Universidad Autónoma de Baja California
Facultad de Contaduría y Administración
Facultad de Ciencias Químicas e Ingeniería
Maestría en Tecnologías de la Información y la Comunicación

Implementación de Operadores de Modificación de Esquema para el Versionamiento de Bases de Datos Empresariales en Microsoft SQL Server

TESIS

Para Obtener el Grado de

Maestro en Tecnologías de la Información y la Comunicación

Presenta:

Betsabé Noemí Alanis Sikisaka

Bajo la dirección de:

Dr. Manuel Castañón Puga

Co-dirigido por:

Dra. Margarita Ramírez Ramírez

Tijuana, Baja California, México

Mayo del 2014

A mi esposo por su paciencia, comprensión y tan necesaria compañía durante estos últimos dos años.

Betsy

Agradecimientos

Deseo agradecer a la Universidad Autónoma de Baja California (UABC), por darme la oportunidad de ingresar al Programa de Maestría en Tecnologías de la Información y la Comunicación (MTIC).

A la Coordinación de Cooperación Internacional e Intercambio Académico por darme la oportunidad de viajar y realizar parte de mis estudios en el extranjero.

A la Dra. Margarita Ramírez Ramírez, por estar siempre pendiente del bienestar de los alumnos del programa y la gestión académica y administrativa, así como su apoyo en la co-dirección de este trabajo de tesis.

A mi director de tesis, Dr. Manuel Castañon Puga por su tiempo, apoyo, paciencia, ayuda y guía para que mi trabajo fuera concluido con éxito.

A la empresa Punto de Desarrollo de Software S.A. de C.V. (*SDPoint*), por permitirme utilizar sus productos como caso de estudio en la investigación preliminar y pruebas de la metodología propuesta en este documento.

A mi jefe, Edaniel Figueroa García, por darme la oportunidad de continuar con mis estudios de posgrado, permitirme laborar en la empresa, brindarme el apoyo económico para subsidiar el costo de los estudios, y confiar su conocimiento sobre las problemáticas que acontecen en la labor diaria de las pequeñas empresas.

Resumen

Este documento propone una solución para controlar las versiones de una base de datos que se ha modificado, incluso después de haber sido distribuida y utilizada por usuarios a través de un sistema informático que ha evolucionado con el tiempo.

Carlo A. Curino, Hyun J. Moon y Carlo Zaniolo propusieron el concepto de *Schema Modification Operators* (SMO) y desarrollaron el sistema PRISM. Los operadores SMO que fueron tomados como base en este trabajo para crear una estructura de metadatos que contiene la información apta para describir el estado de cada una de las versiones de una base de datos, y permite diseñar una estructura base para las bases de datos de desarrollo, así como aquellas que pasan a producción, permitiendo a la propia base de datos ser capaz de moverse entre versiones.

Como caso de estudio se indentificó en una empresa mexicana ubicada en la ciudad de Tijuana, en la cual el proceso de actualización de sus productos de software se volvió cada vez más complejos en la medida que sus sistemas eran utilizados por más usuarios. Se analizó la situación actual de la organización y se planteó una estrategia de versionamiento basada en SMO, con la finalidad de medir la viabilidad y funcionalidad de la metodología propuesta.

Abstract

This paper proposes a solution to control the versions of a database that has been modified, even after being distributed and used by users through a computer system that has evolved over time.

Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo proposed the concept *Schema Modification Operators* (SMO) and developed the PRISM system. By taking the SMO operators as a basis in this work to create a structure of metadata containing suitable information to describe the state of each version of a database, allows us to design a database structure for development databases, as well as production databases, allowing the database itself be able to move between versions.

As a case study, a mexican company located in the city of Tijuana was identified, in which the process of updating its software products became increasingly complex as their systems were used by more users. The current status of the organization was analyzed, and versioning strategy based SMO was proposed, in order to measure the feasibility and practicality of the proposed methodology.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El presente documento describe una metodología diseñada para cubrir las necesidades de evolución del software con respecto a las bases de datos, junto con el desarrollo de las herramientas de soporte necesarias para navegar de una versión a otra.

Resume, como caso de estudio, la experiencia de la implementación de la metodología propuesta en los productos de software de una empresa local.

1.1. Antecedentes

Los sistemas de información surgen por la necesidad, no solo de modernizarse, sino también de automatizar, controlar y agilizar procesos en las instituciones de cualquier rubro, o actividad. En la actualidad podemos encontrar aplicaciones tanto gratuitas como comerciales que fueron desarrolladas como auxiliares para realizar distintas diligencias, pero, ¿Qué hay detrás de toda esta industria de software? ¿Cuáles son las implicaciones para desarrollar una aplicación?. Si bien es cierto que los sistemas de información fueron diseñados para cubrir una necesidad y agilizar procesos, ¿Qué elementos lo componen?.

El software está compuesto principalmente de procesos que son codificados a través de un lenguaje de programación y que permiten manipular datos para convertirlo información que comúnmente son almacenados mediante un motor de bases de datos, estos elementos trabajan en conjunto para crear un todo funcional y eficiente. Esto indica que detrás del desarrollo de software hay todo un proceso para la codificación y control de dicha aplicación pero también del lugar físico y lógico en donde será almacenada y gestionada la información que es procesada dentro del sistema y esto a su vez también sigue estándares y normas de diseño, así como el control de su desarrollo.

Sin embargo, debido a la continua y perpetua naturaleza evolutiva del software provocada por nuevas necesidades de los usuarios, cambios a las disposiciones de ley, mejoras en el rendimiento y la funcionalidad de los sistemas, entre otras razones, es que se crearon diferentes herramientas y mecanismos basados en diferentes metodologías para administrar dicha evolución, controlar el que, el quien y el tiempo en que se realizaron cambios en el código y de esta forma auxiliar al equipo de desarrollo en la gestión eficiente y ordenada cada una de las versiones de la aplicación.[1]

No obstante, muchos problemas surgen con el paso evolutivo del desarrollo de aplicaciones, sobre todo cuando hablamos de lo que implica dicho cambio en la estructura física y lógica del almacenamiento de la información. [2]

La masificación de uso de un software está directamente relacionada con la cantidad de instalaciones que luego deberán ser mantenidas y administradas. En este contexto, el tiempo de las actualizaciones (planeadas o no) que deben realizarse de forma manual y de manera secuencial por un implementador experto, ésto se convierte en un factor bloqueante para los procesos de los clientes que usan los sistemas de información, y un punto crítico para el éxito y crecimiento de la empresa que los provee.

Cuando hablamos de administración de datos nos encontramos con distintos manejadores

de bases de datos bajo licenciamiento pero también de distribución gratuita, como lo son MS SQL, Oracle y MySQL entre otros.

Este tipo de herramientas son sin duda de gran ayuda a la hora de diseñar y modelar el entorno en donde la información estará almacenada, su contenido, cada uno de sus atributos, índices, llaves, etc. Pero se encuentran limitados en cuanto al soporte de la evolución lógica del esquema de bases de datos, dejando a los administradores de bases de datos (DBA), y desarrolladores de software, la tarea de gestionar la evolución de la base de datos, en ocasiones poco ordenada e ineficiente de las actualizaciones tanto de la aplicación como de la base de datos.

Esto crea una necesidad de simplificar y automatizar el proceso de evolución de la base de datos, con la finalidad de asegurar su independencia lógica sobre los cambios del esquema [3], disminuir el costo generado por el tiempo durante el desplegado (deployment) y actualización de la base de datos.[4]

1.1.1. Tecnologías Disponibles

En este apartado se describen algunas tecnologías que se encuentran, tanto en el mercado, como en línea de investigación que contemplan el versionamiento de esquemas de bases de datos, que utilizan distintas tecnologías y metodologías en particular y son descritos de forma breve a continuación:

Rails: Es un marco de trabajo de código abierto para el desarrollo de aplicaciones en Ruby, el cual posee características que permiten al desarrollador administrar la evolución de esquemas de bases de datos a través de migraciones.[5]

Microsoft Entity Framework: Se refiere a un *sistema de Mapeo Objeto-Relacional* (*ORM*)[5] con capacidades de mapeo flexibles. Su entorno gráfico permite al desarrollador crear un modelo de bases de datos y ligarlo de forma manual a una localidad de almacenamiento o utilizar el esquema de mapeo predefinido para generar una nueva base de datos. Lo que da como resultado un modelo que persiste correctamente, en donde además se crean los scripts para las llamadas migraciones, que son necesarios para la evolución y actualización de la base de datos. [5]

1.1.2. Productos y Servicios Disponibles en el Mercado

En la actualidad ya existen algunas herramientas que son capaces de comparar esquemas y facilitar la creación y de scripts que permitan versionar las bases de datos.

SQL Compare. Herramienta comercial para comprar y desplegar cambios en las bases de datos desarrollada por Redgate, permite generar scripts de sincronización y posee la posibilidad de ejecutarlo automáticamente en la base de datos que se desea adaptar. [6]

SQL Source Control. Es parte del paquete de desarrollo de SQL desarrollado por Red-Gate, en donde te permite conectar SQL Server Management Studio a un controlador de versiones que puede estar alojado en algunos de los repositorios para desarrollo de software más comunes, tales como, SVN, Git, Mercurial, Vault, entre otros. [7]

1.2. Planteamiento del problema

En la empresa *SDPoint* el proceso de actualización de un sistema de información en la máquina de cualquiera de sus clientes conlleva a la actualización del esquema de base de

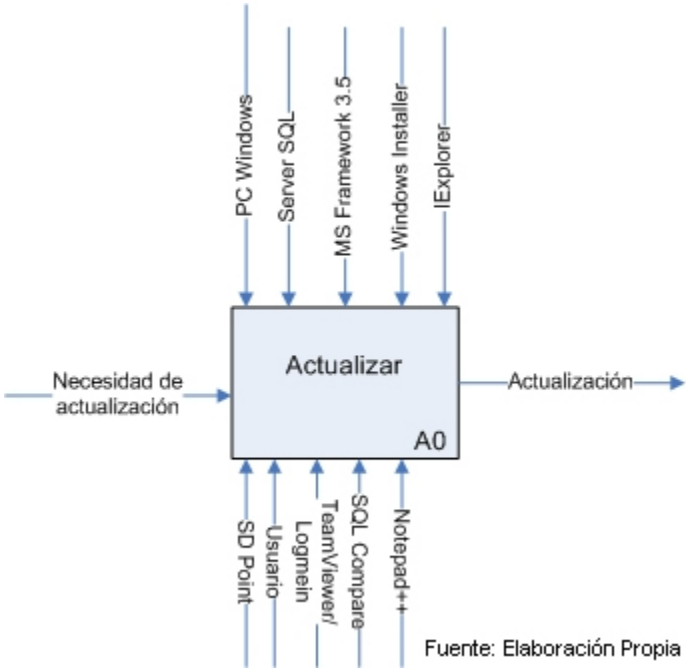


Figura 1.1: Flujo de actualización IDEF Nivel 0

datos, proceso que no se encuentra controlado, estandarizado ni automatizado, por lo cual no es una tarea que el usuario final pueda realizar de forma transparente en conjunto a la actualización del sistema. Por esta razón, es necesario que alguno de los integrantes de dicha empresa tenga que involucrarse de forma directa en la actualización del sistema. Lo que trae consigo un consumo considerable en el tiempo de desarrollo del día en producción.

Se muestra una serie de diagramas bajo el esquema IDEF con la finalidad de expresar de forma más gráfica el problema propuesto, y donde se describe desde nivel abstracto hasta el específico, el proceso que involucra la actualización de un producto de software de la empresa *SD Point*, que usted puede visualizar en la Figura 1.1.

En la Figura 1.1 se puede observar los agentes que intervienen en la actualización de un sistema, así como las herramientas necesarias para su conclusión. De una forma más específica, como se presenta a continuación en la figura 1.2 se puede observar que para realizar una actualización integral a un sistema de la empresa *SDPoint*, deben realizarse 3 instalaciones

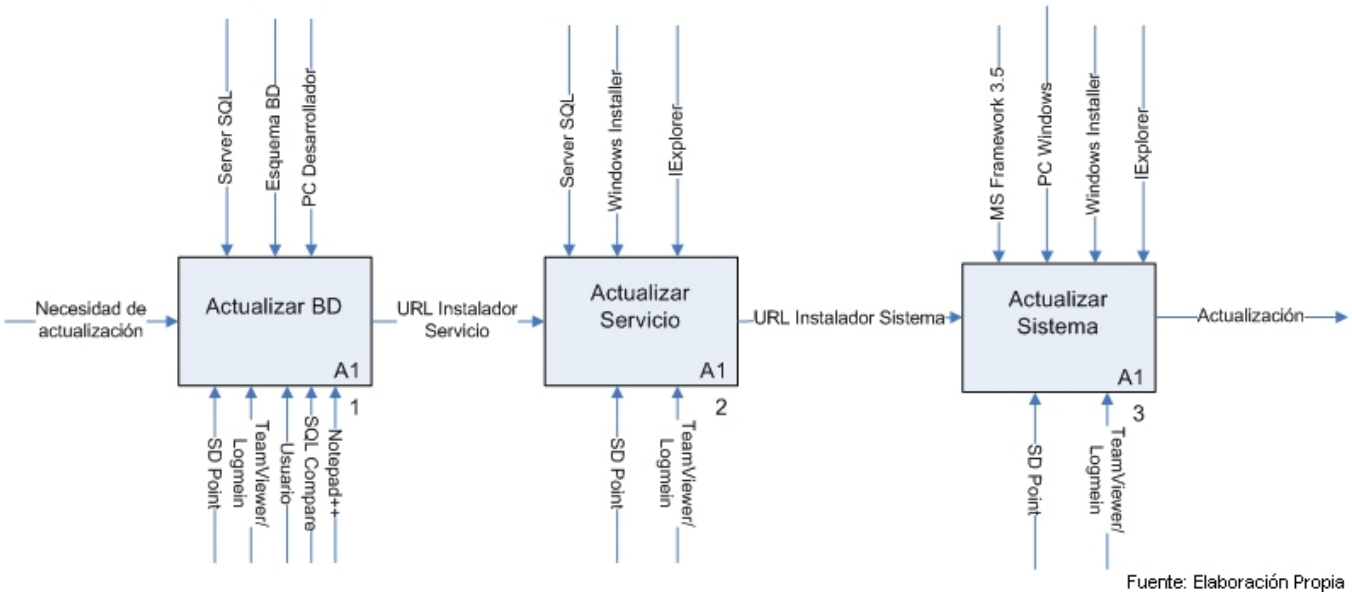


Figura 1.2: Flujo de actualización de Aplicaciones, Servicio y Base de datos IDEF Nivel 1

de forma independiente y las tres quedar en versiones compatibles para un funcionamiento correcto.

Para este caso práctico el enfoque de este trabajo fué únicamente en relación a la actualización de la base de datos, proceso que requiere de mayor atención debido al tiempo que éste requiere para su ejecución. En la Figura 1.3 se detalla cada uno de los pasos, los involucrados y las herramientas necesarias para esta tarea.

1.3. Objetivos

1.3.1. Objetivo general

Identificar e implementar una metodología dentro de la gestión del ciclo de vida de un proyecto de software, que contemple la actualización ordenada y sincronizada de ambos componentes del sistema (la aplicación y la base de datos), de manera tal que al aplicar cualquier

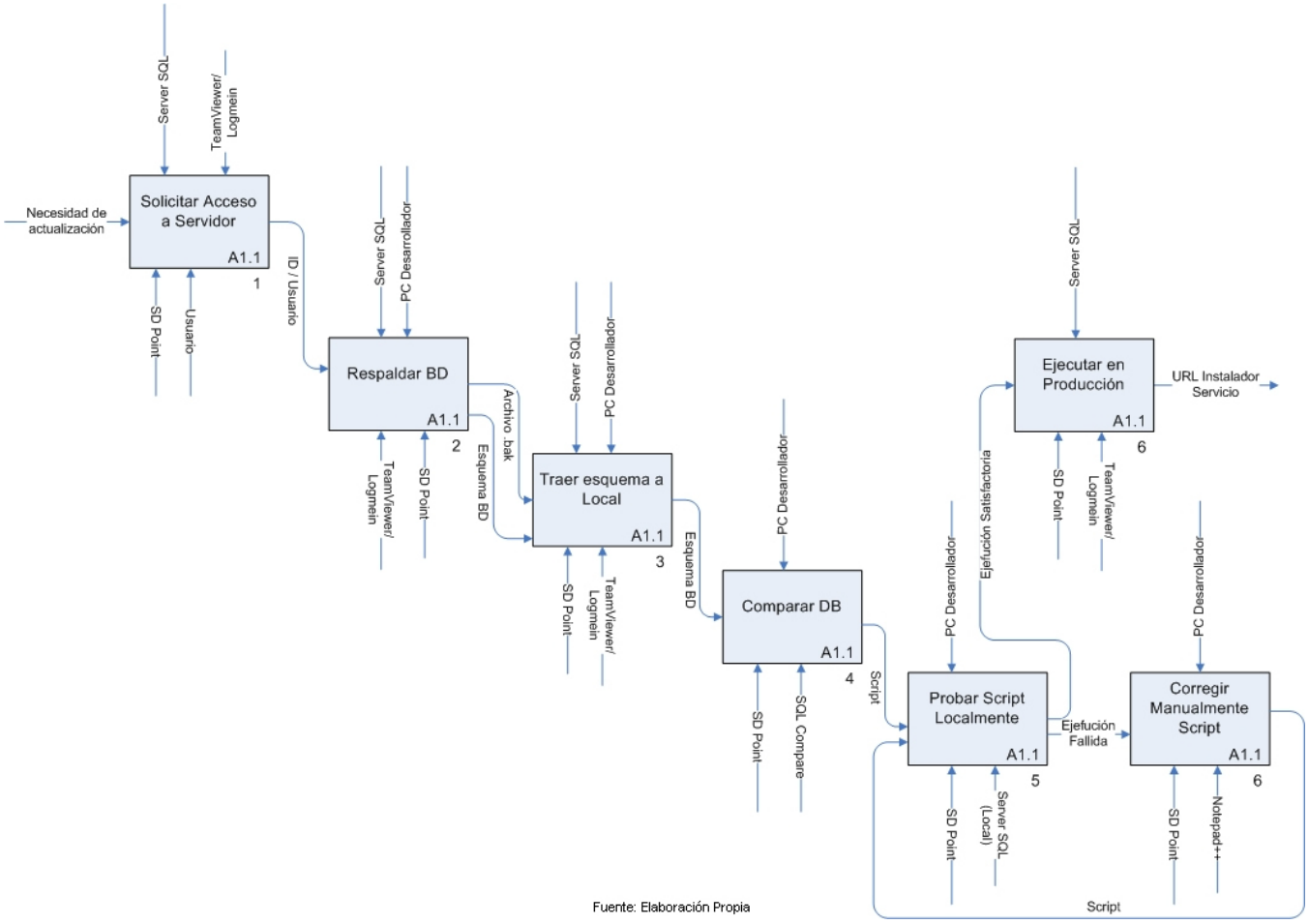


Figura 1.3: Flujo de actualización de Aplicaciones, Servicio y Base de datos IDEF Nivel 1.1

actualización al mismo, tanto la base de datos como el software de aplicación queden en versiones compatibles.

1.3.2. Objetivos específicos

- Definir una metodología para versionar el esquema de base de datos.
- Automatizar el proceso de versionamiento.
- Aplicar el proceso en la actualización en una de las bases de datos de la empresa *SDPoint*.

1.4. Justificación

Uno de los principales problemas en el desarrollo del software que se presenta de forma casi inevitable, proviene de la naturaleza evolutiva de los mismos, que traen consigo una serie de cambios, no solo de la aplicación misma, sino también de la estructura física de sus bases de datos y la migración de la información existente.

Actualmente, la mayoría de los sistemas de bases de datos facilitan el diseño, modelado y mantenimiento de bases de datos relacionales, sin embargo se encuentran limitados para dar soporte y gestionar la evolución del esquema lógico de la base de datos, tarea que asumen los programadores y que en muchos casos se realiza de una forma informal y parcialmente manual.

Por esta razón se debe controlar el proceso evolutivo del esquema de base de datos, con la finalidad de establecer una metodología que facilite la actualización de versiones entre bases de datos minimizando no solo el margen de error que se produciría en un proceso manual,

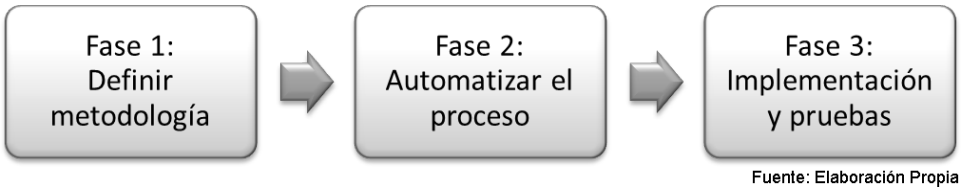


Figura 1.4: Alcances del trabajo

sino también el tiempo de actualización que viene ligado a la inactividad en el ambiente de producción.

El crecimiento de una empresa de software está asociado a la eficiencia y capacidad de dar pronta respuesta a las necesidades de sus clientes, si bien muchas tareas es posible realizarlas de modo manual en grupos reducidos de instalaciones, y alcanzar una cantidad suficiente de sistemas desplegados, se llega al límite del rendimiento marginal, en donde la empresa debe optimizar sus procesos mediante la automatización de los mismos, la generación de metodologías de trabajo y herramientas que le den soporte y faciliten las tareas más comunes, críticas y repetitivas con la finalidad de mantenerse vigente y competitiva.

1.5. Alcances del Trabajo

El alcance de este trabajo se delimitó a la selección de una metodología de versionamiento de bases de datos que pueda adaptarse a las necesidades de la empresa objeto de estudio y que permita llevar al control de las versiones, la automatización del proceso de versionamiento de bases de datos y por último a la implementación sobre una base de datos en un ambiente real y la realización de pruebas sobre la misma. Esto está ilustrado en la Figura 1.4.

Fase 1. Definir una metodología para versionar el esquema de base de datos: Se aplican técnicas de diseño en las cuales se define la solución con suficiente detalle para permitir su interpretación y realización física. Diseño de datos, arquitectónico, de interfaces,

y procedimientos y métodos

Fase 2. Automatizar el proceso de versionamiento: Diseñar el proceso de despliegue del sistema, para utilizar la metodología definida en la fase anterior, de forma tal que su utilización sea automática durante el proceso de instalación.

Fase 3. Aplicar el proceso en la actualización en una base de datos de los sistemas de la empresa: En esta fase se asegura que la solución funcione de acuerdo a los requerimientos y que los usuarios puedan operarlo de una forma transparente.

1.6. Metas

- Contar con un estudio de las tecnologías existentes para el versionamiento de bases de datos.
- Implementar un proceso de software que automatice el versionamiento de bases de datos.
- Tener al menos un caso de estudio donde se haya aplicado el proceso.

