE @anombreasig text,@agradoasig
integer,@ecodigo integer SET @anombreasig=(SELECT anombreasig
FROM INSERTED) SET @agradoasig=(SELECT agradoasig FROM
INSERTED) SET @ecodigo=(SELECT ecodigo FROM INSERTED)  EXEC
PROCRN#4 @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0
```

```
) BEGIN RAISERROR ( 'No se pudo completar el cálculo', 16, 1 )
ROLLBACK TRANSACTION END END

CREATE TRIGGER TDRN#4_3 ON es AFTER DELETE  AS DECLARE
@anombreasig text,@agradoasig integer,@ecodigo integer SET
@anombreasig=(SELECT anombreasig FROM DELETED) SET
@agradoasig=(SELECT agradoasig FROM DELETED) SET
@ecodigo=(SELECT ecodigo FROM DELETED)  EXEC PROCRN#4
@anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN
RAISERROR ( 'No se pudo completar el cálculo', 16, 1 )
ROLLBACK TRANSACTION END
```

RN#12

El promedioTCP en cursa para PROMEDIOTCP es calculado como avg(sujeto.estudiante.cursa.asignatura.tcp.nota)

```
CREATE FUNCTION promedioTCP (@anombreasig text,@agradoasig
integer,@ecodigo integer) RETURNS float AS BEGIN DECLARE @y
float SET @y = (SELECT AVG( e.nota ) FROM cursa a , estudiante
b , cursa c , asignatura d , tcp e WHERE (a.ecodigo=b.codigo)
AND (b.codigo=c.ecodigo) AND (c.anombreasig=d.nombreasig) AND
(c.agradoasig=d.gradoasig) AND (d.nombreasig=e.anombreasig)
AND (d.gradoasig=e.agradoasig)) RETURN @y END

CREATE PROCEDURE PROCRN#2 (@anombreasig text,@agradoasig
integer,@ecodigo integer) AS UPDATE cursa  SET PROMEDIOTCP =
DBO.promedioTCP ( @anombreasig, @agradoasig, @ecodigo ) WHERE
(cursa.anombreasig=@anombreasig)  AND
(cursa.agradoasig=@agradoasig) AND (cursa.ecodigo=@ecodigo)

CREATE TRIGGER TIRN#2_1 ON tcp AFTER INSERT  AS DECLARE
@anombreasig text,@agradoasig integer,@ecodigo integer SET
@anombreasig=(SELECT anombreasig FROM INSERTED) SET
@agradoasig=(SELECT agradoasig FROM INSERTED) SET
```

```
@ecodigo=(SELECT ecodigo FROM INSERTED)      EXEC PROCRN#2 @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN RAISERROR ( 'No se pudo completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END
```

```
CREATE TRIGGER TURN#2_2 ON tcp AFTER UPDATE  AS IF (  UPDATE (nota)     ) BEGIN DECLARE @anombreasig text,@agradoasig integer,@ecodigo integer SET @anombreasig=(SELECT anombreasig FROM INSERTED) SET @agradoasig=(SELECT agradoasig FROM INSERTED) SET @ecodigo=(SELECT ecodigo FROM INSERTED)   EXEC PROCRN#2 @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN RAISERROR ( 'No se pudo completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END END
```

```
CREATE TRIGGER TDRN#2_3 ON tcp AFTER DELETE   AS DECLARE @anombreasig text,@agradoasig integer,@ecodigo integer SET @anombreasig=(SELECT anombreasig FROM DELETED) SET @agradoasig=(SELECT agradoasig FROM DELETED) SET @ecodigo=(SELECT ecodigo FROM DELETED)      EXEC PROCRN#2 @anombreasig, @agradoasig, @ecodigo IF ( @@error <> 0 ) BEGIN RAISERROR ( 'No se pudo completar el cálculo', 16, 1 ) ROLLBACK TRANSACTION END
```

Implementación formal de algunas de las reglas analizadas en PostgreSQL en sus tres niveles de expresión.

- El porciento de aprobados se calcula como el porciento de estudiantes con promedio mayor que 60

RN#1

El porcientoAprobados es calculado como 100 * sizeof(estudiante.promedio< 60) / sizeof(estudiante.codigo)

```
CREATE OR REPLACE FUNCTION porcientoAprobados () RETURNS float AS $$ DECLARE y float; BEGIN  (100 * (SELECT COUNT(
```

```
a."promedio" )  ::float  into y FROM "estudiante" a WHERE
(a."promedio" < 60 ))) / (SELECT COUNT( a."codigo" )  ::float
into y FROM "estudiante" a) ; RETURN y; END;$$ LANGUAGE
plpgsql
```

- El promedio en estudiante para promedio se calcula como la suma del promedio de TCP más el promedio de ES más el máximo de las EF.

RN#6

El promedio en estudiante para promedio es calculado como avg(sujeto.es.notaes)+avg(sujeto.tcp.nota)+max(sujeto.ef.notaef)