1.7. Como está organizado este documento

El presenta trabajo consta de cinco capítulos. En el primer capítulo se encuentra una descripción del caso de estudio, un breve resumen de antecedentes, los objetivos de este trabajo, delimitación de los alcances del mismo. Durante el segundo capítulo, se describe la metodología y la forma de trabajo que fue seleccionada para el desarrollo del proyecto. El tercer capítulo, hace una revisión del estado de la técnica y los fundamentos teóricos que

sirven de base para las técnicas, lenguaje de programación y procedimientos propuestos y el desarrollo de la solución. En el cuarto capítulo, se explica la metodología de versionamiento basada en *SMO* y el desarrollo de los operadores para bases de datos en Microsoft SQL 2008R2, donde se describen cada uno de los procedimientos almacenados que lo componen. En el quinto y penúltimo capítulo, se describen las pruebas realizadas, explicando la función que esta cumple en la metodología propuesta y los resultados obtenidos. Por último, en el sexto capítulo presentamos las conclusiones así como nuevas oportunidades de continuidad del trabajo.

Capítulo 2

Marco teórico

2.1. Bases de Datos

Una base de datos es un lugar para almacenar datos, diseñada para soportar de forma eficiente no solo su almacenamiento, si no también, la consulta y el mantenimiento [8]. Existen diferentes tipos de bases de datos que permiten adecuarse a las necesidades de la industria.

Una base de datos puede especializarse en almacenar archivos binarios, documentos, imagánes, videos, datos relacionados, datos multidimensionales, datos transaccionales, análisis de datos o datos geográficos, por nombrar algunos ejemplos.

A los datos almacenados de forma tabular se le llama base de datos relacional. Cuando son organizados en una estructura de árbol entonces es llamada bases de datos jerárquica y por último cuando son almacenados en forma de grafos representando relaciones entre objetos, se hace referencia a una base de datos neuronal [8].

Expresándonos de forma técnica, una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas [9].

2.1.1. Manejador de Base de datos

También conocido como DBMS, un sistema de manejo de base de datos es un conjunto de herramientas de software que permiten el control de acceso, la organización, almacenamiento, administración, consulta y mantenimiento de los datos en una base de datos [8].

2.2. Bases de datos relacionales

Una base de datos relacional es un repositorio compartido de datos. Para permitir que los datos de una base de datos relacional se encuentren disponibles para los usuarios, tenemos que tomar en cuenta diferentes problemas. Uno de ellos es como los usuarios realizaran peticiones a la base de datos:

- ¿Cuál de los lenguajes se van a utilizar?
- Integridad y seguridad de los datos

El diseño de esquemas de bases de datos relacionales es el primer paso para crear una aplicación de bases de datos [10].

2.2.1. ACID

Para asegurarnos de la integridad de la información, se necesita que el sistema de bases de datos mantenga las siguientes propiedades de transacción:

Atomicidad Esto significa que las transacciones son “todo o nada”, se realizan todas ellas o ninguna de ellas.

Consistencia Se asegura de que la información es válida tanto antes como después de la transacción. La integridad de la información debe ser mantenida (por ejemplo, referencias a llaves foráneas) y la estructura interna de la información debe estar en un estado válido.

Insolación Requisito en donde las transacciones no pueden ser dependientes de otras operaciones que se puedan estar realizando simultáneamente (ya sea al mismo tiempo o que se superponen). Una transacción no puede ver los datos de otra transacción que está en un estado intermedio, pero en su lugar ve los datos, ya que era o bien antes de la transacción o después comenzó.

Durabilidad Significa que los efectos de las transacciones son permanentes después de que la transacción ha sido realizada, y cualquier cambio podría traer errores al sistema[11].

ACID es un acrónimo derivado de la primera letra de estas propiedades[10].

2.2.2. Lenguaje estructurado de consulta de datos (SQL)

Los lenguajes de consulta (*Structured Query Language*: SQL) de datos fueron diseñados para datos persistentes que residen en los discos, no para datos transitorios. Es decir, para datos almacenados de forma estructurada [12].

Por definición SQL es un lenguaje declarativo de álto niveĺde acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ellas. [13]

2.2.3. Lenguaje de Manipulación de Datos (DML)

Es un lenguaje de programación que tiene una capacidad poderosa de cálculo, flujo de control, entrada salida, también tiene constructores sintácticos para acceso a base de datos (actualización, recuperación e intercambio dinámico de datos entre el programa y la base de datos). El DML es utilizado por el programador de la aplicación. Un lenguaje de manipulación de datos puede ser:

- Un DML stand-alone. En este caso el SMBD provee de un compilador o interprete para el DML. La desventaja de este lenguaje es que no puede ser usado para programas complejos, los cuales ejecutan algún acceso a la base de datos, pero simultáneamente ejecutan otras tareas, por ejemplo, cálculos numéricos.

- Una interface para llamadas al sistema. El usuario escribe un programa en un lenguaje de programación tradicional. El usuario ejecuta accesos a la base de datos por llamadas a subrutinas al SMBD. Las llamadas al sistema son interpretadas en tiempo de ejecución del programa. Una desventaja es que si la llamada al sistema contiene una solicitud incorrecta, el usuario no puede ser notificado en tiempo de compilación, sino que tiene que esperar hasta que el programa aborte.

- Un DML Incrustado en un Lenguaje de Programación Anfitrión. Este es una extensión de acceso a base de datos de un lenguaje de programación de propósito general. El SMBD pre compila el programa en el lenguaje anfitrión sin las sentencias del DML. Durante la pre compilación el SMBD válida la sintaxis y la compatibilidad con el esquema de la base de datos. El SMBD puede también ejecutar optimización del algoritmo del usuario[14].

El programa resultante es traducido por el compilador del lenguaje anfitrión. Cuando el programa se ejecuta, este puede comunicarse con el SMBD, pero las llamadas al sistema de

esta comunicación son transparentes al usuario.

Un ejemplo de sentencia DML en T-SQL es: SELECT para realizar consultas, e INSERT, UPDATE y DELETE para hacer mantenimiento de los datos. [9]

2.2.4. Sentencias DML en SQL Server

A continuación se definen algunas de las sentencias más comunes, o más utilizadas en SQL Server:

FROM.

Especifica la tabla, vista, tablas derivadas y tablas cruzadas que desean ser llamadas en las sentencias *DELETE*, *SELECT* y *UPDATE.* En una sentencia SELECT la cláusula FROM is estrictamente requerida exceptuando cuando la selección solo contiene constantes, variables y expresiones aritmeticas, es decir; no incluye nombres de columnas.

INSERT.

Agrega uno o más registros a una tabla o vista en SQL Server.

Script 2.1: Sintáxis básica de la sentencia DML: Insert

```
INSERT INTO dbf_name [(FieldName1 [, FieldName2, ...])]
   VALUES (eExpression1 [, eExpression2, ...])
```

SELECT

La sentencia *SELECT* es utilizada para obtener datos de una o mas tablas o vistas. Cuando los datos son obtenidos a través de mas de una tabla, se utiliza una sentencia *JOIN*.

Este comando puede ser utilizado especificando las columnas que se desean obtener, seguida de la sentencia *FROM* y las tablas de las cuales se realiza la consulta. (Véase lst:SintaxisSELECT)

Script 2.2: Sintáxis básica de la sentencia DML: SELECT

```
SELECT column_name, column_name
FROM table_name
```

Es posible indicarle al motor de base de datos que se requieren todas las columnas sin necesidad de escribirlas explícitamente, haciendo uso del caracter *, tal como se muestra en el Script 2.3.

Script 2.3: SELECT *

```
SELECT * FROM TableName
```

UPDATE

El comando *UPDATE* se utiliza para realizar cambios en los valores de los datos que se encuentran almacenados en una base de datos.

Script 2.4: Sintáxis básica de la sentencia *UPDATE*

```
UPDATE tablename
```

```
SET columnname =expression
[WHERE condition];
```

[15]

2.2.5. Lenguaje de Definición de Datos (DDL)

El lenguaje de definición de datos es un vocabulario utilizado para definir estructuras de datos en SQL, tales como sentencias de creación (*CREATE*), modificación (*ALTER*) o eliminación (*DROP*) de estas estructuras en una instancia de SQL Server. [16]

Sentencia ALTER

SQL Server Transact-SQL incluye algunas sentencias *ALTER* que son utilizadas para realizar alguna modificación a un objeto de base de datos ya existente. Por ejemplo, usted puede utilizar esta sentencia para agregar, quitar o modificar columnas dentro de una tabla. [16]. La sintaxis para realizar esta tarea es ilustrada en el Script 2.5.

Script 2.5: Sintaxis para agregar una columna

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

Sentencia DROP

Esta instrucción en SQL Server Transact-SQL permite eliminar objetos existentes en una base de datos. Por ejemplo, usted puede utilizar esta sentencia para eliminar una tabla, o una base de datos.

Script 2.6: Sintaxis para eliminar una tabla

```
DROP TABLE table_name
```

Es importante remarcar, que una tabla no puede ser eliminada mientras tenga referencia con otra tabla que aun exista en la base de datos. Para eliminar una tabla que tiene una referencia, usted puede primero eliminar la tabla hija y después la padre, o romer las referencias entre las tablas. Según sea necesario.

Script 2.7: Sintaxis para eliminar una base de datos

```
DROP DATABASE database_name
```

En el caso de la eliminación de bases de datos (Script 2.7), esta no puede ser eliminada mientras se encuentre en uso por cualquier usuario. Una de las alternativas para eliminar una base de datos y protejerla de su uso, es utilizando una sentencia DDL de *ALTER* sobre la base de datos, para definirla como *SINGLE_USER* (Solo un usuario a la vez puede estar utilizandola). Véase Script 2.8.

Script 2.8: Sintaxis para establecer una base de datos como Single User

```
ALTER DATABASE AdventureWorks2012
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE;
```

2.3. Modelos de Datos

Detrás de la estructura de la base de datos se encuentra el modelo de datos que comprende una colección de herramientas conceptuales para describir los datos, las relaciones entre los datos, la semántica de los datos y las restricciones entre de datos. El diseño general de la

base de datos se conoce como esquema de la base de datos. Un esquema de base de datos es especificado por un conjunto de definiciones que son expresadas usando un lenguaje de definición de datos (DDL).

El concepto del esquema de base de datos puede ser entendido mediante una analogía con un programa escrito en algún lenguaje de programación. El esquema de la base de datos correspondería a la declaración de variables junto con su definición de tipo en un programa. Los valores de estas variables en un programa en un momento dado corresponden a lo que se conoce como instancia de un esquema de base de datos.

2.4. Esquemas de Bases de Datos

Los sistemas de bases de datos tienen varios esquemas particionados de acuerdo a los niveles de abstracción.

Una base de datos puede también tener varios esquemas desde un nivel de vista, a menudo se llaman "sub esquemas", que describen diferentes vistas de las base de datos.

De todos estos el esquema lógico es por mucho el más importante en cuanto a su efecto en los programas de aplicación, dado que los programadores construyen aplicaciones por medio del uso del esquema lógico. El esquema físico está oculto debajo del esquema lógico, y a menudo puede ser cambiado con facilidad sin afectar los programas de aplicación. Se dice que los programas de aplicación presentan "independencia física de datos", si no dependen del esquema físico y por lo tanto no necesitan ser reescritos si el esquema físico cambia.

2.4.1. Esquemas en SQL Server 2008

Existe una diferencia entre lo que llamamos 'Esquema de bases de datos' para referirnos a la estructura física de una base de datos, y los esquemas de bases de datos en *Microsoft SQL Server*.

Un esquema es un contenedor logico de la base de datos, en SQL se define un esquema como un nombre para un grupo de objetos [17]. Crear objetos puede ser util cuando se desean agrupar objetos, tales como Tablas, vistas, procedimientos, para definirlos como parte de un rol o para hacer referencia a segmento del proceso que describe la base de datos.

SQL Server 2008 soporta esquemas de bases de datos como una forma física de agrupar objetos de la base de datos. Estos esquemas son utilziados para agrupar tablas, procedimientos almacenados, vistas y funciones definidas por el usuario con la finalidad de proveer una herramienta de administración y seguridad.

Cuando el usuario crea un nuevo objeto de bases de datos, tal como tablas, y este no especifica un esquema, este objeto es automáticamente creado bajo el esquema 'default', el cual normalmente es 'dbo', sin embargo los administradores de bases de datos (DBAs) pueden asignar diferentes esquemas por defecto a diferentes usuarios. Por esta razón se recomienda siempre especificar especificar de forma explícita el nombre del esquema cuando se creen nuevos objetos de bases de datos.[18]

2.5. Evolución de esquemas de bases de datos

Evolución del esquema es la capacidad de cambiar esquemas ya desplegados, es decir, las estructuras de metadatos que describen formalmente las bases de datos.

De forma inevitable, la necesidad de evolución del esquema de la base de datos se produce

muy a menudo con el fin de hacer frente a requerimientos nuevos o cambiantes [19], para corregir las deficiencias de los esquemas actuales o migrar hacia una nueva plataforma.

El apoyo efectivo de la evolución del esquema es un reto ya que los cambios de esquema pueden tener que ser propagados, correcta y eficientemente, a datos de instancia, vistas, aplicaciones y sus componentes de sistema dependientes. Lo ideal es que hacer frente a estos cambios deberían exigir poco trabajo manual y falta de disponibilidad del sistema. Por ejemplo, cambios en un esquema de base de datos"S" debe ser propagado a los datos de instancias y vistas definidas en "S" con una mínima intervención humana.

La evolución de esquemas ha sido un área de investigación activa durante un largo tiempo. Sin embargo, la necesidad de fuertes evoluciones de esquemas ha incrementado y muchos artículos han aparecido recientemente. Por otra parte, los sistemas de bases de datos comerciales, como *IBM DB2*, *Oracle* y *Microsoft SQL Server* han comenzado a apoyar las capacidades de evolución del esquema en línea.

2.6. Versionamiento de bases de datos

El concepto de versionamiento comprende los procesos de registrar y manejar los cambios en una base de datos multiusuarios al crear una 'versión' de la base de datos, una alternativa independiente, persistente vista de la base de datos no requiere la creación de una copa de la información y soporta múltiples editores.

2.6.1. Versión

Cada versión representa una única vista de la base de datos, que puede ser distinguida de otra versión por un conjunto de modificaciones realizadas después de que la version ha sido creada. Las versiones difieren únicamente por el numero de registros que representa cada

objeto de base de datos (tablas), cualquier objeto nuevo, modificado o eliminado de la base de datos. [20]

Por ejemplo, en una versión una tabla de productos en un sistema de ventas puede contener una serie de atributos, tales como *Descripcion*, *PrecioVenta*, *PrecioCompra*, *TasaIVA*. Y un sistema de software puede estar por mucho tiempo operando con este tipo de tablas. Sin embargo, con el paso del tiempo nuevas especificaciones pueden surgir por la evolución de leyes financieras que ahora podrían requerir la inclusión de otras tablas. (Véase Figura 2.1)

Fuente: Elaboración Propia

Figura 2.1: Tabla producto - Versión 1(Fuente: Elaboración propia)

Como por ejemplo, utilizando el mismo escenario del sistema de ventas y la tabla de productos, esta podría en un futuro contener *Descripción*, *PrecioVenta*, *PrecioCompra*, *TasaIVA* y *TasaIEPS*, generando de esta forma una nueva versión de la base de datos tal como se muestra en la Figura 2.2.

Fuente: Elaboración Propia

Figura 2.2: Tabla producto - Versión 2(Fuente: Elaboración propia)

2.6.2. Modificación de Esquemas

Se dice que existe una modificación a un esquema de base de datos cuando el sistema de base de datos permite cambios en la definición del esquema de una base de datos poblada con información.[4]

Evolución de esquemas de bases de datos

Se dice que un esquema de datos evoluciona cuando un sistema de bases de datos facilita la modificación del esquema de base de datos sin la pérdida de los datos existentes.[4]

2.6.3. Versionamiento de esquemas de bases de datos

Se dice que existe una version de base de datos cuando un sistema de base de datos permite el acceso a todos los datos, tanto retrospectiva como prosepectivamente, a través de la versión que puede ser definida a través de interfaces.[4]

Versionamiento parcial de esquemas de bases de datos

El versionamiento parcial de esquemas de bases de datos se da cuando un sistema de bases de datos permite la visualización de todos los datos, tanto restrospectiva como prospectivamente atravéś de interfaces definidas por el usuario. La actualización de datos es permitida a través de a través de la referencia del diseño de un esquema únicamente definido.[4]

Versionamivento completo de esquemas de bases de datos

El versionamiento completo de bases de datos se da cuando un sistema de bases de datos permite visualizar y actualizar todas los datos, tanto retrospectiva como prospectivamente a

través de interfaces definidas por el usuario. [4]

2.7. Operadores Modificadores de Esquemas (SMO)

Operadores de Modificación de Esquemas inicialmente propuestos por Shneiderman y Thomas en 1982 como un set de cambios a un esquema de base de datos de forma comprensivo, incluyendo cambios de estructura y cambios relacionados con llaves y dependencias de una base de datos.

La sintaxis del SMO Proporciona una manera concisa para describir los cambios típicos de un esquema de base de datos y la migración de datos correspondiente. Cada SMO toma como entrada un esquema y produce como salida una nueva versión del mismo esquema. Es posible también combinar varios SMO simples en secuencia, para describir intercambio estructural complejo.

La Figura 2.3 presenta la semántica utilizada por los operadores SMO, expresando la entrada y salida de estos operadores. Cada SMO toma como entrada un esquema produciendo como salida una nueva versión del mismo esquema. [3]

SMO Syntax	Input rel.	Output rel.	Forward DEDs	Backward DEDs
CREATE TABLE R($\bar{\mathbf{A}}$)	-	R($\bar{\mathbf{A}}$)	-	-
DROP TABLE R	R($\bar{\mathbf{A}}$)	-	-	-
RENAME TABLE R INTO T	R($\bar{\mathbf{A}}$)	T($\bar{\mathbf{A}}$)	R($\bar{\mathbf{x}}$) → T($\bar{\mathbf{x}}$)	T($\bar{\mathbf{x}}$) → R($\bar{\mathbf{x}}$)
COPY TABLE R INTO T	R_{V_i}($\bar{\mathbf{A}}$)	$R_{V_{i+1}}$($\bar{\mathbf{A}}$), T($\bar{\mathbf{A}}$)	R_{V_i}($\bar{\mathbf{x}}$) → $R_{V_{i+1}}$($\bar{\mathbf{x}}$) R_{V_i}($\bar{\mathbf{x}}$) → T($\bar{\mathbf{x}}$)	$R_{V_{i+1}}$($\bar{\mathbf{x}}$) → R_{V_i}($\bar{\mathbf{x}}$) T($\bar{\mathbf{x}}$) → R_{V_i}($\bar{\mathbf{x}}$)
MERGE TABLE R, S INTO T	R($\bar{\mathbf{A}}$), S($\bar{\mathbf{A}}$)	T($\bar{\mathbf{A}}$)	R($\bar{\mathbf{x}}$) → T($\bar{\mathbf{x}}$); S($\bar{\mathbf{x}}$) → T($\bar{\mathbf{x}}$)	T($\bar{\mathbf{x}}$) → R($\bar{\mathbf{x}}$) ∨ S($\bar{\mathbf{x}}$)
PARTITION TABLE R INTO S WITH *cond*, T	R($\bar{\mathbf{A}}$)	S($\bar{\mathbf{A}}$), T($\bar{\mathbf{A}}$)	R($\bar{\mathbf{x}}$), *cond* → S($\bar{\mathbf{x}}$) R($\bar{\mathbf{x}}$), ¬*cond* → T($\bar{\mathbf{x}}$)	S($\bar{\mathbf{x}}$) → R($\bar{\mathbf{x}}$),*cond* T($\bar{\mathbf{x}}$) → R($\bar{\mathbf{x}}$),¬*cond*
DECOMPOSE TABLE R INTO S($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$), T($\bar{\mathbf{A}}$,$\bar{\mathbf{C}}$)	R($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$,$\bar{\mathbf{C}}$)	S($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$), T($\bar{\mathbf{A}}$,$\bar{\mathbf{C}}$)	R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$) → S($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$) R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$) → T($\bar{\mathbf{x}}$,$\bar{\mathbf{z}}$)	S($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$) → ∃$\bar{\mathbf{z}}$ R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$) T($\bar{\mathbf{x}}$,$\bar{\mathbf{z}}$) → ∃$\bar{\mathbf{y}}$ R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$)
JOIN TABLE R, S INTO T WHERE *cond*	R($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$), S($\bar{\mathbf{A}}$,$\bar{\mathbf{C}}$)	T($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$,$\bar{\mathbf{C}}$)	R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$), S($\bar{\mathbf{x}}$,$\bar{\mathbf{z}}$), *cond* → T($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$)	T($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$,$\bar{\mathbf{z}}$) → R($\bar{\mathbf{x}}$,$\bar{\mathbf{y}}$),S($\bar{\mathbf{x}}$,$\bar{\mathbf{z}}$),*cond*
ADD COLUMN C [AS *const*|$func$($\bar{\mathbf{A}}$)] INTO R	R($\bar{\mathbf{A}}$)	R($\bar{\mathbf{A}}$,C)	R($\bar{\mathbf{x}}$) → R($\bar{\mathbf{x}}$, *const*|$func$($\bar{\mathbf{x}}$))	R($\bar{\mathbf{x}}$,C) → R($\bar{\mathbf{x}}$)
DROP COLUMN C FROM R	R($\bar{\mathbf{A}}$,C)	R($\bar{\mathbf{A}}$)	R($\bar{\mathbf{x}}$,z) → R($\bar{\mathbf{x}}$)	R($\bar{\mathbf{x}}$) → ∃z R($\bar{\mathbf{x}}$,z)
RENAME COLUMN B IN R TO C	R_{V_i}($\bar{\mathbf{A}}$,B)	$R_{V_{i+1}}$($\bar{\mathbf{A}}$,C)	R_{V_i}($\bar{\mathbf{x}}$,y) → $R_{V_{i+1}}$($\bar{\mathbf{x}}$,y)	$R_{V_{i+1}}$($\bar{\mathbf{x}}$,y) → R_{V_i}($\bar{\mathbf{x}}$,y)
NOP	-	-	-	-

Autor: Carlo A. Curino et al. 2008

Figura 2.3: Sintáxis de SMO

2.8. Despliegue (*Deployment*)

Los problemas del despliegue de software se origina desde el proceso de la ingeniería del mismo[21], y esto se debe a que los sistemas de software dejaron de ser aplicaciones '*sand-alone*'. De forma creciente, las aplicaciones desarrolladas son la integración de componentes tanto ejecutables como de datos, posiblemente dispersos en diferentes puntos de una red, o incluso a través de Internet. [22]

Entre el punto en donde una aplicación está disponible y la aplicación puede ser utilizada, hay varios pasos que deben realizarse:

Ubicación del Software. Primeramente, el usuario debe encontrar la aplicación. Por ejemplo, cuando el usuario descarga un archivo PDF, el o ella primeramente debieron buscar algún software para poder leerlo. Incluso después de haber sido instalado, este lector de archivos PDF se encuentra trabajando sobre el sistema operativo cada vez que el documento que fue descargado es abierto.

Recepción del Software. El usuario debe obtener una copia de la aplicación. Algunos de estos pueden ser descargados a través de Internet, comprados en alguna tienda o solicitados a través del sistema postal, algunos otros se refieren a software a la medida que es ejecutada a través de una red en la organización.

Instalación del Software. El programa debe ser cargado en la computadora del usuario.

Actualización del Software. Cuando existe una nueva versión de la aplicación, algunos usuarios preferirán utilizar una nueva versión a alguna más vieja. Sin embargo los pasos anteriormente mencionados deben volver a repetirse y algunas actividades necesitan ser realizadas.[21]

Los autores llaman a este el proceso de desplegado. (Tabla 2.1)

Tabla 2.1: Actividades del Software y sus participantes

Actividad	Participantes Comunes
1. Análisis y especificación de software	Administrador del Proyecto, Analista de sistemas
2. Diseño del Software	Analista de sistemas, programador
3. Implementación, pruebas	Programador, Administrador de calidad
4. Documentación	Analista de sistemas, programador, escritor técnico
5. Entrega del software	Administrador del proyecto, vendedor del software, usuario
6. Instalación, Actualización	Usuario, Soporte técnico
7. Entrenamiento, Operación	Usuario, Soporte técnico
(Fuente: Jai 2003)	

El esfuerzo invertido en el desplegado del software es proporcional a:

1. El número de aplicaciones a ser desplegadas.

2. El número de usuarios utilizando el software.

3. El numero de releases del software.

4. La cantidad de veces que el software ha sido desplegado.[21]

2.9. Catálogos de Sistema de bases de datos

En el Estándar SQL, un *CATALOG* es una colección de esquemas que contiene entre otras cosas, INFORMATION_SCHEMA. Está compuesto por tablas y vistas que proporcionan toda la información acerca de los objetos y registros definidos en la base de datos: esquemas, tablas, privilegios, entre otras cosas. El ultimo estándar también incluye la estructura e integridad de las referencias entre tablas, asi como la información de seguridad ya autorización para la base de datos.

La idea principal es proporcionar tanto a usuarios como a los administradores de la base

de datos información de la base de datos de una forma consistente y estandarizada de acceso a los metadatos.

Por definición, las tablas y vistas de INFORMATION SCHEMA no pueden ser actualizadas directamente, aunque algunos manejadores de bases de datos como IBM o DB2 lo permiten. [17].

2.9.1. Catálogos de sistema en SQL Server 2008 R2

Microsoft SQL Server 2008 proporciona diferentes formas de obtener información a través de la vista INFORMATION_SCHEMA o de los procedimientos almacenados y funciones del sistema. [17] Algunos autores enfatizan que sacar información de la base de datos a través de la vista de INFORMATION_SCHEMA es más recomendada que realizarlo a través de los procedimientos y funciones, sin embargo se deja a criterio del usuario de las bases de datos.

Las vistas de información del esquema proporcionan un vistazo interno a los metadatos de SQL Server, permitiendo a algunas aplicaciones funcionar correctamente a pesar de cambios significativos que pudieron haberse realizando en las tablas del sistema. La información del esquema incluida en SQL Server cumple con el standard de defunción ISO para INFORMATION_SCHEMA.

Cada vista de información del esquema contiene toda la metadata de todos los objetos almacenados en una base de datos particular, es decir, la descripción de cada uno de ellos.

Una de las formas más sencillas de obtener información de la base de datos en SQL Server 2008 es realizarlo directamente enviando consultas (querys) sobre las tablas y vistas de sistema, aunque no es una de las recomendaciones que Microsoft hace a cerca de ello, puesto que existe la posibilidad que con la liberación de nuevas versiones de SQL Server, estas tablas cambien su estructura.

2.9.2. Objetos de sistema (sys.objects)

Esta tabla contiene una fila por cada objeto que se encuentra dentro del esquema definido por el usuario en una base de datos, a fin de describirla. Las Tablas 2.2, 2.3 y 2.4 se describen algunas de las columnas que contiene esta tabla y a que se refiere cada una de ellas. [23]

2.10. Metodologías de Desarrollo de Software

Existen numerosas propuestas metodológicas que inciden a distintas dimensiones del proceso de desarrollo de software. Por una parte, tenemos aquellas propuestas de forma más tradicional centradas especialmente en el control del proceso estableciendo rigurosamente actividades, artefactos, herramientas y las notaciones que se utilizarán. Este tipo de metodologías han demostrado su efectividad en un gran número de proyectos, pero también han presentado problemas en algunos otros. Una mejora posible es incluir los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo aun más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto de software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está demostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software.[24]

Tabla 2.2: sys.objects

Columna	Tipo de datos	Descripción
name	**sysname**	Nombre del objeto.
object_id	**int**	Número de identificación del objeto. Es único en una base de datos.
principal_id	**int**	Identificador del propietario individual, si es diferente del propietario del esquema. De forma predeterminada, los objetos contenidos en el esquema pertenecen al propietario del esquema. No obstante, es posible especificar un propietario alternativo mediante la instrucción ALTER AUTHORIZATION para cambiar la propiedad. Es NULL si no existe ningún propietario individual alternativo. Es NULL si el tipo de objeto es uno de los siguientes: C = Restricción CHECK D = DEFAULT (restricción o independiente) F = Restricción FOREIGN KEY PK = Restricción PRIMARY KEY R = Regla (estilo antiguo, independiente) TA = Desencadenador de ensamblado (integración CLR) TR = Desencadenador SQL UQ = Restricción UNIQUE
schema_id	**int**	Identificador del esquema que contiene el objeto. Los objetos de sistema con ámbito de esquema siempre están contenidos en los esquemas sys o INFORMATION_SCHEMA.
parent_object_id	**int**	Identificador del objeto al que pertenece este objeto. 0 = No es un objeto secundario.

Tabla 2.3: sys.objects

Columna	Tipo de datos	Descripción
type	**char(2)**	Tipo de objeto: AF = Función de agregado (CLR) C = Restricción CHECK D = DEFAULT (restricción o independiente) F = Restricción FOREIGN KEY FN = Función escalar de SQL FS = Función escalar del ensamblado (CLR) FT = Función con valores de tabla de ensamblado (CLR) IF = Función SQL con valores de tabla insertados IT = Tabla interna P = Procedimiento almacenado de SQL PC = Procedimiento almacenado del ensamblado (CLR) PG = Guía de plan PK = Restricción PRIMARY KEY R = Regla (estilo antiguo, independiente) RF = Procedimiento de filtro de replicación S = Tabla base del sistema SN = Sinónimo SQ = Cola de servicio TA = Desencadenador DML del ensamblado (CLR) TF = Función con valores de tabla SQL TR = Desencadenador DML de SQL TT = Tipo de tabla U = Tabla (definida por el usuario) UQ = Restricción UNIQUE V = Vista X = Procedimiento almacenado extendido

Tabla 2.4: sys.objects

type_desc	**nvarchar(60)**	Descripción del tipo de objeto: AGGREGATE_FUNCTION CHECK_CONSTRAINT DEFAULT_CONSTRAINT FOREIGN_KEY_CONSTRAINT SQL_SCALAR_FUNCTION CLR_SCALAR_FUNCTION CLR_TABLE_VALUED_FUNCTION SQL_INLINE_TABLE_VALUED_FUNCTION INTERNAL_TABLE SQL_STORED_PROCEDURE CLR_STORED_PROCEDURE PLAN_GUIDE PRIMARY_KEY_CONSTRAINT RULE REPLICATION_FILTER_PROCEDURE SYSTEM_TABLE SYNONYM SERVICE_QUEUE CLR_TRIGGER SQL_TABLE_VALUED_FUNCTION SQL_TRIGGER TABLE_TYPE USER_TABLE UNIQUE_CONSTRAINT VIEW EXTENDED_STORED_PROCEDURE
create_date	**datetime**	Fecha de creación del objeto.
modify_date	**datetime**	Fecha en que se modificó el objeto por última vez con una instrucción ALTER. Si el objeto es una tabla o una vista, modify_date también cambia cuando se crea o modifica un índice clúster en la tabla o la vista.

2.10.1. Metodología Evolutiva Incremental (MEI)

Es una metodología sistémica de desarrollo e implantación de sistemas, que combina las técnicas estructuradas con el prototipadoén un contexto metodológico evolutivo-incremental. (Véase Figura 2.4) Esta metodología está apoyada en dos analogías paradigmáticas: la teoría evolutiva (con origen en ciencias biológicas) y la planificación incremental (con orígenes en economía y teoría de decisiones), de ahí que su denominación sea Evolutivo-Incremental. [25]

Algunas de las fases principales de esta metodología son: Análisis, Diseño de prototipos evolutivos e implementación del sistema.

Fase de Análisis

Durante el proceso de análisis se realiza un estudio de la situación actual del proceso en el dominio del problema, sobre los cuales se desea dar soporte a través del uso de las tecnologías de la información, específicamente de un sistema de información automatizado; y a su vez, la organización y definición de las necesidades funcionales, no funcionales y de información prioritarias para los dueños de los procesos.

Diseño de prototipos evolutivos

En esta faase se elaboran los prototipos del model del sistema propuesto elaborados en la fase de análisis, los cuales serán refinados iterativamente (prototipos y modelos del sistema) a través de iteraciones. El prototipo y sus respectivos modelos propuestos, progresan en forma evolutiva.

Implementación del sistema.

En esta fase se realizan las pruebas finales del software desarrollado y base de datos que garanticesn un óptimo funcionamiento del sistema. [26]

Algunas de las características de la metodología son:

- Se deriva de la metodología estructurada
- Permite seguir secuencias ascendentes o descendentes en las etapas del desarrollo
- Permite cumplir etapas o fases en paralelo, por lo que es más flexible que la estructurada.[27]

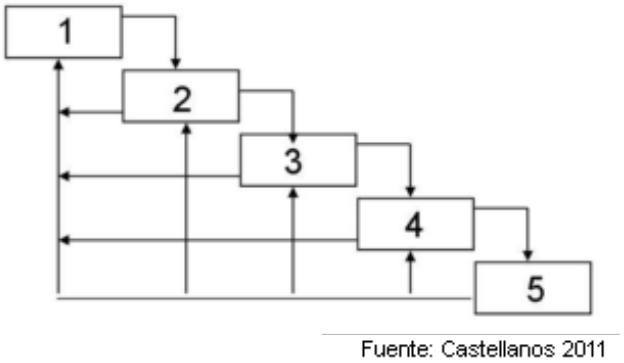


Figura 2.4: Esquema de una metodología evolutiva incremental de desarrollo de sistemas (Fuente: Castellanos 2011)

2.11. Términos y Definiciones

Independencia lógica: Posibilidad en donde diferentes procesos pueden tener diferentes versiones lógicas de una misma base de datos, en donde estas mismas versiones se puedan mantener de forma independiente a la base de datos, y entre ellas mismas. Lo cual da flexibilidad y elasticidad a los cambios lógicos (Hay independencia lógica cuando los usuarios no se ven afectados por los cambios en el nivel lógico de la base de datos). [9]

Atomicidad: La atomicidad implica que las transacciones son un "todo o nada" de entidad que lleva acabo todos los pasos o ninguno en absoluto.[11] Migraciones (También conocido como evolución del esquema)

SMO (Schema Modification Operators): Propuestos por Shneiderman y Thomas en 1982, son operadores que capturan la esencia del esquema actual, pero también expresan y

describen la evolución del esquema de bases de datos que no fueron modelados en enfoques anteriores y dan soporte a un historial de peticiones a través de las versiones.[28]

Capítulo 3

Metodología

3.1. Investigación del estado del arte y selección de una Técnica

Para el desarrollo de este trabajo, se seleccionó una investigación de tipo documental propositiva.

Documental porque el instrumento predominante para analizar y seleccionar una solución al problema, consistió en la revisión de libros y artículos relacionados con la evolución de los esquemas y el versionamiento de bases de datos. En donde se pudo indagar sobre los estudios previos realizados y las soluciones propuestas por otros autores.

Propositiva porque a raíz de la investigación realizada se propuso una Metodología para agilizar el proceso de actualización y control de esquemas bases de datos que utilizan los sistemas desarrollados por la empresa *SDPoint*.

3.2. Descripción del caso de estudio y planeación del desarrollo

Para la elaboración de la solución se seleccionó una metodología evolutiva incremental (MEI), la cual permite adaptarse fácilmente a los estándares de programación y diseño de la empresa objeto de estudio.

Se decidió utilizar esta metodología por la flexibilidad que presenta a la hora de la construcción del producto, el cual se especifica de forma general pero evoluciona a lo largo del proyecto.

El proyecto fue dividido en pequeñas partes que a su vez comprenden una pequeña entrega funcional de la solución, tales son descritos a continuación:

- Definir la estructura de metadatos
- Establecer los operadores SMO que van a utilizarse
- Identificar el mecanismo para realizar la implementación de los operadores.
- Diseñar la estructura de la base de datos de desarrollo, que incluye la implementación de los operadores seleccionados.

  Tablas (Crear, Eliminar, Renombrar)

  Columnas (Crear, Eliminar, Renombrar, Modificar)

  Funciones (Crear, Eliminar, Renombrar)

  Procedimientos almacenados (Crear, Eliminar, Renombrar)

  Llaves Primarias y Foráneas (Crear, Eliminar, Renombrar)

  Vistas (Crear, Eliminar)

Figura 3.1: Actividades para cada Iteración

- Diseñar la estructura de la base de datos en producción.

3.3. Desarrollo de Prototipos

Cada iteración del desarrollo constará de una serie de actividades que permitirán asegurar la calidad de la solución a fin de cumplir con cada una de las entregas que se establecieron, que además puede ser observado en la Figura 3.1.

Análisis de la solución. Durante esta etapa se determinan con precisión cada una de las tareas que deberá realizar la solución y como se logrará realizarlas. Y provee la información necesaria para las siguientes etapas.

Diseño de la solución. En base a la selección de la etapa anterior, se determinó el flujo del proceso que se requiere para realizar cada una de las tareas con apoyo de la técnica de diagramas de flujo listos para pasarse a la etapa de desarrollo.

Desarrollo de la solución. Etapa que consiste propiamente en el proceso de desarrollo del producto de software, siguiendo los requerimientos establecidos en la etapa anterior.

Pruebas. Durante esta etapa, la versión del software creada se someterá a diversas pruebas de calidad, funcionalidad y de concepto, con la finalidad de detectar las debilidades y errores para ser corregidos. Los resultados de las pruebas serán registradas y anexadas a este documento.

3.4. Herramientas para el desarrollo

Durante la construcción del producto de software se utilizaron herramientas de desarrollo para apoyar las actividades de modelado obtenidos de los requerimientos, documentación, seguimiento de tareas, desarrollo de software, las cuales son descritas a continuación:

Microsoft SQL Server 2008 R2 El manejador de base de datos *Microsoft SQL 2008 R2* se utilizó como plataforma de desarrollo utilizando T-SQL como lenguaje de programación; A fin de mejorar el rendimiento de la metodología al registrar una versión o migrar una base de datos, así como dejar abierta la metodología, no a un lenguaje de programación, si no a un lenguaje de base de datos, con la posibilidad

ER-Studio Esta herramienta fue utilizada para realizar el modelado de las bases de datos que se necesitaron para diseñar e implementar la metdología de versionamiento de bases de datos basado en SMO, dado a la facilidad visual que el software tiene para crear diagramas Entidad-Reación, y la funcionalidad que provee de crear directamente las bases de datos.

3.5. Análisis y Reporte de Resultados

En esta etapa se crea una serie de escenarios base para realizar pruebas de funcionalidad a la metodología y de esta forma corregir posibles problemas de programación y/o lógica en el desarrollo.

Capítulo 4

Versionamiento basado en SMO

Se seleccionó como base para nuestra solución la arquitectura *PRISM* y el concepto de Operadores Modificadores de Esquema (SMO) explicada anteriormente. El concepto de SMO consiste en crear operadores capaces de interpretar los cambios realizados en un esquema de bases de datos, y darle la capacidad de migrar de una versión a otra, abriendo la posibilidad de automatizar el proceso de versionamiento de bases de datos sin la necesidad de la intervención de recursos humanos para realizar dicha tarea.

4.1. Arquitectura de la solución

Diseñamos una solución ajustándonos a las tecnologías utilizadas en la empresa y sus necesidades basada en la arquitectura utilizada en *PRISM* por propuesta por Carlo A. Curino et. Como resultado tenemos una propuesta para el versionamiento de base de datos en SQL Server 2008 R2 misma que es representada en la Figura 4.1.

Esta arquitectura cuenta con tres elementos escenciales:

1. Una colección de datos que almacenan la estructura de la base de datos versionada a

través del tiempo, la cual se denominó Metadatos.

2. Un mecanismo para versionar bases de datos, que en este trabajo se refiere a una serie de procedimientos almacenados contenidos en la base de datos de desarrollo.

3. Y una serie de *SMO* que se encuentran dentro de la base de datos de los usuarios finales.

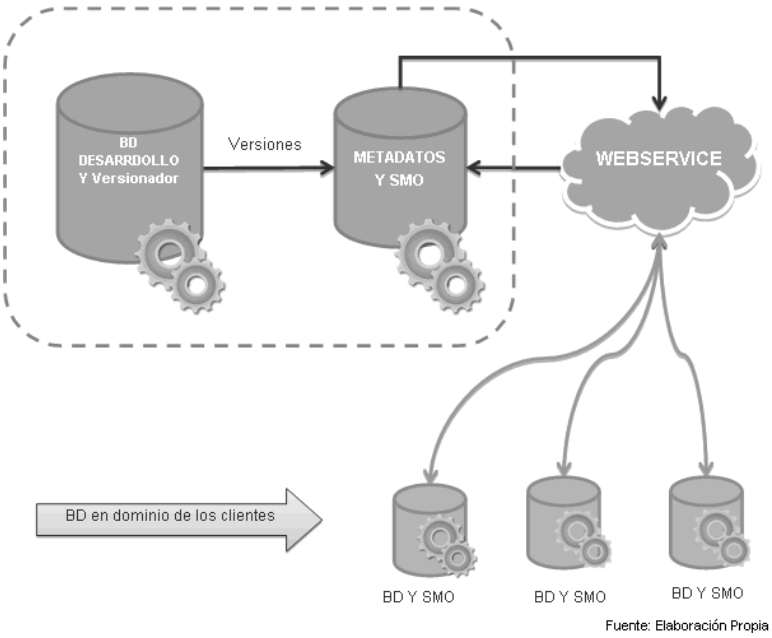


Figura 4.1: Arquitectura de la solución

Como funciona? la persona se sube, pedalea (describir como se utiliza la maquina)

4.2. Mecanismo para versionar

Tal como se mencionó anteriormente, dentro de las bases de datos de desarrollo hay procedimientos encargados de analizar la base de datos actual contra la última versión registrada en los metadatos y de crear reglas de actualización utilizando los *SMO* en dos sentidos:

1. Creación del elemento registrado

2. Eliminación del mismo

Esto permite retroceder entre versiones de una forma conveniente y sin la necesidad de tener que comparar nuevamente las bases de datos.

4.2.1. Procedimientos almacenados para versionar

Estos procedimientos se encargan de registrar una versión de la base de datos y se encuentran únicamente contenidos en la base de desarrollo (Véase Figura 4.2), misma que se considera la última versión disponible y es utilizada por los programadores del sistema de Software.

Figura 4.2: Base de datos de desarrollo y procedimientos para versionar

Su principal tarea es identificar cualquier diferencia en la base de datos, comparándola contra la última versión registrada en los metadatos. También, crear las reglas de actualización que competen a tablas, columnas, llaves primarias y foráneas, vistas y rutinas, para almacenarlas en los metadatos, etiquetándola como la última versión.

Para realizar esta tarea, cada procedimiento almacenado contenido dentro de *VersionarDB* realiza consultas directamente los metadatos de Microsoft SQL Server que se encuen-

tran en las tablas y vistas de sistema, tales como *INFORMATION_SCHEMA* y *sys.objects*, para comparar dicha estructura con nuestros Metadatos.

A continuación se describe cada uno de estos procedimientos, se incluyen diagramas de flujo de cada uno de ellos, sin embargo, es importante mencionar que por cuestiones de espacio en el documento, el estándar que fue utilizado no es en orden descendente.

VersionarDB

El objetivo de este procedimiento almacenado es ejecutar de forma jerárquica una serie de algoritmos que analizan los objetos encontrados en la base de datos de desarrollo y registrar una nueva versión. (Véase Figura 4.3)

Figura 4.3: Algoritmos que ejecuta VersionarDB

Parámetros del procedimiento VersionarDB

DB (varchar(100)) *Nombre de la base de datos que es versionada.*

VersionaTabla

Este procedimiento se encarga de encontrar todas las tablas que fueron creadas, aquellas que cambiaron de nombre e incluso las tablas que fueron eliminadas en la versión actual de la bse de datos de desarrollo. Almacena en los metadatos estas diferencias y las reglas de actualización y de eliminación de las mismas. (Véase Figura 4.4).

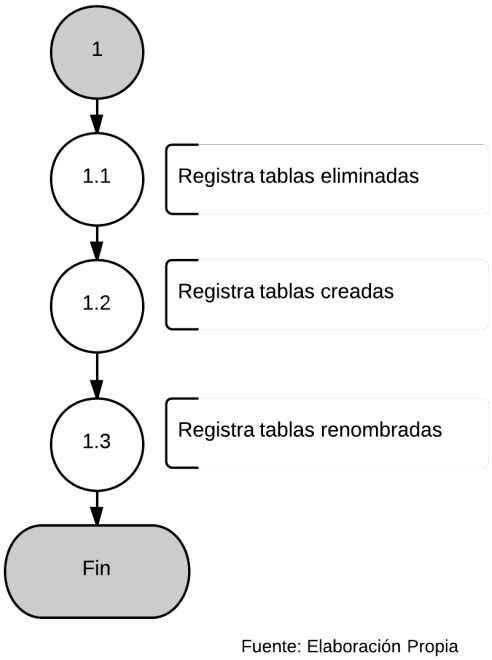


Figura 4.4: Diagrama de flujo de *VersionaTabla*

Parámetros del procedimiento *VersionaTabla* Los parámetros que pertenecen a *VersionaTabla* son calculados y enviados directamente por el procedimiento que lo ejecuta, en este caso *VersionaDB*:

Version (int) Última versión encontrada en los metadatos

DB_id (int) Identificador de la base de datos, de acuerdo a los metadatos

Este procedimiento ejecuta una serie de pasos de forma ordenada que son descritos a continuación (Véase Figura 4.4), esto con la finalidad de registrar correctamente todos los movimientos encontrados en las tablas de la base de datos versionada, de proporcionar y almacenar en los metadatos los *SMO* que necesitan ser ejecutados ordenadamente para evitar colisiones cuando se realice una actualización de una base de datos a esta versión.