```
CREATE OR REPLACE FUNCTION promedio (codigo integer) RETURNS
float AS $$ DECLARE y float; BEGIN y = ( ((SELECT AVG(
b."notaes" ) FROM "estudiante" a , "es" b WHERE
(a."codigo"=b."ecodigo")) + (SELECT AVG( b."nota" ) FROM
"estudiante" a , "tcp" b WHERE (a."codigo"=b."ecodigo"))) +
(SELECT MAX( b."notaef" ) FROM "estudiante" a , "ef" b WHERE
(a."codigo"=b."ecodigo")) ) ; RETURN y; END;$$ LANGUAGE
plpgsql;
;
CREATE OR REPLACE FUNCTION PROCRN_6 (codigo integer) RETURNS
void AS $$ BEGIN UPDATE estudiante  SET promedio = (SELECT
promedio ( $1 ) )  WHERE (estudiante.codigo=$1); END; $$
LANGUAGE plpgsql;
;
CREATE OR REPLACE FUNCTION FTIRN6_1() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  NEW.ecodigo ) ; RETURN NEW;
EXCEPTION WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se
pudo completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TIRN6_1 AFTER INSERT  ON "es" FOR EACH ROW
EXECUTE PROCEDURE FTIRN6_1();
;
CREATE OR REPLACE FUNCTION FTURN6_2() RETURNS trigger AS $$
BEGIN IF( NEW."notaes" <> OLD."notaes" ) THEN EXECUTE PROCRN_6
(  NEW.ecodigo ); END IF; RETURN NEW; EXCEPTION WHEN SQLSTATE
```

```
'39000' THEN RAISE EXCEPTION 'No se pudo completar el
cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TURN6_2 AFTER UPDATE  ON "es" FOR EACH ROW
EXECUTE PROCEDURE FTURN6_2();
;
CREATE OR REPLACE FUNCTION FTDRN6_3() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  OLD.ecodigo ); RETURN NEW; EXCEPTION
WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se pudo
completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TDRN6_3 AFTER DELETE  ON "es" FOR EACH ROW
EXECUTE PROCEDURE FTDRN6_3();
;
CREATE OR REPLACE FUNCTION FTIRN6_4() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  NEW.ecodigo ) ; RETURN NEW;
EXCEPTION WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se
pudo completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TIRN6_4 AFTER INSERT  ON "tcp" FOR EACH ROW
EXECUTE PROCEDURE FTIRN6_4();
;
CREATE OR REPLACE FUNCTION FTURN6_5() RETURNS trigger AS $$
BEGIN IF( NEW."nota" <> OLD."nota" ) THEN EXECUTE PROCRN_6 (
NEW.ecodigo ); END IF; RETURN NEW; EXCEPTION WHEN SQLSTATE
'39000' THEN RAISE EXCEPTION 'No se pudo completar el
cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TURN6_5 AFTER UPDATE  ON "tcp" FOR EACH ROW
EXECUTE PROCEDURE FTURN6_5();
;
CREATE OR REPLACE FUNCTION FTDRN6_6() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  OLD.ecodigo ); RETURN NEW; EXCEPTION
WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se pudo
completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
```

```
CREATE TRIGGER TDRN6_6 AFTER DELETE  ON "tcp" FOR EACH ROW
EXECUTE PROCEDURE FTDRN6_6();
;
CREATE OR REPLACE FUNCTION FTIRN6_7() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  NEW.ecodigo ) ; RETURN NEW;
EXCEPTION WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se
pudo completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TIRN6_7 AFTER INSERT  ON "ef" FOR EACH ROW
EXECUTE PROCEDURE FTIRN6_7();
;
CREATE OR REPLACE FUNCTION FTURN6_8() RETURNS trigger AS $$
BEGIN IF( NEW."notaef" <> OLD."notaef" ) THEN EXECUTE PROCRN_6
(  NEW.ecodigo ); END IF; RETURN NEW; EXCEPTION WHEN SQLSTATE
'39000' THEN RAISE EXCEPTION 'No se pudo completar el
cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TURN6_8 AFTER UPDATE  ON "ef" FOR EACH ROW
EXECUTE PROCEDURE FTURN6_8();
;
CREATE OR REPLACE FUNCTION FTDRN6_9() RETURNS trigger AS $$
BEGIN EXECUTE PROCRN_6 (  OLD.ecodigo ); RETURN NEW; EXCEPTION
WHEN SQLSTATE '39000' THEN RAISE EXCEPTION 'No se pudo
completar el cálculo'; END; $$ LANGUAGE plpgsql;
;
CREATE TRIGGER TDRN6_9 AFTER DELETE  ON "ef" FOR EACH ROW
EXECUTE PROCEDURE FTDRN6_9();
;
```

- Un estudiante es definido como desaprobado si su promedio es menor que 60

RN#8

Un estudiante es definido como desaprobado si sujeto.promedio<60

```
CREATE OR REPLACE FUNCTION desaprobado ( codigo integer )
RETURNS BOOLEAN AS $$ BEGIN RETURN (( SELECT COUNT(*) FROM
```

```
estudiante sujeto WHERE ( (SELECT a."promedio" FROM
"estudiante" ) AND (sujeto.codigo=$1)) < 60))!=0 ); END $$
LANGUAGE plpgsql
```

- Notificar estudiante mal en ES si el promedio de sus ES es menor que 30

RN#11

Notificar 'estudiante mal en ES' si avg(estudiante.es.notaes)<30

```
CREATE OR REPLACE FUNCTION FRN11_1() RETURNS trigger AS $$
BEGIN if(EXISTS(SELECT * FROM VRN11 ) ) then RAISE NOTICE
'estudiante mal en ES'; END IF; RETURN NEW; END;  $$ LANGUAGE
plpgsql;
;
CREATE TRIGGER TIRN11_1 AFTER  INSERT  ON es FOR EACH ROW
EXECUTE PROCEDURE FRN11_1();
;
CREATE TRIGGER TURN11_1 AFTER  UPDATE  ON es FOR EACH ROW
EXECUTE PROCEDURE FRN11_1();
;
CREATE VIEW VRN11 AS SELECT * FROM "estudiante" WHERE (
(SELECT AVG( b."notaes" ) FROM "estudiante" a , "es" b WHERE
(a."codigo"=b."ecodigo")) < 30 );
;
```