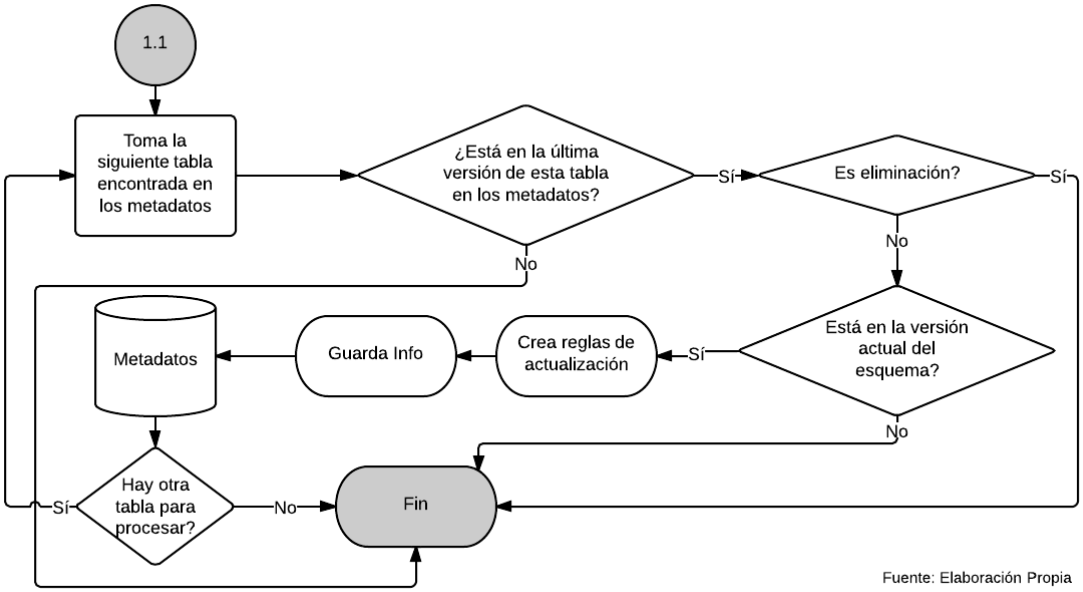


Figura 4.5: Diagrama de flujo de Registra Tablas Eliminadas

1. Registra Tablas Eliminadas. Durante este paso, el procedimiento se encarga encontrar todas las tablas que no aparecen en la versión de base de datos que se está versionando en comparación a la última versión registrada en los Metadatos identificándolas como tablas que fueron eliminadas (Véase Figura 4.5.)

2. Registra Tablas Creadas. En esta parte el procedimiento se encarga de buscar y registrar todas las tablas que se encuentran en la versión actual, las cuales no existen en la última versión registrada en los Metadatos, considerándolas como tablas nuevas. (Véase Figura 4.6.)

3. Registra Tablas Renombradas. En este paso, el versionador identifica todas las tablas que cambiaron de nombre en la nueva versión de base de datos y registra los *SMO* necesarios para realizar dicho renombramiento (Véase Figura 4.7.)

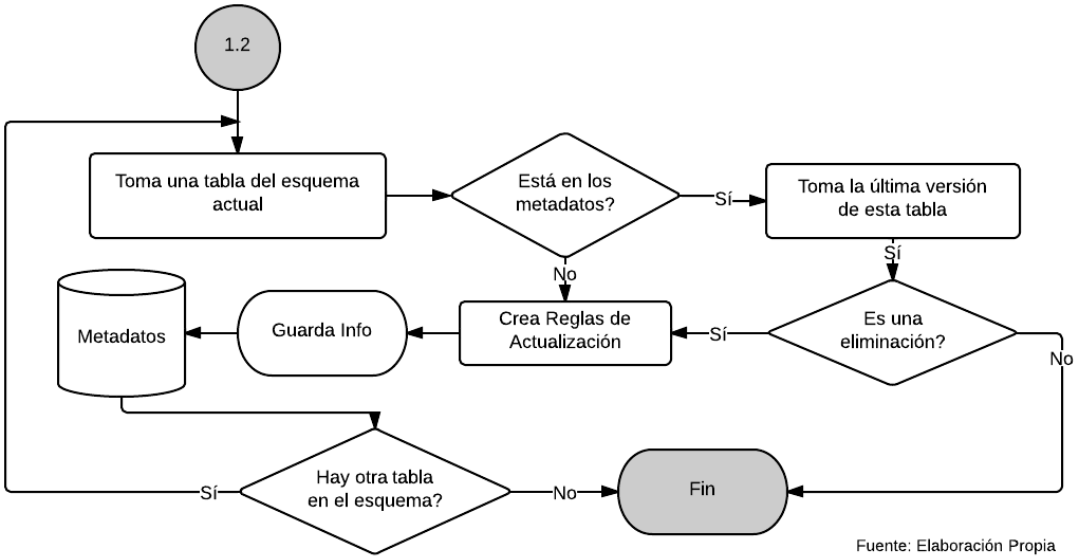


Figura 4.6: Diagrama de flujo de Registra Tablas Creadas

VersionaColumna

El procedimiento almacenado *VersionaColumna* se encarga de buscar todos los cambios realizados en relación a las columnas contenidas en cada una de las tablas entre la base de datos de desarrollo y la última versión almacenada en los metadatos. Esto incluye columnas nuevas, columnas que han cambiado de nombre y columnas que fueron eliminadas durante el desarrollo.

Parámetros del procedimiento almacenado *VersionaColumna*

- Version (int)
- DB_id (int)

Este procedimiento almacenado, sigue una serie de pasos ordenados para registrar estos cambios en busca de no probocar colisiones a la hora de realizar una migración de versión.

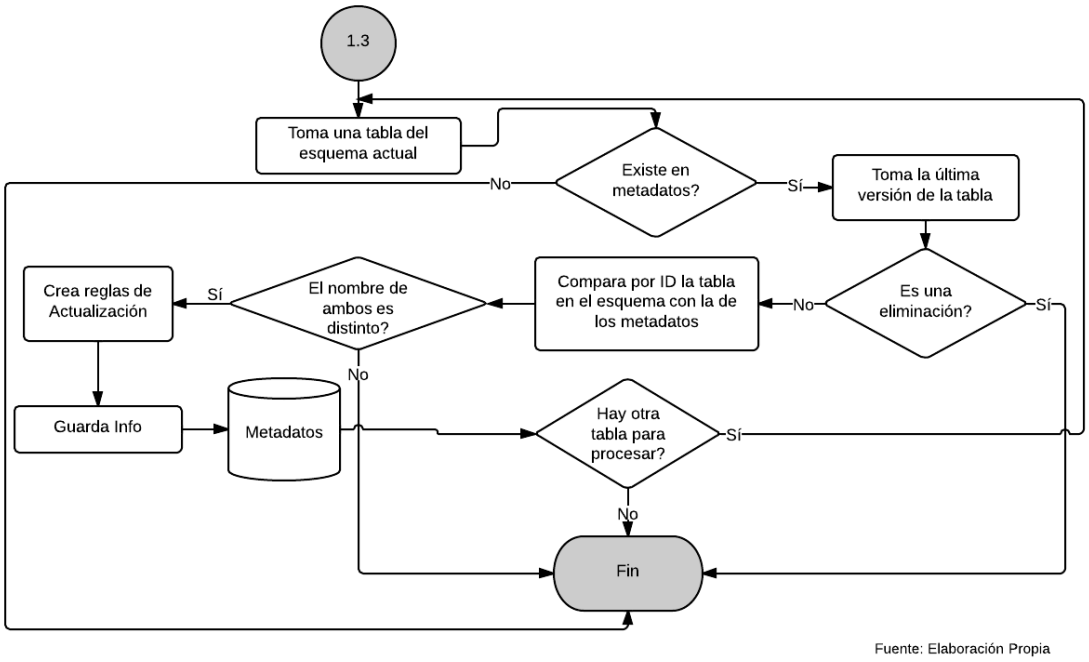


Figura 4.7: Diagrama de flujo de Registra Tablas Renombradas

(Véase Figura 4.8)

1. Registra Columnas Eliminadas Durante este paso, el procedimiento se encarga de identificar todas aquellas columnas que existen en la versión anterior registrada en los *Metados*, pero que no se encuentran en la nueva versión y las marca como columnas eliminadas creando la sintaxis de actualización. (Véase Figura 4.9).

2. Registra Columnas Creadas En esta parte del procedimiento para versionar columnas se identifican todas las columnas nuevas comparándolas con las tablas existentes en la versión anterior. (Véase Figura 4.10).

3.Registra Columnas Renombradas En caso de que la nueva versión de base de datos haya realizado alguna modificación en el nombre de las columnas durante este paso serán identificadas y registradas para la nueva versión. (Véase Figura 4.11).

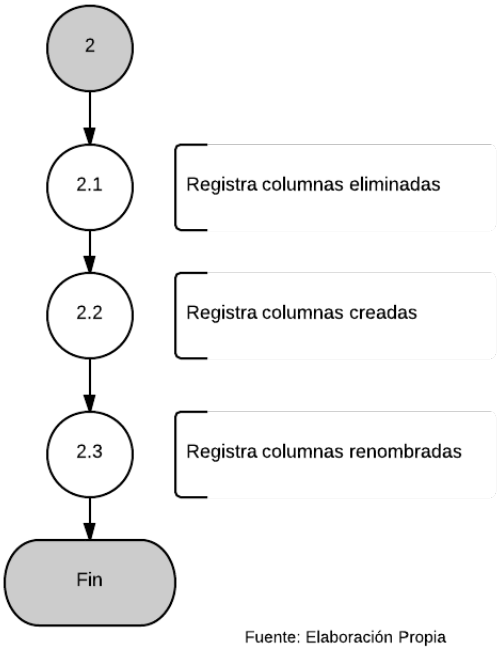


Figura 4.8: Diagrama de flujo *VersionaColumna*

VersionaPrimaria

El procedimiento almacenado *VersionaPrimaria* fue diseñado para identificar y registrar todos los cambios en la base de datos que está siendo versionada con relación a las llaves primarias que tienen cada una de las tablas, las operaciones que fueron consideradas para este caso son la creación de llaves primarias, la eliminación de llaves primarias e incluso el cambio de nombre de llaves primarias. (Véase Figura 4.12).

Parámetros del procedimiento almacenado *VersionaPrimaria*

Version (int) . Número de versión a la que se desea migrar.

DB_id (int) . Identificador de la base de datos en los Metadatos.

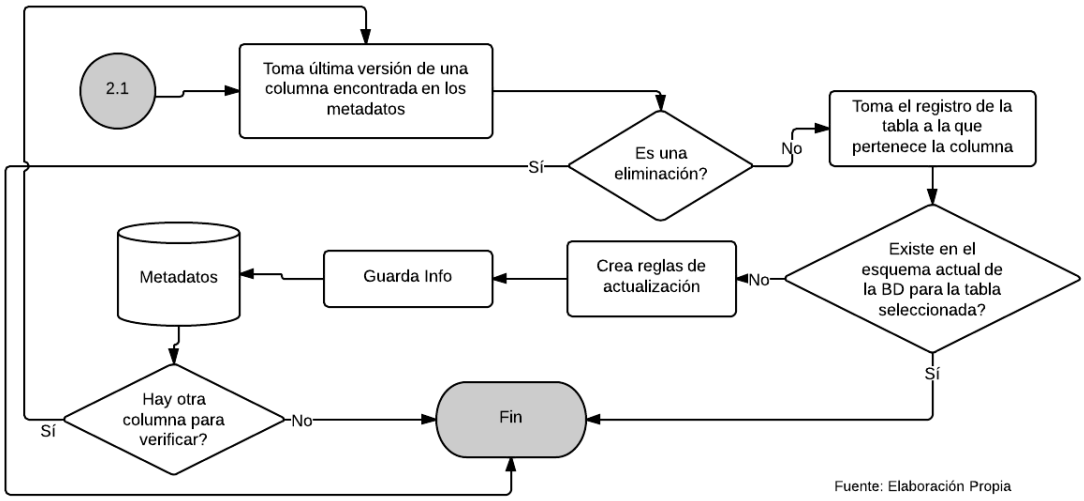


Figura 4.9: Diagrama de flujo para registrar columnas eliminadas

Los pasos que se realizan dentro del procedimiento son descritos a continuación y se ejecutan en tal orden. (Véase Figura 4.12)

1. Registra Llaves Primarias Eliminadas Durante esta etapa del proceso de registro de cambios en llaves primarias, el procedimiento almacenado identifica todas aquellas llaves que fueron eliminadas, es decir, que no existen en la base de datos que esta siendo versionada, pero existen en la versión de base de datos anterior. (Véase Figura 4.13).

2. Registra Llaves Primarias Creadas Cuando el procedimiento llega a este punto, se identifican todas aquellas llaves primarias que aparecen en la nueva versión de base de datos, y se crean las reglas de actualización para guardar en los *Metadatos*. (Véase Figura 4.14).

3. Registra Llaves Primarias Renombradas Durante esta parte del proceso, se registran todos los cambios de nombre efectuados en las llaves primarias de cada una de las tablas, realizando la comparación entre la base de datos actual contra la base de datos previa a esta. (Véase Figura 4.15).

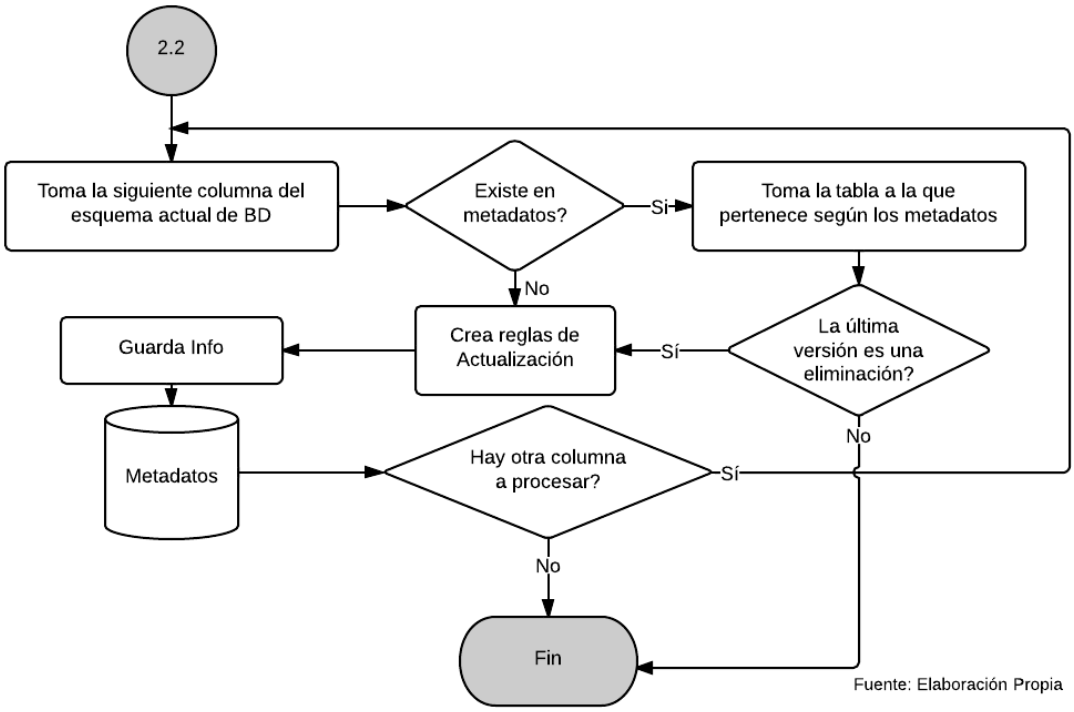


Figura 4.10: Diagrama de flujo para registrar columnas creadas

4. Registra Llave Primaria Modificada Cuando hablamos de una llave primaria modificada, nos referimos a las columnas que lo componen, escenario que puede llegar a darse cuando hay evolución en los sistemas de información y por ende en las bases de datos.

Esta parte del proceso de versionar llaves primarias se encarga de identificar todas aquellas llaves primarias que han sufrido una modificación en su estructura, que no necesariamente se refiere al cambio de nombre. (Véase **??**).

VersionaForanea

Procedimiento que se encarga de registrar todas las llaves foráneas encontradas en una versión de base de datos aun no etiquetada. Así como identificar aquellas ya existentes que fueron eliminadas, renombradas o que las columnas referenciadas han cambiado.

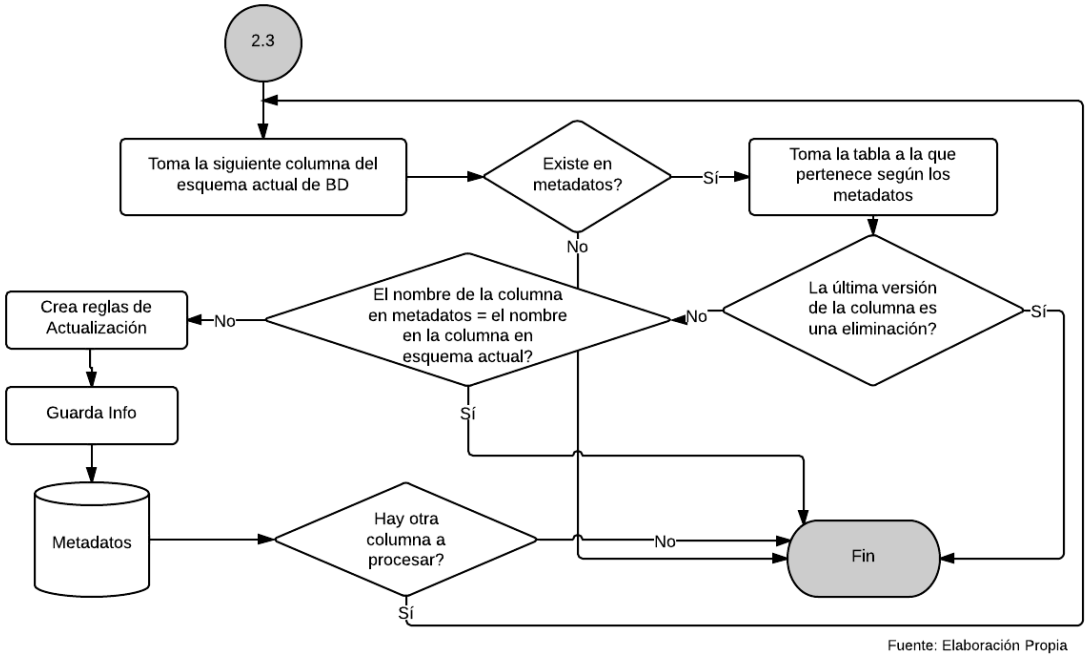


Figura 4.11: Diagrama de flujo para registrar columnas renombradas

Parámetros del procedimiento almacenado *VersionaForanea*

Version (int). Número de versión a la que se desea migrar.

DB_id (int). Identificador de la base de datos en los Metadatos.

Este procedimiento es auxiliado por tres subprocesos que a continuación son mencionados y que permiten registrar todos los cambios en las llaves foráneas. (Véase Figura 4.17)

1. Registra llaves foráneas eliminadas Durante esta parte del proceso de versionamiento de llaves foráneas se identifican todas aquellas llaves que desaparecen en la nueva versión de base de datos, y que existían en la versión anterior, se crean las reglas de actualización y son almacenadas en los *Metadatos.*(Véase 4.18).

2. Registra llaves foráneas creadas Cuando el procedimiento *VersionaForanea* llega a esta etapa del proceso, se realiza una revisión de las llaves foráneas entre tablas que no se

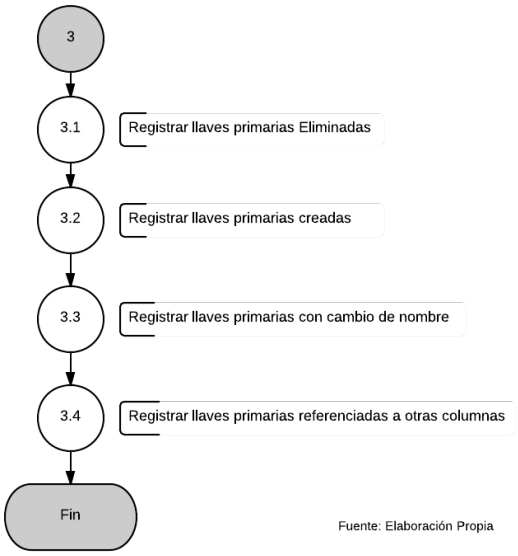


Figura 4.12: Diagrama de flujo para *VersionaPrimaria*

encuentran presentes en la versión anterior de base de datos, se registran todos estos casos y se crean las reglas de actualización. (Véase 4.19).

3. Registra llaves foráneas renombradas Para el caso particular de cambio de nombre de llaves foráneas, un caso poco común, pero posible y de importancia para el correcto funcionamiento de los otros casos, este procedimiento fue diseñado para identificar todas aquellas llaves foráneas presentes en ambas versiones (la versión nueva que está siendo creada y la versión previa a la misma), pero que poseen una diferencia en la definición de sus nombres. (Véase 4.20).

4. Registra llaves foráneas modificadas Todas aquellas llaves foráneas que presentan cambios de estructura, es decir, de columnas y/o tablas involucradas en esta llave, son identificadas en esta parte del proceso y se crean las reglas de actualización pertinentes. (Véase 4.21).

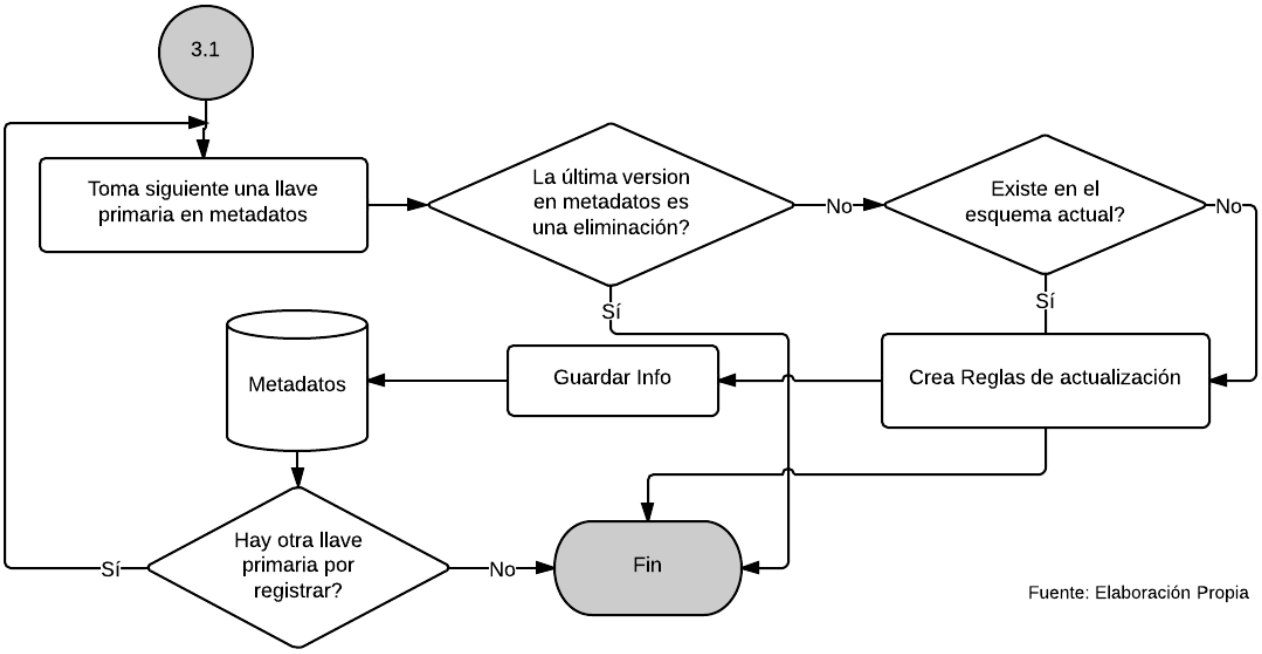


Figura 4.13: Diagrama de flujo para registrar llaves primarias eliminadas.

VersionaVista

Procedimiento almacenado que se encarga de registrar todos los cambios efectuados y encontrados en una nueva versión de base de datos, con respecto a las vistas en la base de datos.

Parámetros del procedimiento almacenado *VersionaVista*

Version (int). Número de versión a la que se desea migrar.

DB_id (int). Identificador de la base de datos en los Metadatos.

Las operaciones consideradas para el registro de cambios en las vistas, se resume en todas aquellas que son eliminadas, creadas o que su contenido es distinto en una nueva versión de base de datos, con respecto a la versión anterior que se encuentre en los *Metadatos*.

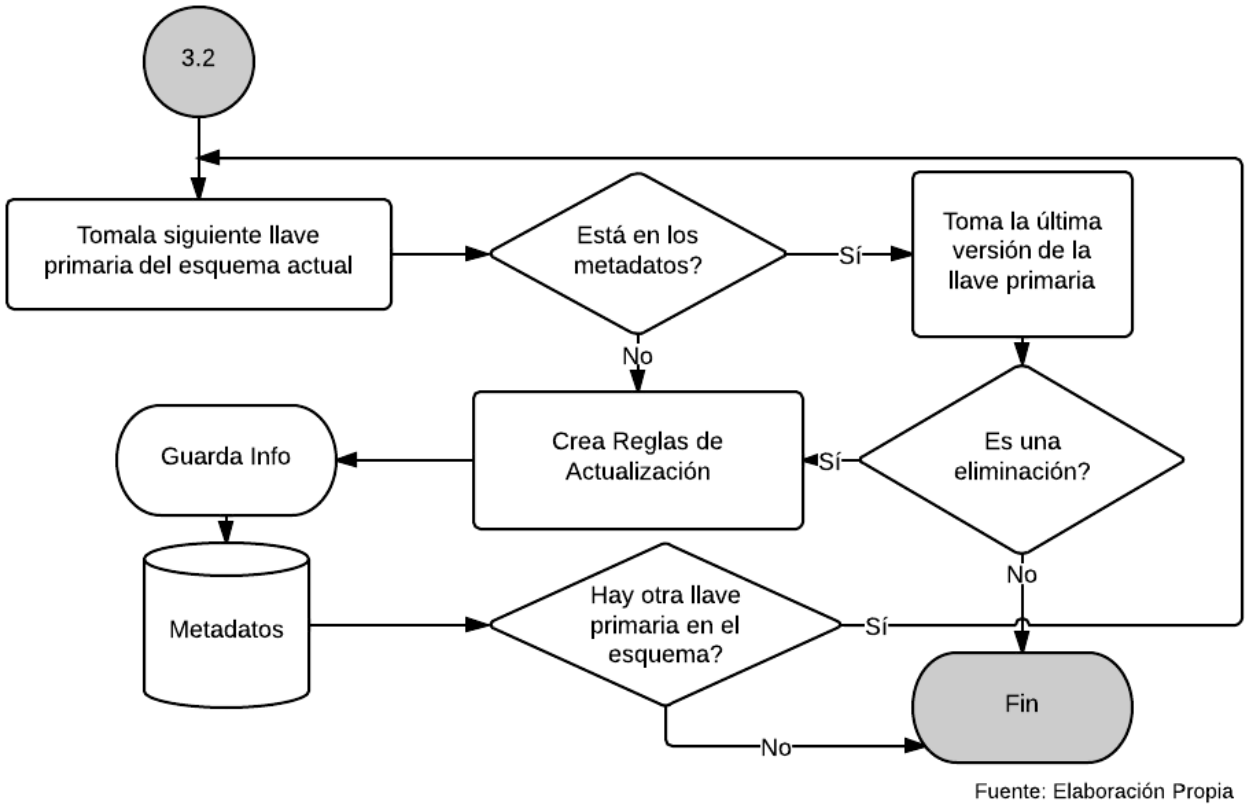


Figura 4.14: Diagrama de flujo para registrar llaves primarias creadas.

Este procedimiento se auxilia de tres subprocesos (Véase Figura 4.22) que son ejecutados de forma ordenada, los cuales son descritos a continuación:

1. Registra Vistas Eliminadas El primer paso para registrar una versión con cambios en las vistas de una base de datos, se encuentra en primeramente identificar todas aquellas vistas que desaparecen en la nueva versión y que se encontraban presentes en la versión anterior a la misma. Se crean las reglas de actualización y se registran en los *Metadatos* con la finalidad de proporcionar la migración de un estado a otro entre versiones. (Véase Figura 4.23).

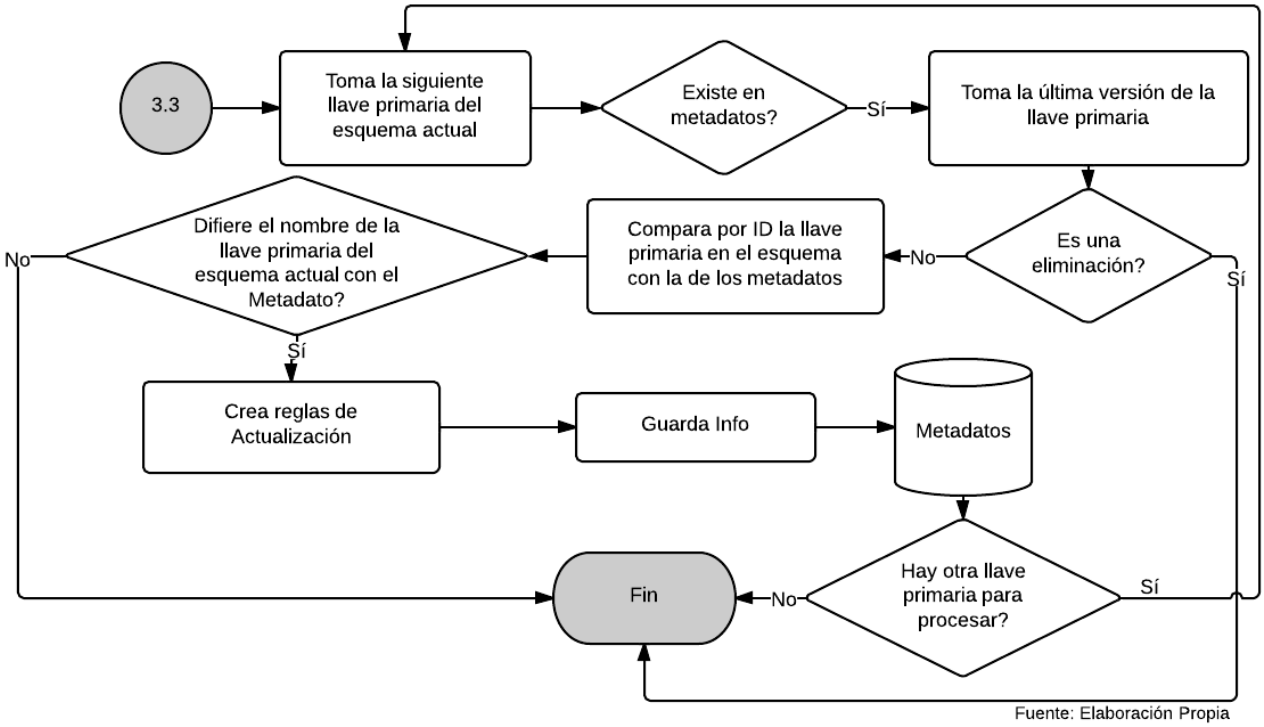


Figura 4.15: Diagrama de flujo para registrar llaves primarias renombradas.

2. Registra Vistas Creadas Durante esta parte del proceso de versionamiento, se identifican todas aquellas vistas nuevas, y se registran junto con sus reglas de actualización. (Véase Figura 4.24).

3. Registra Cambio en Vistas Existentes Por último, el procedimiento almacenado *VersionaVista* compara todas las vistas presentes en la nueva versión de base de datos, contra aquella previa en los *Metadatos* a fin de identificar aquellas vistas que presentan cambios en su estructura y registrando dichos cambios utilizando las reglas de actualizacion *SMO*. (Véase Figura 4.25).

Es importante mencionar, que para la información de esquema de base de datos que proporciona *Microsoft SQL Server 2008 R2* con respecto a las vistas, solo se consideran los casos de creación y eliminación de vistas descartando aquellos movimientos de cambio de

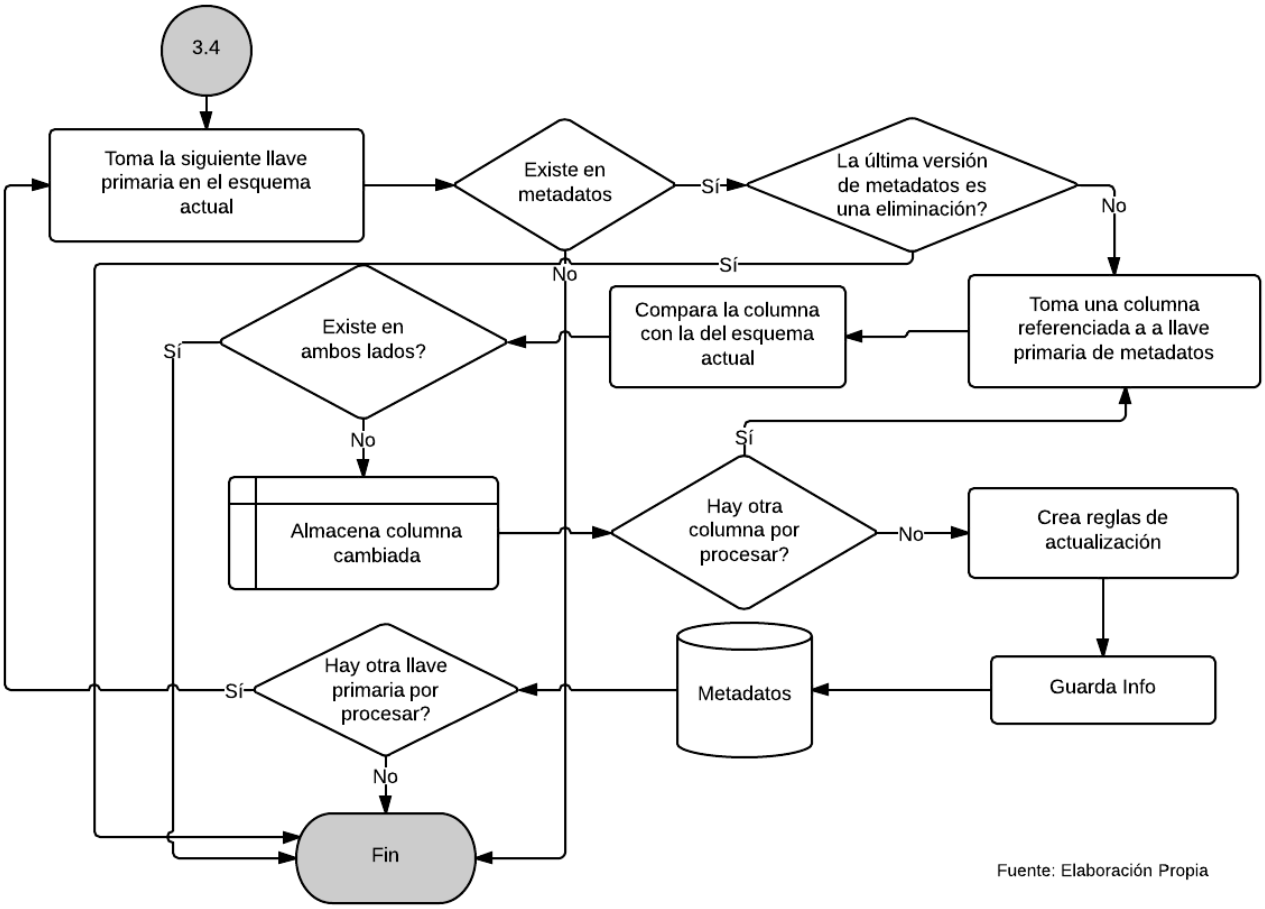


Figura 4.16: Diagrama de flujo para registrar llaves primarias renombradas.

nombre. Cuando una vista cambia de nombre, *SQL Server* lo registra como una vista nueva, lo que significa que el nombre anterior se identifica como una vista eliminada.

VersionaRutina

Este procedimiento almacenado se encarga de revisar la aparición o desaparición de rutinas en una base de datos, tomando en cuenta que la información en los metadatos de SQL Server 2008 se considera a una rutina como a todos los procedimientos almacenados y funciones definidas por el usuario. (Véase Figura 4.26).

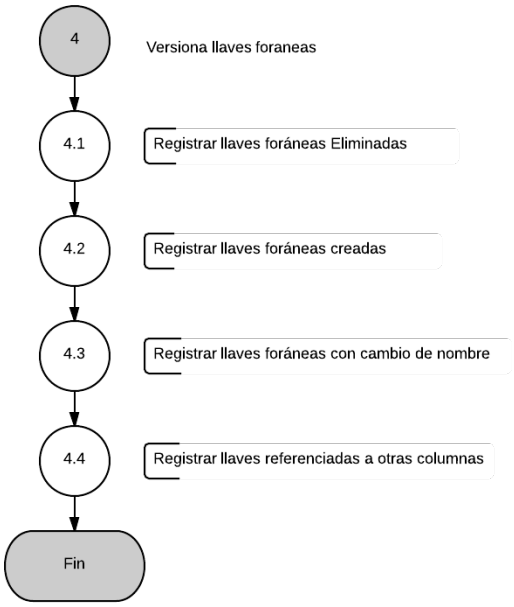


Figura 4.17: Diagrama de Flujo del procedimiento *VersionaForanea*

Dado que, cuando se realiza una modificación en el nombre de una rutina, SQL La identifica como una nueva, no se considera el escenario de 'Renombrar', sin embargo, el procedimiento que versiona si revisa el contenido y la estructura de las rutinas.

Parámetros del procedimiento almacenado *VersionaRutina*

- Version (int)
- DB_id (int)

Para lograr su objetivo, *VersionaRutina* ejecuta una serie de pasos de forma ordenada que son descritos a continuación:

1. Registra Rutinas Eliminadas El primer paso del procedimiento *VersionaRutina* se refiere a localizar todos aquellos procedimientos almacenados y/o funciones definidas por el usuario que desaparecen en la nueva versión de base de datos. (Véase Figura 4.27).

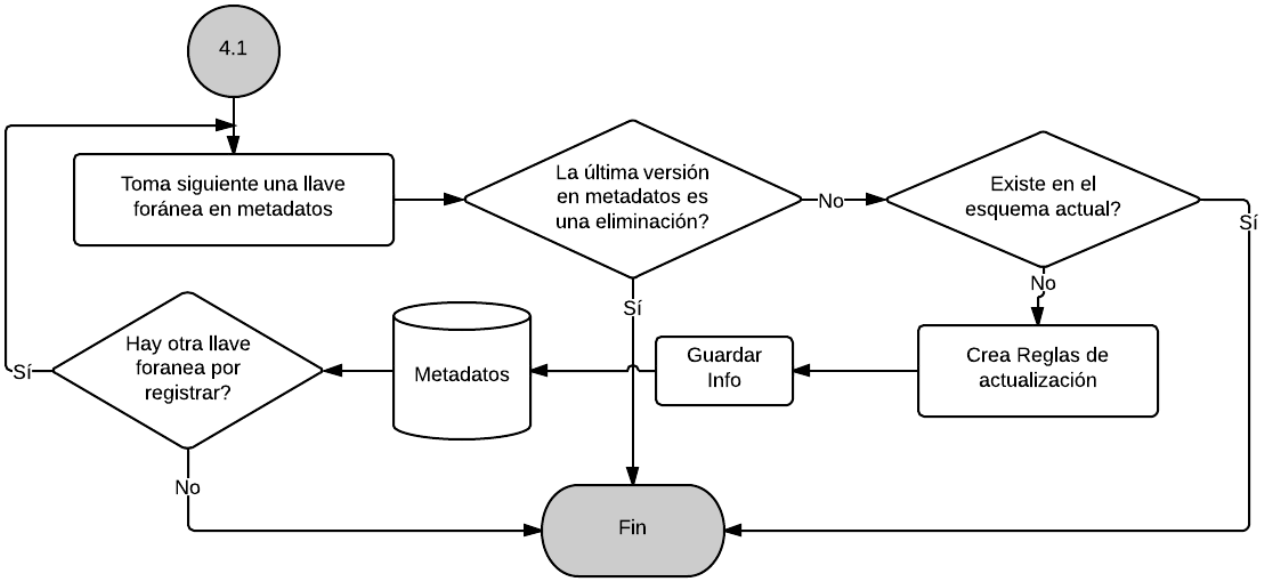


Figura 4.18: Diagrama de Flujo para registrar llaves foráneas eliminadas

2. Registra Rutinas Creadas Durante la segunda etapa, se realiza una revisión de todas aquellas rutinas que no se encuentran presentes en la versión de base de datos anterior pero que aparecen en la base de datos que está siendo versionada. Se crean las reglas de actualización *SMO* y se registran en los *Metadatos*. (Véase Figura 4.28)

3. Registra Cambio en Rutinas Existentes En el último paso, el proceso toma todas aquellas rutinas presentes en ambas versiones, la última versión en metadatos y la versión que está siendo creada y se realiza una revisión en cuanto a la estructura de la misma, de forma tal, que si se encuentra alguna diferencia se registra la rutina como parte de la nueva versión de base de datos. (Véase Figura 4.29).

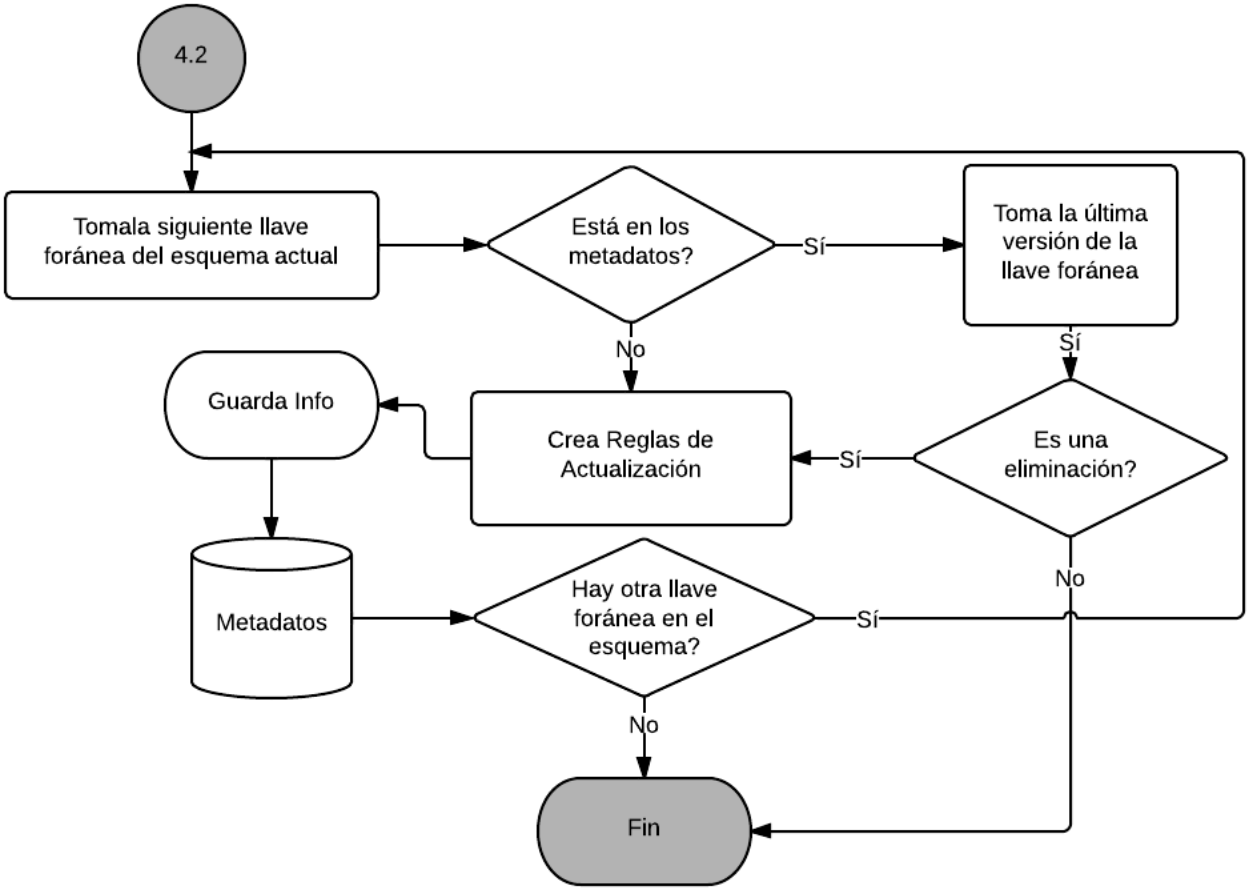


Figura 4.19: Diagrama de Flujo para registrar llaves foráneas creadas

4.3. Descripción de los SMO creados

Se crearon una serie de procedimientos almacenados denominados SMO, que actúan como intérprete entre los cambios encontrados en un esquema de base de datos y el lenguaje del motor de base de datos, en este caso T-SQL. Estos procedimientos están dentro de cada base de datos en producción bajo el esquema *"ver"* con la finalidad de proporcionar la funcionalidad a la base de datos de poder migrar de una versión a otra por "si misma".

La Tabla 4.1 explica los SMO dedicados exclusivamente para los movimientos en una *tabla* de la base de datos, son descritos con los parámetros que reciben y una breve definición

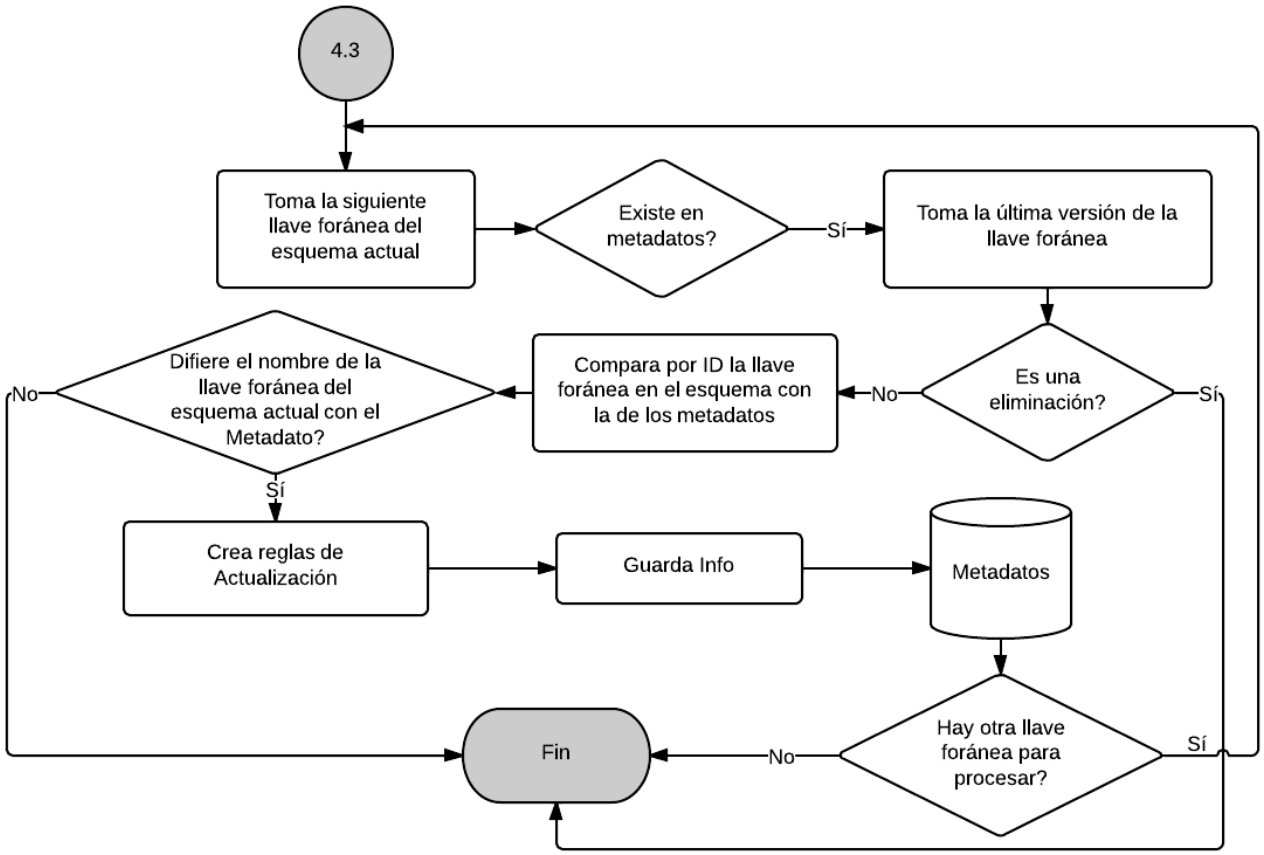


Figura 4.20: Diagrama de Flujo para registrar llaves foráneas renombradas

de cada uno de ellos.

La Tabla 4.2 contiene los SMO, sus parámetros y una breve descripción en cuanto a los movimientos que pueden ser efectuados en una columna de la base de datos.

En la Tabla 4.3 son explicados todos los procedimientos almacenados que son necesarios para realizar cambios en lo que compete a las llaves primarias que se encuentran en la base de datos.

Una vez creadas las tablas, sus columnas y sus llaves primarias, es posible crear las relaciones entre las tablas existentes, razón por la cual se crearon los SMO encargados de realizar los movimientos sobre las llaves foráneas, ya sea de inserción, actualización, modificación y

Tabla 4.1: SMO para actualizar tablas.

Nombre	Parámetros	Descripción
UN	TablaNombre (varchar(50)), Vid(int)	Procedimiento que se encarga de crear una tabla
UR	NombreOriginal (varchar(50)), NuevoNombre (varchar(50)), Vid(int)	Procedimiento auxiliar para cambiar el nombre de una tabla, si la necesidad de borrarla y volverla a crear.
UD	TablaNombre (varchar(50)), Vid int	Eliminación de tabla

Tabla 4.2: SMO para actualizar columnas.

Nombre	Parámetros	Descripción
CN	TablaNombre(varchar(50)), ColumnaNombre (varchar(50)), Tipo (varchar(50)), Null (varchar(50)), Default (varchar(50)), Agrega (bit)	Crear una nueva tabla
CD	TablaNombre(varchar(50)), ColumnaNombre (varchar(50)), Vid int	Elimina las columnas que ya no aparezcan en la versión a la que se está migrando.
CR	Tabla (varchar(50)), ColumnaOriginal (varchar(50)), ColumnaNueva (varchar(50))	Procedimiento que se utiliza para cambiar el nombre de una columna contenida en una tabla en particular.

Tabla 4.3: SMO para actualizar llaves primarias.

Nombre	Parámetros	Descripción
PKN	TablaNombre (varchar(50)), NombrePrimaria(varchar(50)), Csv_object_id_Columna(varchar(50)), Vid int	Crea llaves primarias, para la tabla y columnas que le sean solicitadas.
PKR	TablaNombre (varchar(50)), OldNombrePrimaria(varchar(50)), NewNombrePrimaria (varchar(50)), Vid int	Renombra llaves primarias
PKM	TablaNombre (varchar(50)), NombrePrimaria (varchar(50)), Csv_object_id_Columna(varchar(50))	Cambia las columnas de la llave primaria
PKD	TablaNombre (varchar(50)), NombrePrimaria (varchar(50)), Vid int	Cambia las columnas de la llave primaria

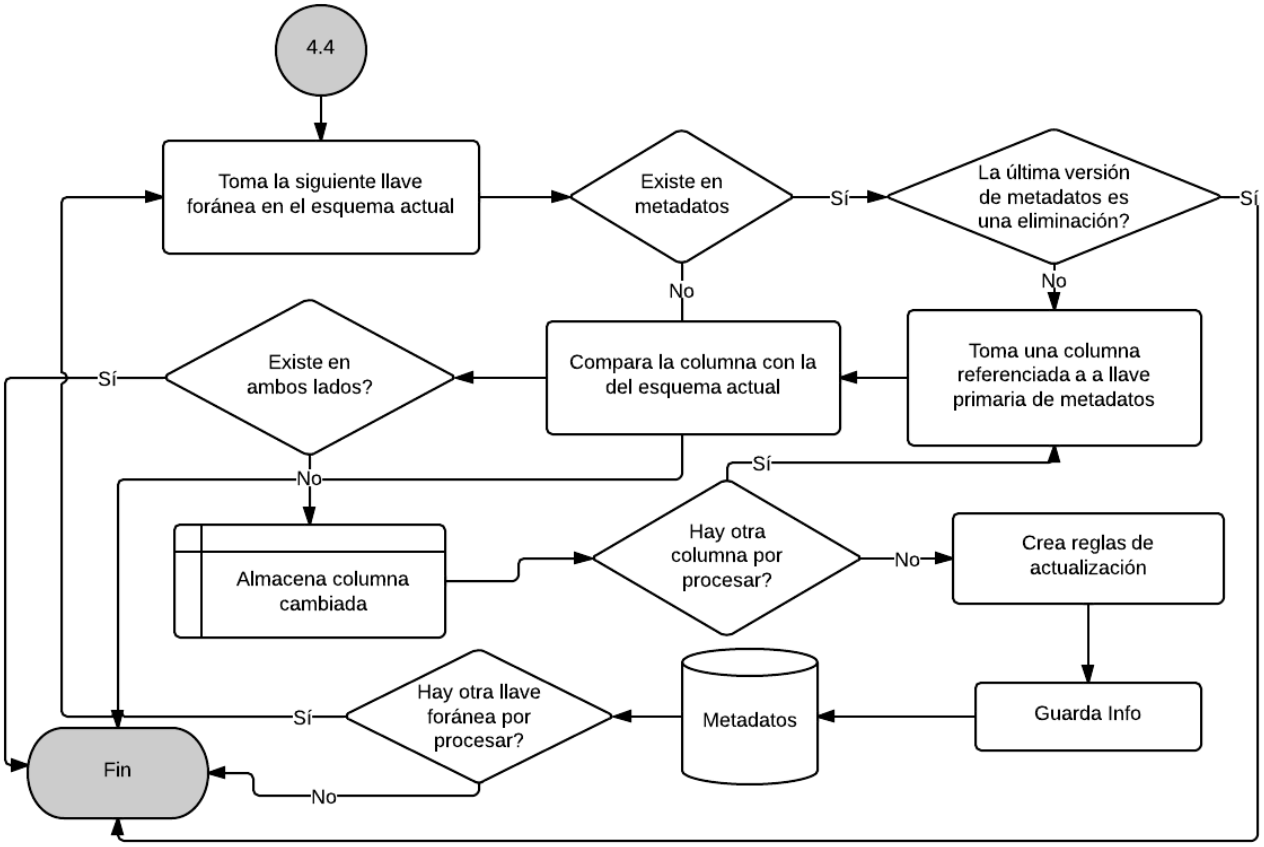


Figura 4.21: Diagrama de Flujo para registrar llaves foráneas modificadas

eliminación de las mismas. Esta información podrá ser visualizada en la Tabla 4.4.

En la Tabla 4.5 se encuentra la descripción de los SMO que se crearon para realizar los cambios encontrados en las vistas de la base de datos.

En la Tabla 4.6 usted podrá encontrar la descripción de los SMO creados para efectuar los cambios encontrados en una base de datos, en relación a las rutinas tanto como procedimientos almacenados, como funciones.

Con la finalidad de trabajar bajo un servidor de Microsoft SQL, éstos SMO fueron creados en procedimientos almacenados, auxiliados por 4 funciones principales: (hace falta algo aqui)

Tabla 4.4: SMO para actualizar llaves foráneas.

Nombre	Parámetros	Descripción
FKN	TablaPadreNombre (varchar(50)), TablaReferenciadaNombre (varchar(50)), NombreForanea (varchar(50)), Csv_object_id_ColumnaOrigen (varchar(50)), Csv_object_id_ColumnaReferenciada (varchar(50)), Vid int	Crea llaves foráneas
FKR	TablaNombre (varchar(50)), OldNombrePrimaria (varchar(50)), NewNombrePrimaria(varchar(50)), Vid int	Renombra llaves foráneas
FKM	TablaPadreNombre (varchar(50)), TablaReferenciadaNombre (varchar(50)), NombreForanea (varchar(50)), Csv_object_id_ColumnaOrigen (varchar(50)), Vid int	Cambia las columnas involucradas en llave foránea
FKD	TablaNombre (varchar(50)), NombreForanea (varchar(50)), Vid int	Elimina llaves foráneas

Tabla 4.5: SMO para actualizar vistas.

Nombre	Parámetros	Descripción
VN	NombreVista (varchar(50)), Script (varchar(max)), Vid int	Eliminación de una vista
VD	NombreVista (varchar(50)), Vid int	Cambio nombre de una vista
VM	NombreVista (varchar(50)), Script (varchar(max)), Vid int	Creación de una Rutina

Tabla 4.6: SMO para actualizar rutinas.

Nombre	Parámetros	Descripción
RN	NombreRutina (varchar(50)), Script (varchar(max)), Vid int	Eliminación de una Rutina
RD	NombreRutina (varchar(50)), Vid int	Cambio nombre de una Rutina
RM	NombreRutina (varchar(50)), Script (varchar(max)), Vid int	Eliminación de una vista

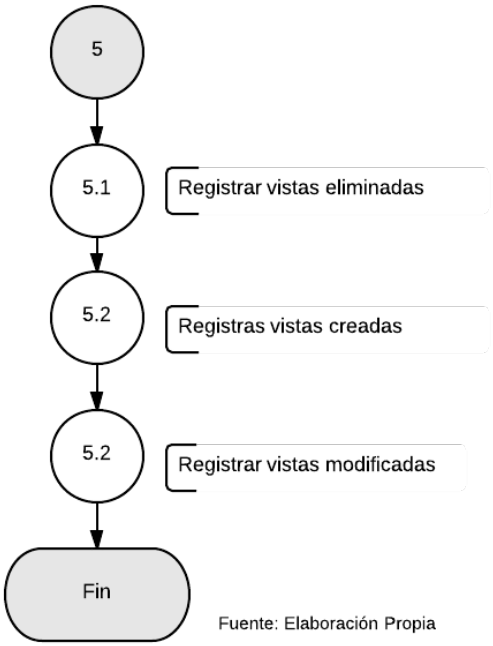


Figura 4.22: Diagrama de Flujo del procedimiento *VersionaVista*

4.4. Descripción de los artefactos necesarios para la implementación de la metodología

Dentro de la metodología que se propuso como solución al versionamiento de bases de datos empresariales para productos *SDPoint*, se diseñaron tres modelos entidad relación en donde se guarda la información de las bases de datos a través de su evolución, tanto los Metadatos, que contienen la descripción de la estructura de cada versión como el esquema base para las bases de datos que se encuentren en producción (generalmente en el dominio de los clientes).

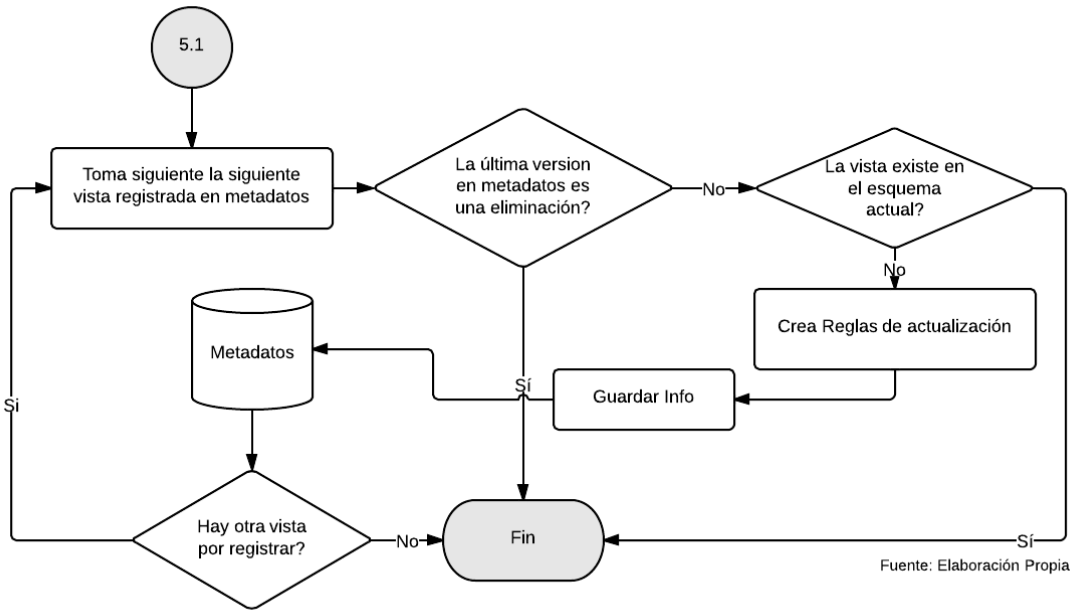


Figura 4.23: Diagrama de Flujo para registrar vistas eliminadas

Base de datos Metadatos

Con la finalidad de poder almacenar cada una de las versiones por las que una base de datos atraviesa, sus cambios y cuando ocurrieron. Se diseñó una base de datos a la cual se le denominó *Metadatos*, capaz de almacenar la información de las tablas, columnas, llaves primarias, llaves foráneas, vistas, procedimientos almacenados y funciones contenidas en la base de datos de desarrollo.

El diagrama entidad relación correspondiente a los Metadatos, se encuentra en el Apéndice **??**, al cual se podrá acudir para conocer de forma detallada ilustrado en la Figura A.1. Esta base de datos contiene las tablas necesarias para registrar una nueva versión de base de datos, con todos sus elementos de esquema.

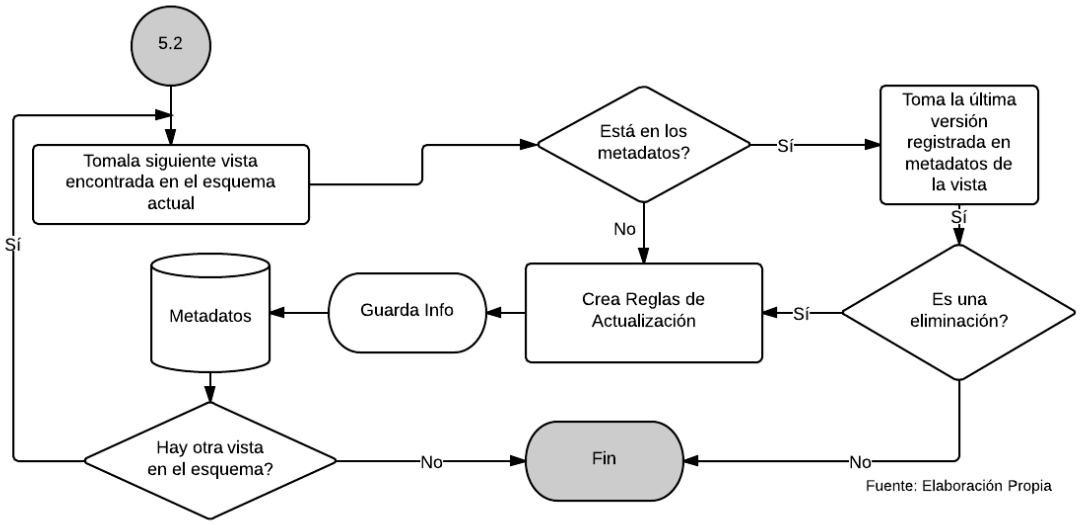


Figura 4.24: Diagrama de Flujo para registrar vistas creadas

Base de datos "base"para desarrollo

En cuanto al esquema de base de datos base que deberán tener las bases de datos en el ambiente de desarrollo que vayan a ser controladas, ésta deberá contar con los procedimientos que fueron descritos anteriormente, que corresponden únicamente a los siguientes: Todas las funciones y los procedimientos almacenados en la base de datos de desarrollo se encuentran bajo el dominio del esquema *ver*, esto con la finalidad de poder diferenciarlo entre las funciones y/o procedimientos que la base de datos pueda tener en consecuencia de la información del software al que corresponde. Así como la posibilidad de ocultarlas para ciertos usuarios, según sea necesario.

Base de datos "base"para producción

La base de datos *base* para producción consta de una tabla (Véase la Figura 4.30), la cual almacenará la información del versionamiento ocurrido en dicha base de datos, y es complementada por los operadores de evolución de esquema (*SMO*), que se encargan de

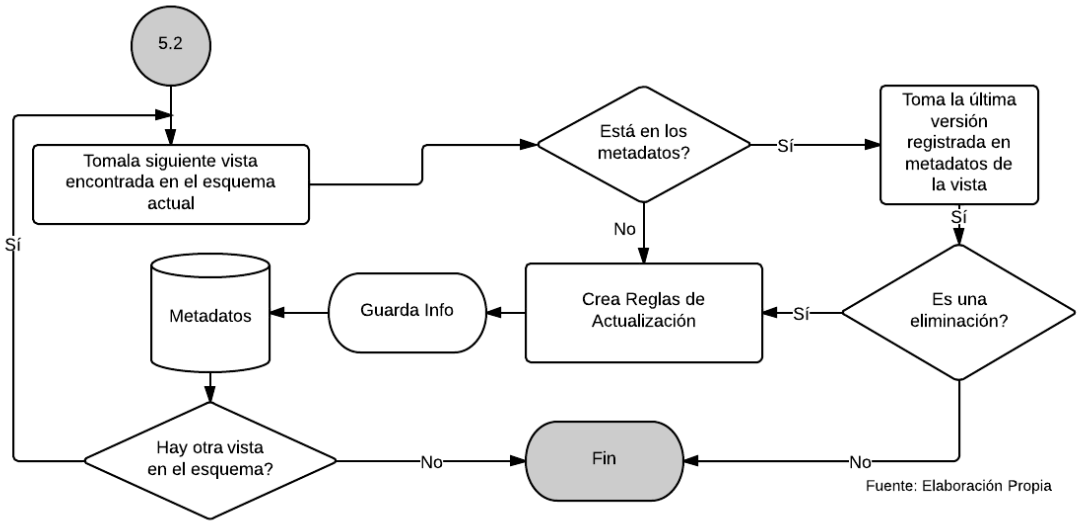


Figura 4.25: Diagrama de Flujo para registrar vistas modificadas

habilitarla para migrar de una versión a otra, tanto hacia adelante como hacia atrás, los cuales tendrán que ser incluidos en la base de datos como procedimientos almacenados bajo el esquema *ver* antes de querer migrar una nueva versión.

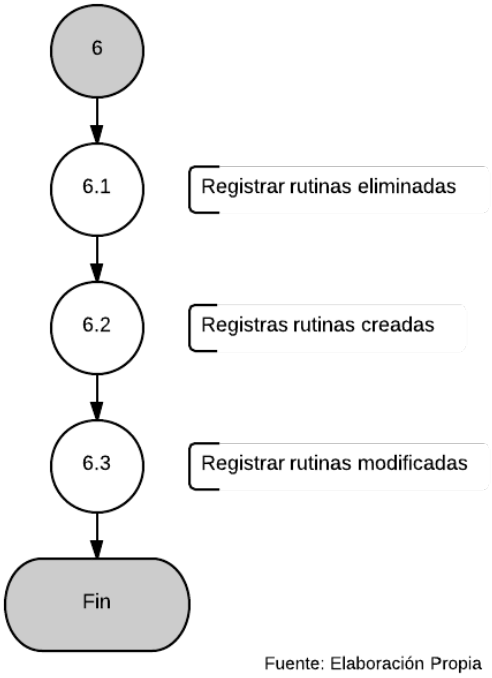


Figura 4.26: Diagrama de Flujo de *VersionaRutina*

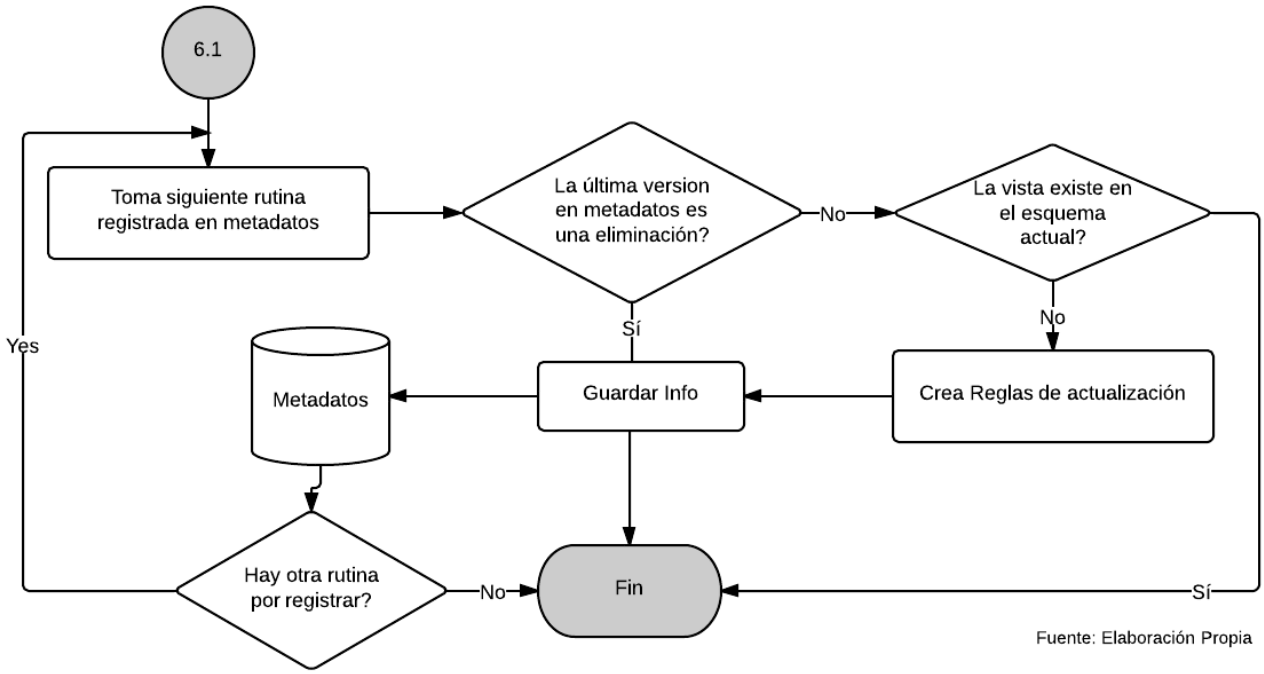


Figura 4.27: Diagrama de flujo para versionar rutinas eliminadas

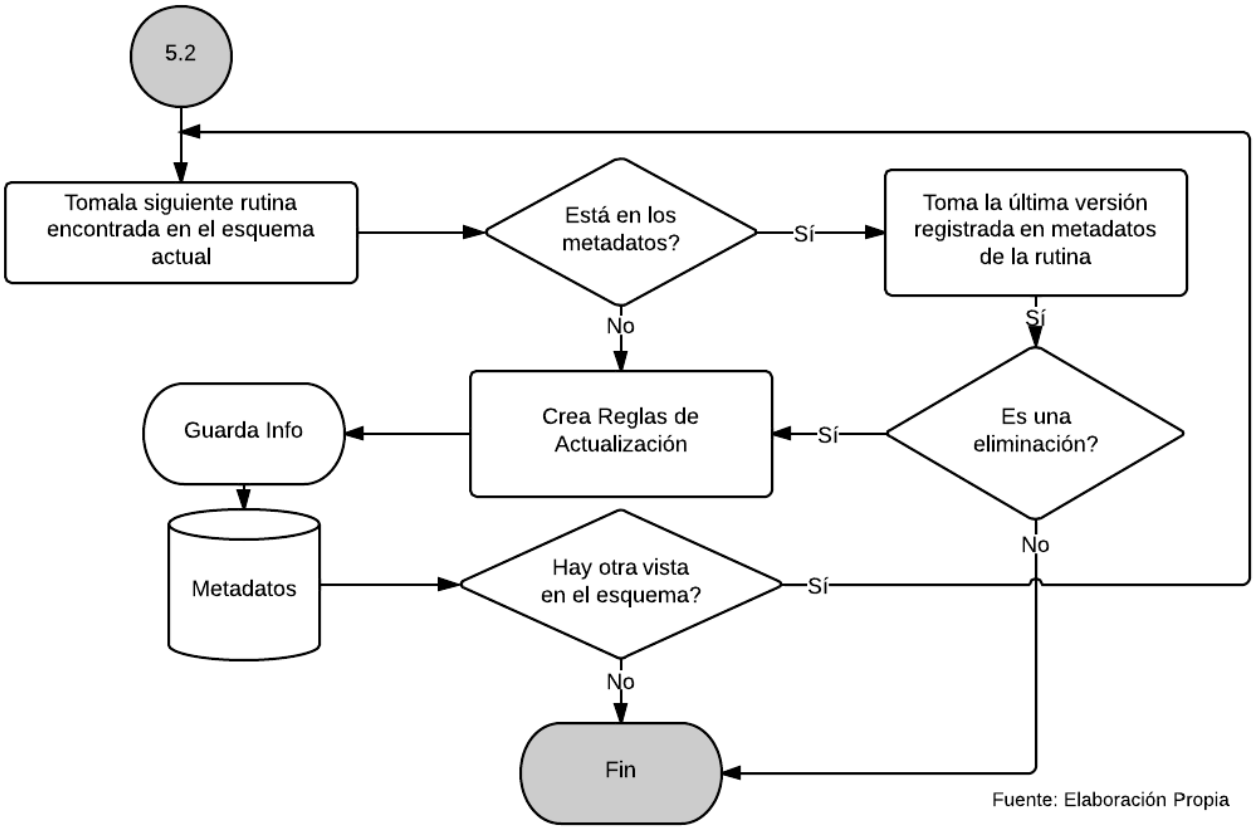


Figura 4.28: Diagrama de flujo para versionar rutinas creadas

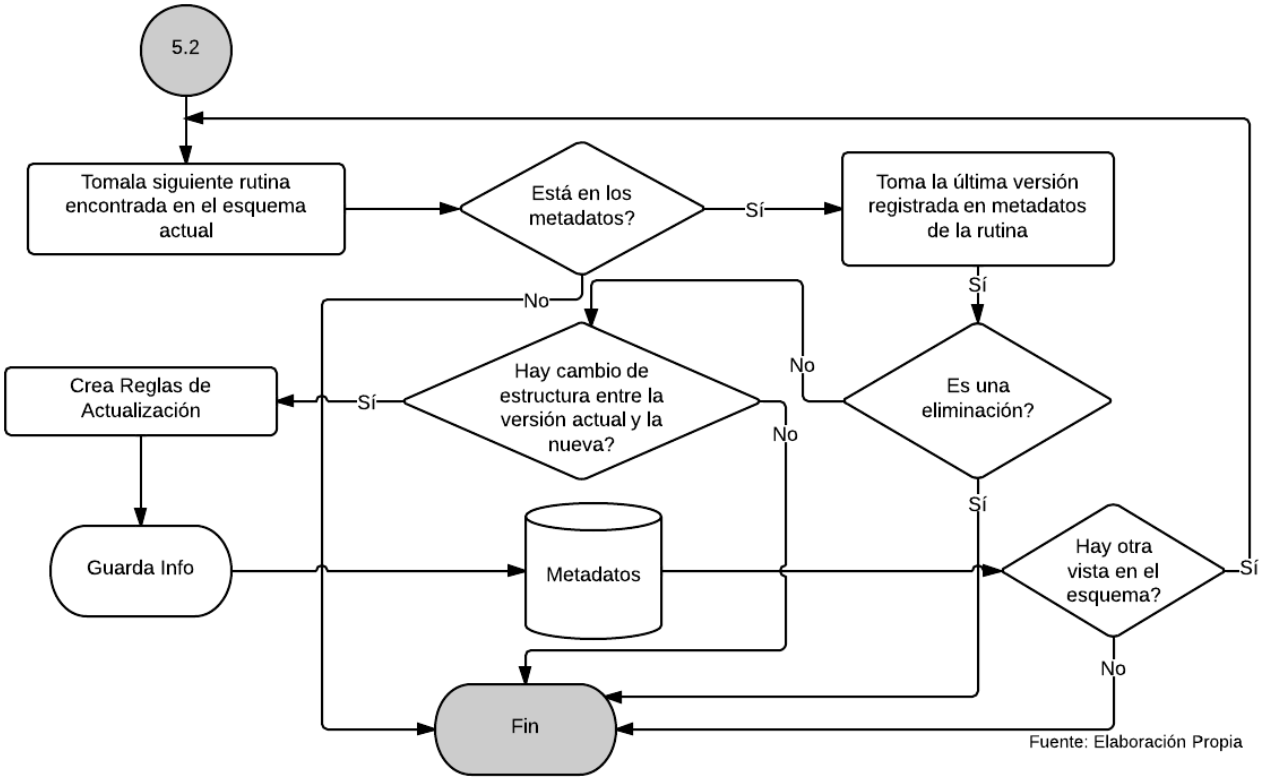


Figura 4.29: Diagrama de flujo para versionar rutinas con cambios registrados

Figura 4.30: Tabla versión - Ubicada en la base de datos en producción

Capítulo 5

Pruebas y resultados

Las pruebas se realizaron en un ambiente controlado sobre el sistema operativo *Microsoft Windows 7*, con un servidor de SQL Server 2008 R2, y el objetivo de los mismos fue comprobar la funcionalidad de la metodología de versionamiento propuesta, así como ejemplificar de forma específica que es lo que ocurre detrás de una actualización de un sistema de software para mantener la base de datos en una versión compatible con el sistema, de acuerdo a esta metodología.

La conexión se realizó de forma directa en el manejador y las bases de datos se montaron sobre el mismo servidor de SQL, la arquitectura de escenarios de prueba es descrita en la sección 5.1

5.1. Arquitectura de escenario de pruebas

Tal como se mencionó anteriormente, el ambiente de trabajo para pruebas controlado se estableció sobre un mismo servidor de SQL Server 2008 R2, en donde se montaron las 3 bases de datos principales a las que llamamos:

Metadatos 1.4

Compuesta por todas las tablas necesarias para registrar una versión, a fin de almacenar toda la información de la estructura actual de la base de datos versionada. La base de datos Metadatos 1.4 ilustrada en la Figura 5.1 corresponde a los metadatos mencionados en el capítulo 4 en donde se describieron los artefactos necesarios para la implementación de la metodología.

DB_DESARROLLO

Esta base de datos simula la base de datos del sistema de software que se encuentra en desarrollo, la cual con el paso del tiempo va cambiando a la par de los requerimientos del software.

DB_PRODUCCION

Esta base de datos representa aquella en dominio de los usuarios del software que ya ha sido liberado, la cual a diferencia de la base de datos de desarrollo sufre cambios en lapsos de tiempo más extendidos y rara vez se encuentra bajo la misma versión que la de desarrollo.

La distribución de dichas bases de datos para el ambiente de pruebas se estableció en un mismo servidor y es ilustrado en la Figura 5.1 y sobre ésta se realizaron las pruebas que son descritas más adelante.

Se diseñaron una serie de escenarios para ejecutar las pruebas de funcionalidad a la metodología desarrollada, y esas a su vez se dividieron en 2 tipos de escenarios; El primero de ellos corresponde a versionar una base de datos que está todavía en desarrollo y comprende entre las pruebas 1 a la 8.

Las últimas 2 pruebas (9 y 10) corresponden a aplicar el método de versionamiento en una de las bases de datos de los productos utilizados como objeto de estudio, con la finalidad

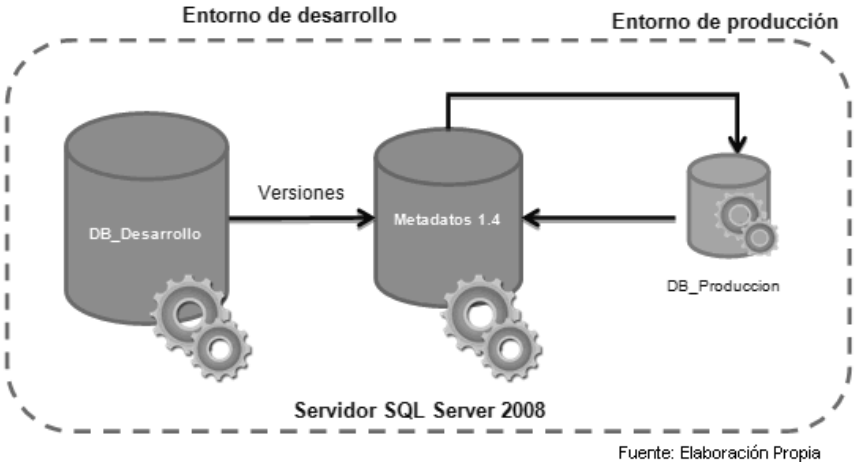


Figura 5.1: Arquitectura del entorno de pruebas

de comprobar si la metodología propuesta podrá ser utilizada en sistemas ya desarrollados.

A continuación se explica el objetivo, el procedimiento y los resultados obtenidos en cada una de las pruebas realizadas.

5.2. Pruebas sobre una base de datos nueva

Se hace referencia a pruebas sobre una base de datos nueva, cuando desde la primera versión de la base de datos se comienza a utilizar la metodología de versionamiento propuesta en este trabajo, esto significa que se llevó el control del inicio al final del ciclo de desarrollo.

A continuación se explican las pruebas realizadas.

5.2.1. Prueba 01: Preparación del entorno de pruebas

Durante esta prueba, se ejecutaron los scripts para crear nuestro entorno de desarrollo, el prototipo de producción y la base de datos para los metadatos.

Los script se encuentran previamente preparados y anexados en la documentación del

trabajo.

Procedimiento

1. Se crea una base de datos nueva llamada *Metadatos1.4* (Véase Script 5.1).

Script 5.1: Crear base de datos Metadatos 1.4

```
CREATE DATABASE [Metadatos1.4]
```

2. Se ejecuta el Script B.1 sobre la nueva base de datos el cual se encarga de crear las tablas de *Metadatos1.4*.

3. Se crea una base de datos vacía llamada **DB_DESARROLLO.** (Véase Script 5.2)

Script 5.2: Crear base de datos DB_DESARROLLO

```
CREATE DATABASE [DB_DESARROLLO]
```

4. Se ejecuta el Script B.2 (*ScriptsVersionamiento.sql*).

5. Se crea una base de datos vacía llamada: **DB_PRODUCCION** (Véase Script 5.3).

Script 5.3: Crear base de datos DB_PRODUCCION

```
CREATE DATABASE [DB_PRODUCCION]
```

6. Se ejecuta el Script B.3 (*ScriptDeployment.sql*) sobre la base de datos *DB_PRODUCCION* el cual se encarga de crear todos los operadores *SMO* bajo el esquema *'ver'*.

7. Insertar en **Metadatos1.4** el nombre de la base de datos que se va a versionar (Véase Script 5.4), en este caso '*DB_DESARROLLO*'.

Script 5.4: Registrar DB en Metadatos

```
INSERT INTO [Metadatos1.4].[dbo].[DB](NombreDB)
        VALUES('DB_DESARROLLO')
```

Resultados

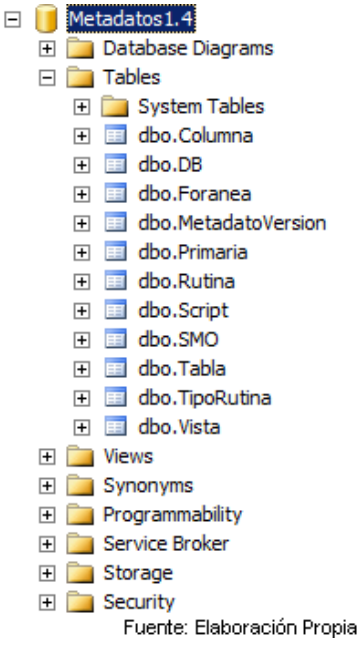
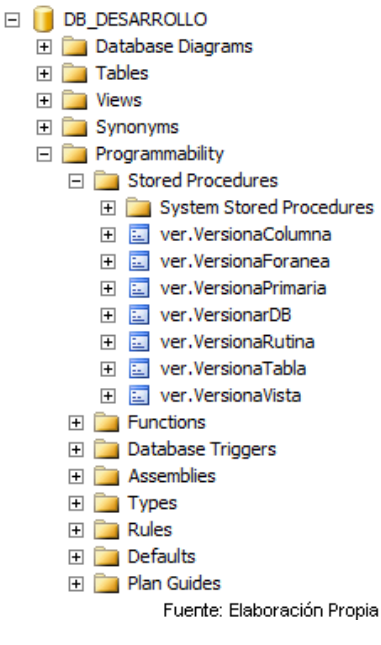
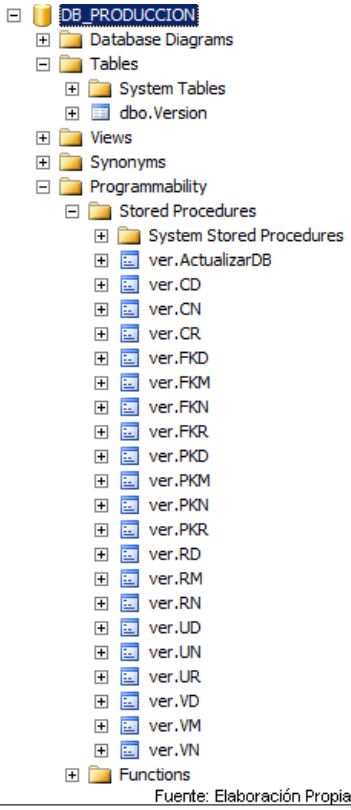
En la Tabla 5.1 se observa los resultados en las tres bases de datos que fueron afectadas y/o creadas durante la ejecución de esta prueba.

Como se puede obesrvar en la primera imagen, la base de datos Metadatos 1.4 contiene una serie de tablas que contendrán la descripción de la base de datos que es versionada. Ej. *DB, Tabla, Columna,* etc., creadas durante la ejecución del Script B.1 (*ScriptsCreacionTablasMetadatos1.4.sql*).

En la base de datos DB_DESARROLLO, primeramente se creó un esquema de base de datos denomiando '*ver*' y sobre el dominio de dicho esquema se crearon los procedimientos almacenados encargados de analizar dos bases de datos para determinar las diferencias y registrarlas en la base de datos Metadatos 1.4, que puede observarse en la segunda imagen de la Tabla 5.1.

Y por último se obtuvo una base de datos DB_PRODUCCION con un esquema de base de datos 'ver' y bajo su dominio los procedimientos almacenados '*SMO*' encargados de interpretar las reglas de actualización. Y una tabla que almacena la información de la versión en la que se encuentra en la tercera imagen de la Tabla 5.1.

Tabla 5.1: Tabla de resultados prueba 01: Preparación del entorno de pruebas

Metadatos1.4	DB_DESARROLLO	DB_PRODUCCION
 Fuente: Elaboración Propia	 Fuente: Elaboración Propia	 Fuente: Elaboración Propia

5.2.2. Prueba 02: Creación de tablas

El objetivo de esta prueba es crear un par de tablas en la base de datos de desarrollo, para después registrar una primera versión de base de datos en los Metadatos.

Procedimiento

Para ejecutar la prueba sobre la creación de tablas se deben seguir los siguientes pasos:

1. Se ejecuta en **DB_DESARROLLO**, el Script 5.5 el cual crea dos tablas, *Cliente* y *Producto* con sus respectivas columnas y llaves.

Script 5.5: Registrar tablas en desarrollo

```
USE DB_DESARROLLO
CREATE TABLE Cliente(
    ClienteID           int             IDENTITY(1,1),
    Nombre              varchar(50)     NULL,
    PrimerApellido      varchar(70)     NULL,
    SegundoApellido     varchar(70)     NULL,
    RFC                 varchar(15)     NULL,
    Telefono            varchar(20)     NULL,
    CONSTRAINT PK2 PRIMARY KEY NONCLUSTERED (ClienteID)
)
go

CREATE TABLE Producto(
    ProductoID      int             IDENTITY(1,1),
    CodigoBarras    varchar(50)     NULL,
```

```
    Descripcion     varchar(250)    NULL,
    Precio          float           NULL,
    CONSTRAINT PK1 PRIMARY KEY NONCLUSTERED (ProductoID)
)
go
```

2. Se ejecuta el procedimiento almacenado *VersionarDB* enviando como parámetro el nombre de la base de datos a versionar ('BD_DESARROLLO'), tal como se indica en el Script 5.6.

Script 5.6: 02 02 VersionarDB

```
USE DB_DESARROLLO
EXEC VER.VersionarDB 'DB_DESARROLLO'
```

3. Para verificar que se haya registrado nuestro cambio, realizamos una consulta *SELECT* sobre la tabla *"Tabla"* dentro de la base de datos **Metadatos1.4.**. (Véase el Script 5.7)

Script 5.7: 02 Revisar Metadatos

```
USE [Metadatos1.4]


SELECT vTablaID, [object_id], SmoID, NombreTabla
FROM [Metadatos1.4].dbo.Tabla
```

Resultados

En la Tabla 5.2 se observa el resultado de la consulta sobre la tabla *Tabla* de la base de datos de Metadatos, la cual registró la creación de las dos nuevas tablas: *Cliente* y *Producto*.

Tabla 5.2: Resultado prueba 02: Creación de tablas - Resultados al registrar una tabla

02_01_RegistroDeTabla

Results | Messages

	vTablaID	object_id	SmoID	NombreTabla
1	2	373576369	1	Cliente
2	3	405576483	1	Producto

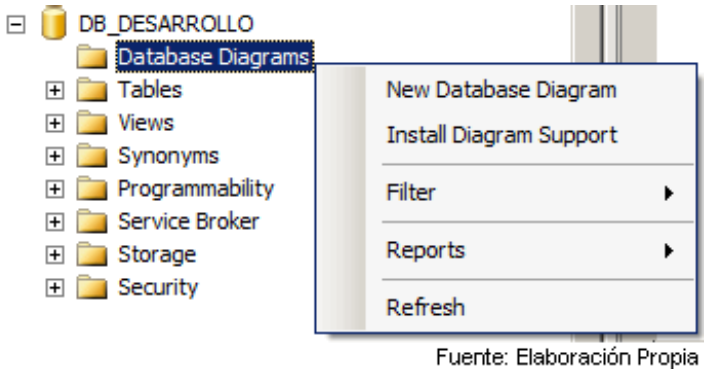
Fuente: Elaboración Propia

5.2.3. Prueba 03: Verificar Columnas

El objetivo de la prueba consiste verificar el registro de las columnas que fueron creadas durante la prueba *02 Creación de tablas*, esto con la finalidad de comprobar que el procedimiento registra correctamente dichos movimientos.

Procedimiento

1. Utilizando SQL Managment, obtenemos el diagrama de la base de datos *DB_DESARROLLO* en su estado actual. (Véase Figura 5.2).



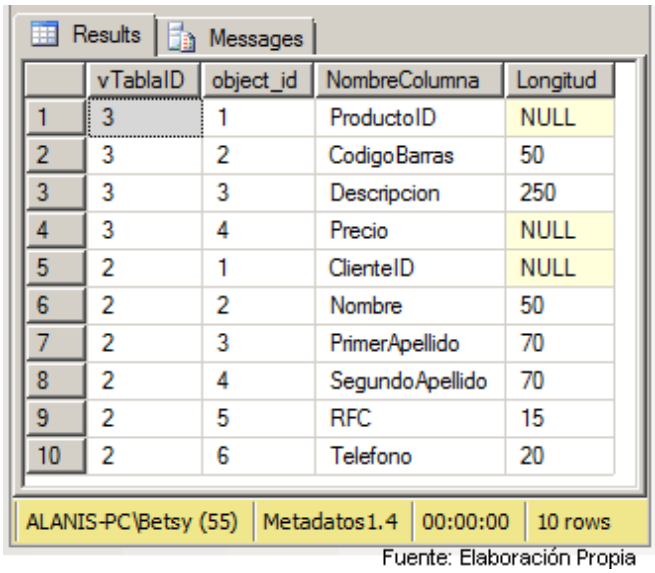
Fuente: Elaboración Propia

Figura 5.2: Obtener diagrama en MSSQL

2. Dentro de la base de datos *Metadatos1.4* ejecutamos la sentencia para verificar el registro de columnas. (Véase Script 5.8).

Script 5.8: 01 Versionar DB

Tabla 5.3: Tabla de resultados prueba 03: Verificar Columnas

03_01_TablasClienteProducto	03_02_RegistroDeColumnas
 Fuente: Elaboración Propia	 Fuente: Elaboración Propia

vTablaID	object_id	NombreColumna	Longitud
3	1	ProductoID	NULL
3	2	CodigoBarras	50
3	3	Descripcion	250
3	4	Precio	NULL
2	1	ClienteID	NULL
2	2	Nombre	50
2	3	PrimerApellido	70
2	4	SegundoApellido	70
2	5	RFC	15
2	6	Telefono	20

```
USE [Metadatos1.4]
SELECT vTablaID, [object_id], NombreColumna, Longitud
        FROM [Metadatos1.4].dbo.Columna
```

Véase resultado 03_02_RegistroDeColumnas en la tabla 5.3.

Resultados

Como resultado del Paso 1 de esta prueba se encuentra la Tabla 5.3 se ilustraron los resultados obtenidos durante la prueba, la primera imagen (03_01_TablasClienteProducto) corresponde a las dos tablas que fueron creadas mientras que la segunda imagen (03_02_RegistroDeColumnas) ilustra el resultado de ejecutar un *SELECT* a la tabla de *Columna* en la base de datos *Metadatos1.4*

5.2.4. Prueba 04: Manipular Columnas

El objetivo de la prueba consiste en comprobar el registro de los cambios en las columnas al alterar una tabla.

Por lo que durante el procedimiento de esta prueba, se realizaron cambios a las columnas previamente creadas.

Se para fines de ejemplo, se asumió que a petición del cliente el sistema requiere ahora llevar el control del mínimo y máximo que puede un producto venderse y que el Precio que tenemos actualmente almacenado corresponde al precio de compra.

Procedimiento

1. Abrir una ventana de *query* dentro de la base de datos **DB_DESARROLLO**.
2. Ejecutar el Script 5.9 (*04 01 ModificarTablaProducto*) para crear nuestras 2 nuevas columnas *MinPrecioVenta* y *MaxPrecioVenta* además modificar el nombre de la columna *Precio* y reemplazarlo por *PrecioCompra*dentro de la tabla *Producto*.

Script 5.9: 04 01 ModificarTablaProducto

```
USE DB_DESARROLLO


--Crear 2 nuevas columnas en la tabla Producto
ALTER TABLE Producto
 ADD MinPrecioVenta    float           NULL,
         MaxPrecioVenta    float           NULL


--Script para renombrar una columna en SQL Server:
EXEC sp_RENAME 'Producto.Precio' , 'PrecioCompra', 'COLUMN'
```

3. Ejecutamos el script para versionar bases de datos dentro de la base de datos **DB_DESARROLLO** (Script 5.10 - *VersionarDB*).

Script 5.10: 04 02 VersionarDB

```
USE [DB_DESARROLLO]


EXEC [ver].VersionarDB 'DB_DESARROLLO'
```

Con la finalidad de verificar los cambios en la base de datos:

4. Utilizando *SQL Managment Studio*, obtenemos el diagrama de la base de datos de la tabla *Producto* de la base de datos **DB_DESARROLLO**.

5. Verificar en la base de datos **Metadatos1.4** el contenido de la tabla *Columna*, en esta ocasión cruzaremos con la tabla *SMO* para observar el cambio que ha sido registrado, y la versión, ejecutando el Script 5.11 (*04 03 Revisión Metadatos Columnas*).

Script 5.11: 04 03 Revisión Metadatos Columnas

```
USE [Metadatos1.4]


SELECT vTablaID, [object_id], NombreColumna, Longitud, c.
   SmoID,
        sm.Descripcion as SMO_Descripcion, c.VersionID
        FROM [Metadatos1.4].dbo.Columna c
        INNER JOIN [Metadatos1.4].dbo.SMO sm on sm.SmoID =
           c.SmoID
        where vTablaID = 3
```

Véase el resultado en la imagen*IMG_04_05_ResultadoVersionarColumna* que se encuentra en la Tabla 5.4.

04_01_DiagramaProducto

Fuente: Elaboración Propia

04_05_ResultadoVersionarColumna

Results | Messages

	vTablaID	object_id	NombreColumna	Longitud	SmoID	SMO_Descripcion	VersionID
1	3	5	MinPrecioVenta	NULL	3	Creacion de columna	3
2	3	6	MaxPrecioVenta	NULL	3	Creacion de columna	3
3	3	4	PrecioCompra	NULL	4	Cambio nombre Columna	3
4	3	1	ProductoID	NULL	3	Creacion de columna	2
5	3	2	CodigoBarras	50	3	Creacion de columna	2
6	3	3	Descripcion	250	3	Creacion de columna	2
7	3	4	Precio	NULL	3	Creacion de columna	2

Fuente: Elaboración Propia

Tabla 5.4: Tabla de resultados prueba 04: Manipular Columnas

Resultados

Los resultados obtenidos durante esta prueba fueron la creación de dos columnas y la modificacion del nombre de una de ellas, que puede obesrvarse en la primera imagen 04_01_DiagramaProducto de la Tabla 5.4.

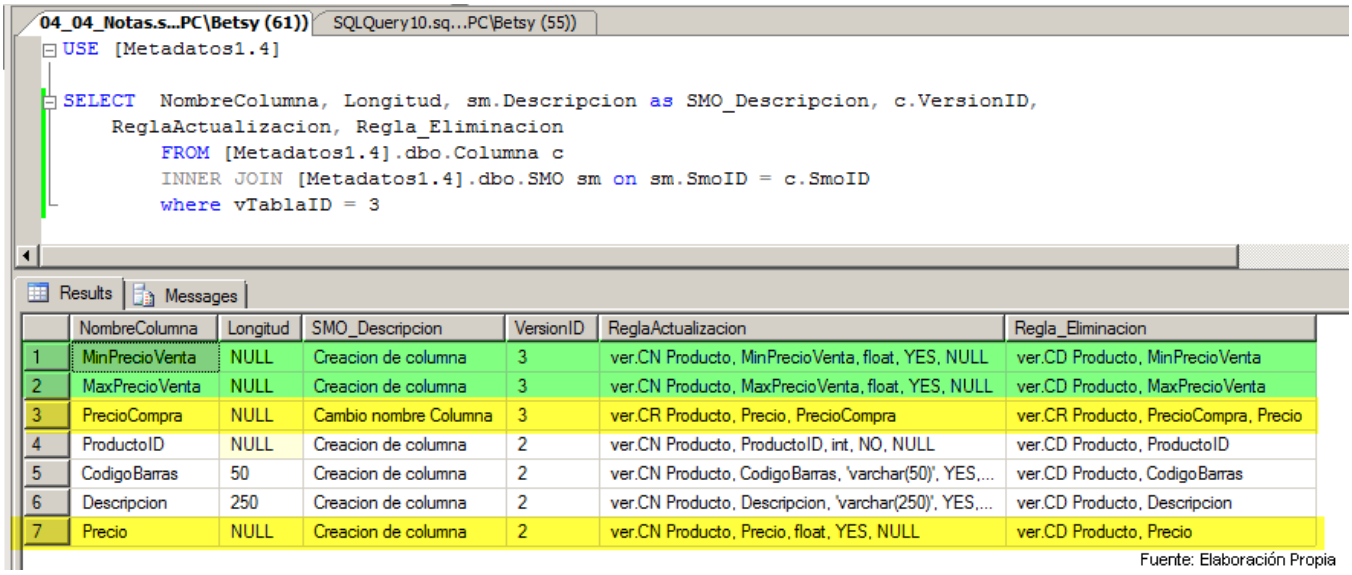
Mientras que el resultado del Script para verificar la inforamción registrada en la tabla *Columna* de la base de datos Metadatos 1.4 (Paso 5) se encuentra contenida en la imagen 04_05_ResultadoVersionarColumna.

Anexo

Tal como se puede observar en la imagen *04 02 ResultadoVersionarColumna* contenida en la Tabla 5.4, el script para versionar bases de datos ha encontrado un cambio en el nombre de la columna Precio de la tabla Producto y ahora la tabla Columna de los Metadatos contiene la definición de ambas columnas en diferentes versiones. De esta forma se asegura el cambio de nombre al migrar de una versión a otra.

En la Figura 5.3 se puede observar el resultado de la consulta, la cual contiene las reglas de actualización que para este caso se refieren a al creación de columnas y al cambio de

nombre.



```
USE [Metadatos1.4]

SELECT  NombreColumna, Longitud, sm.Descripcion as SMO_Descripcion, c.VersionID,
    ReglaActualizacion, Regla_Eliminacion
        FROM [Metadatos1.4].dbo.Columna c
        INNER JOIN [Metadatos1.4].dbo.SMO sm on sm.SmoID = c.SmoID
        where vTablaID = 3
```

Results | Messages

	NombreColumna	Longitud	SMO_Descripcion	VersionID	ReglaActualizacion	Regla_Eliminacion
1	MinPrecioVenta	NULL	Creacion de columna	3	ver.CN Producto, MinPrecioVenta, float, YES, NULL	ver.CD Producto, MinPrecioVenta
2	MaxPrecioVenta	NULL	Creacion de columna	3	ver.CN Producto, MaxPrecioVenta, float, YES, NULL	ver.CD Producto, MaxPrecioVenta
3	PrecioCompra	NULL	Cambio nombre Columna	3	ver.CR Producto, Precio, PrecioCompra	ver.CR Producto, PrecioCompra, Precio
4	ProductoID	NULL	Creacion de columna	2	ver.CN Producto, ProductoID, int, NO, NULL	ver.CD Producto, ProductoID
5	CodigoBarras	50	Creacion de columna	2	ver.CN Producto, CodigoBarras, 'varchar(50)', YES,...	ver.CD Producto, CodigoBarras
6	Descripcion	250	Creacion de columna	2	ver.CN Producto, Descripcion, 'varchar(250)', YES,...	ver.CD Producto, Descripcion
7	Precio	NULL	Creacion de columna	2	ver.CN Producto, Precio, float, YES, NULL	ver.CD Producto, Precio

Fuente: Elaboración Propia

Figura 5.3: Reglas de actualización de columnas

5.2.5. Prueba 05: Crear Vistas

El objetivo de esta prueba será crear una vista y versionar la base de datos, con la finalidad de corroborar que la metodología de *SMO* registra correctamente dicho movimiento.

Por lo que en nuestro ejemplo creamos una vista llamada *Listado_PVP* que nos ayude a calcular el precio del producto con una ganancia del 30 %, utilizando el siguiente criterio:

- Si el precio + la ganancia es menor al precio mínimo, entonces mostrar el precio mínimo de venta.
- Si el precio + la ganancia excede al precio máximo, entonces mostrar el precio máximo de venta.
- De lo contrario, mostrar el Precio de Compra + el 30 %.

Procedimiento

1. **DB_DESARROLLO**: Para efectos de tener algunos datos, vamos a insertar un par de productos en la tabla Producto.

Script 5.12: 05 01 Insertar datos

```
INSERT INTO [DB_DESARROLLO].[dbo].[Producto]([CodigoBarras
  ],
       [Descripcion],[PrecioCompra],[MinPrecioVenta],[
          MaxPrecioVenta])
    VALUES
    ('1234567890123','Leche 1 litro',5,6.5,10),
    ('1234567890124','Coca Cola',10,10,12)
```

2. **DB_DESARROLLO**: Se ejecuta el Script 5.13 (*05 02 Crear la vista Listado_PVP*), el cual se encarga de crear la vista de acuerdo a las características antes mencionadas.

Script 5.13: 05 02 Crear la vista LISTADO_PVP

```
USE DB_DESARROLLO
GO


CREATE VIEW LISTADO_PVP AS
(
SELECT ProductoID, CodigoBarras, Descripcion, PrecioCompra,
    MinPrecioVenta, MaxPrecioVenta,
CASE
WHEN PrecioCompra*1.30>MaxPrecioVenta then MaxPrecioVenta
WHEN PrecioCompra*1.30<MinPrecioVenta then MinPrecioVenta
```

```
ELSE PrecioCompra*1.30
END as PrecioVenta,
((CASE
WHEN PrecioCompra*1.30>MaxPrecioVenta then MaxPrecioVenta
WHEN PrecioCompra*1.30<MinPrecioVenta then MinPrecioVenta
ELSE PrecioCompra*1.30
END) /PrecioCompra-1) as MargenGanancia
from Producto
)
GO
```

3. **DB_DESARROLLO**: Ejecutar el procedimiento almacenado para versionar. (Véase Script 5.14 (*05 03 Versionar DB*)

Script 5.14: 05 03 Versionar DB

```
USE DB_DESARROLLO


EXEC ver.VersionarDB 'DB_DESARROLLO'
```

4. **Metadatos1.4**: Para verificar que se la vista haya sido identificada y registrada, ejecutamos la siguiente sentencia (Script 5.15) en nuestra base de datos de metadatos:

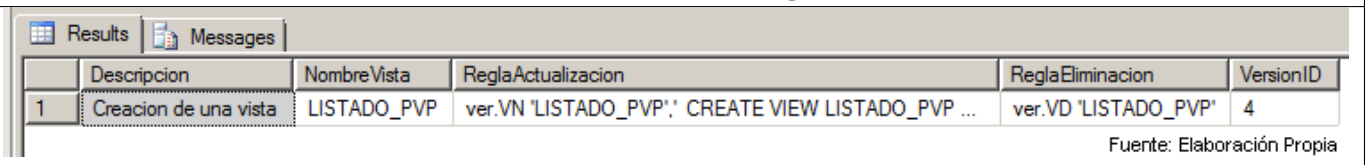
Script 5.15: 05 04 Verificar Registro Vista

```
USE [Metadatos1.4]


SELECT sm.Descripcion, v.NombreVista, v.ReglaActualizacion,
        v.ReglaEliminacion, v.VersionID
```

Tabla 5.5: Tabla de resultados prueba 05: Crear vistas

IMG_05_05_ResultadoRegistroVista



Results | Messages

	Descripcion	NombreVista	ReglaActualizacion	ReglaEliminacion	VersionID
1	Creacion de una vista	LISTADO_PVP	ver.VN 'LISTADO_PVP',' CREATE VIEW LISTADO_PVP ...	ver.VD 'LISTADO_PVP'	4

Fuente: Elaboración Propia

```
        FROM Vista v
                INNER JOIN SMO sm on v.SmoID = sm.SmoID
```

Resultados

El resultado obtenido durante esta prueba es ilustrado en la tabla 5.5 en donde se puede observar que la vista y sus reglas de actualización han sido registradas en los Metadatos.

5.2.6. Prueba 06: Crear Funciones

El objetivo de esta prueba es crear y registrar en una nueva versión de la base de datos con una función definida por el usuario. Para nuestro ejemplo, esta función tendrá como objetivo futuro simplificar la vista que hemos creado en la prueba *05 Crear Vistas*.

Procedimiento

1. **DB_DESARROLLO**: Crear una función que nos permita calcular el precio de un producto dado. Ejecutar el Script B.4 (*06_01_CrearFuncionsql*).

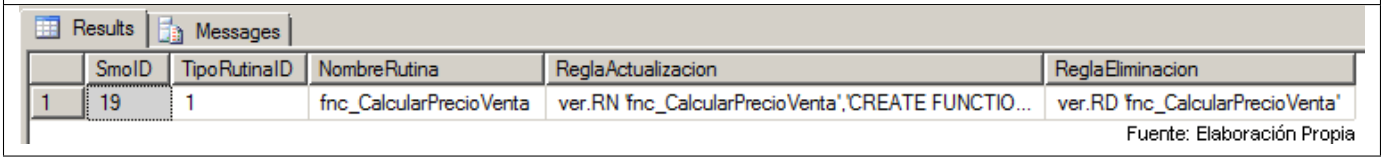
2. **DB_DESARROLLO**: Ejecutar VersionarDB

Script 5.16: 06 03 Revisión Metadatos Columnas

```
USE DB_DESARROLLO
```

Tabla 5.6: Tabla de resultados prueba 06: Crear Funciones

06_01_RegistroFuncion_Procedimiento



Results | Messages

	SmoID	TipoRutinaID	NombreRutina	ReglaActualizacion	ReglaEliminacion
1	19	1	fnc_CalcularPrecioVenta	ver.RN 'fnc_CalcularPrecioVenta','CREATE FUNCTIO...	ver.RD 'fnc_CalcularPrecioVenta'

Fuente: Elaboración Propia

```
EXEC ver.VersionarDB 'DB_DESARROLLO'
```

3. **Metadatos1.4**: Para verificar lo que está en metadatos registrado en la versión 5, Ejecutar el siguiente Script 5.17 (*06 03 Verificar Función*).

Script 5.17: 06 03 Verificar Función

```
USE [Metadatos1.4]


SELECT SmoID, TipoRutinaID, NombreRutina,
   ReglaActualizacion, ReglaEliminacion
FROM Rutina
WHERE VersionID = 5
```

Resultados

En la tabla 5.6 se ilustra el resultadod el Script 5.17 la cual de muestra el registro correcto de la nueva función en la base de datos y sus reglas de actualización.

5.2.7. Prueba 07: Crear Procedimientos Almacenados

El objetivo de esta prueba consiste en crear un procedimiento almacenado ejemplo en la base de datos de desarrollo y registrarlo a través del procedimiento para versionar.

Una vez realizado esto, se pretende verificar el correcto registro de dicha rutina en los metadatos.

Procedimiento

1. **DB_DESARROLLO**: Crear un procedimiento almacenado para insertar registros en la tabla Clientes. (Veáse Script 5.18 - *Script para los escenarios de pruebas; Crear un procedimiento*).

Script 5.18: Script para los escenarios de pruebas; Crear un procedimiento

```
USE [DB_DESARROLLO]


SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO


CREATE PROCEDURE ClientesInsProc
(
        @Nombre VARCHAR(50),
        @PrimerApellido varchar(70),
        @SegundoApellido varchar(70) = NULL,
        @RFC varchar(15) =NULL,
```

```
        @Telefono varchar(20) = NULL
)
AS
BEGIN
        INSERT INTO Cliente (Nombre, PrimerApellido,
           SegundoApellido, RFC, Telefono)
        VALUES ( @Nombre, @PrimerApellido, @SegundoApellido
           , @RFC, @Telefono)
END
GO
```

2. **DB_DESARROLLO**: Ejecutar el Script 5.19 (*07 01 VersionarDB*).

Script 5.19: 07 01 Versionar DB

```
USE DB_DESARROLLO


EXEC ver.VersionarDB 'DB_DESARROLLO'
```

3. **Metadatos1.4**: Verificar el contenido de la versión creada. Ejecutar el Script 5.20 (*07 03 Versionar procedimiento*).

Script 5.20: 07 03 Versionar procedimiento

```
USE [Metadatos1.4]


SELECT TipoRutinaID, NombreRutina, ReglaActualizacion,
   ReglaEliminacion
FROM Rutina
```

Tabla 5.7: Resultado del registro de procedimiento prueba 07: Crear Procedimientos

IMG_07_01_ResultadoRegistroProcedimiento

Results | Messages

	TipoRutinaID	NombreRutina	ReglaActualizacion	ReglaEliminacion
1	1	fnc_CalcularPrecioVenta	ver.RN 'fnc_CalcularPrecioVenta','CREATE FUNCTIO...	ver.RD 'fnc_CalcularPrecioVenta'
2	2	ClientesInsProc	ver.RN 'ClientesInsProc',' CREATE PROCEDURE Cli...	ver.RD 'ClientesInsProc'

Fuente: Elaboración Propia

Resultados

Como resultado de esta prueba se obtuvo el registro del nuevo procedimiento almacenado dentro de la tabla *Rutina* en la base de datos de metadatos, lugar en donde se encuentra también el registro de la función de la prueba anterior. (Véase Tabla 5.7).

5.2.8. Prueba 08: Migrar base de datos vacía a una versión

Esta prueba tiene como objetivo tomar una base de datos vacía y migrarla a una versión previamente registrada durante las pruebas anteriores, de tal forma que se pueda comprobar la versatilidad de actualizar una base de datos a cualquiera de las versiones.

Indicaciones Especiales

Para comprobar la igualdad de las bases de datos, se utilizó *SQL Compare de RedGate*, herramienta que es actualmente utilizada por la empresa para comparar las bases de datos.

Procedimiento

1. Cree una base de datos completamente vacía con el nombre **DB_PRODUCCION_NUEVA**. (Véase Script 5.21).

Script 5.21: Crear base de datos DB_PRODUCCION_NUEVA

```
CREATE DATABASE [DB_PRODUCCION_NUEVA]
```

2. Ejecute el script *ScriptBaseDesarrollo.sql* que contiene los SMO bajo el esquema *ver* que serán contenidos en la base de datos de desarrollo a fin de darle la capacidad de actualizarse.

3. **SQL Compare** Comparamos la base de datos **DB_DESARROLLO** y **DB_PRODUCCION_PRO** Véase el resultado *IMG_08_01_Resultado_Comparacion1RedGate*

4. **DB_PRODUCCION**. Ejecutar el procedimiento almacenado *ActualizarDB* enviando como primer parámetro la versión a la que queremos migrar y un 1 para indicar que deseamos que se ejecute. (Véase Script 5.22).

Script 5.22: 08 01 ActualizarDB

```
use [DB_PRODUCCION_NUEVA]


EXEC ver.ActualizarDB 3,1
```

5. Correr nuevamente *SQL Compare* y observamos los resultados en la ilustración *IMG_08_02_Resultado_* en la cual podemos observar que las diferencias bajo el esquema *dbo* sean reducido a 3.

6. Por último, migrar la base de datos **DB_PRODUCCION_NUEVA** a la última versión registrada en **DB_DESARROLLO** ejecutando la instrucción del Script 5.23.

Script 5.23: 08 02 ActualizarDB

```
use [DB_PRODUCCION_NUEVA]


EXEC ver.ActualizarDB 6, 1
```

7. Ejecutar *SQL Compare* y observar los resultados de la ilustración *IMG_08_05_Resultado_Comparacion2* de la tabla 5.8, en la cual se puede observar que las diferencias bajo el esquema *dbo* han desaparecido, lo cual nos indica que ambas bases de datos (exceptuando los SMO bajo el esquema *ver*), son iguales.

Resultados

Tal como se ilustra en las tres imagenes de la Tabla 5.8 las diferencias entre *DB_DESARROLLO* y *DB_PRODUCCION_NUEVA* fueron disminuyendo con forme *DB_DESARROLLO_NUEVA* fue migrada a la última versión de *DB_DESARROLLO*.

5.3. Pruebas sobre una base de datos existente

Pruebas sobre una base de datos existente se refiere a tomar una base de datos que se encuentra en uso por algún producto de software y comenzar a registrar la primer versión a partir del momento en que se dispone de esta metodología.

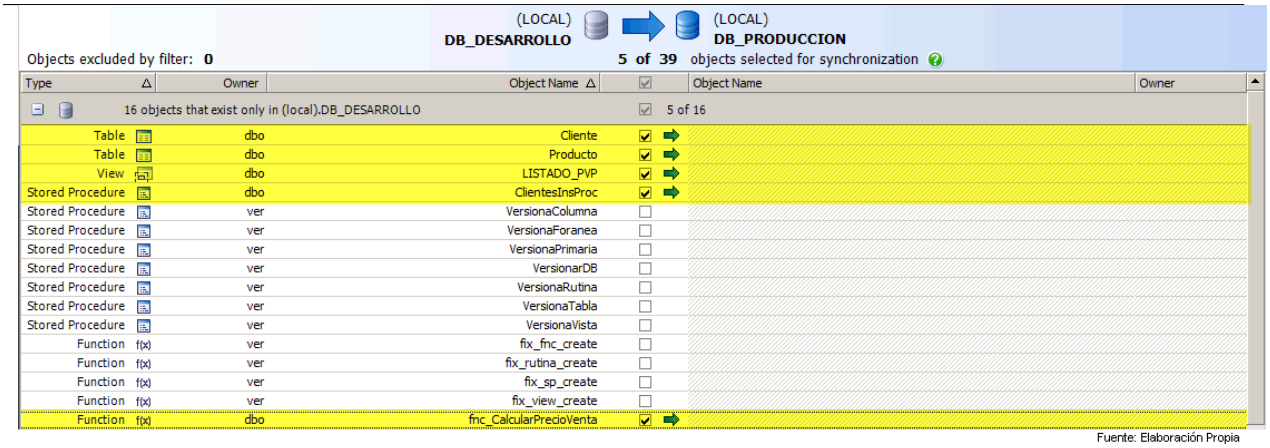
5.3.1. Prueba 09: Versionar una base de datos de un sistema dado

La prueba consiste en tomar una base de datos de un sistema actual de la empresa y versionarlo, de esta forma los metadatos contienen una base de datos que ya es utilizada en producción, y ésta sirvió posteriormente para utilizar el esquema registrado en para migrar una base de datos vacía.

Durante nuestras pruebas se seleccionó la base de datos llamada *FacturacionElectronica* por lo que aparece en los siguientes Scripts.

Tabla 5.8: Tabla de resultados prueba 08: Migrar base de datos vacía a una nueva versión

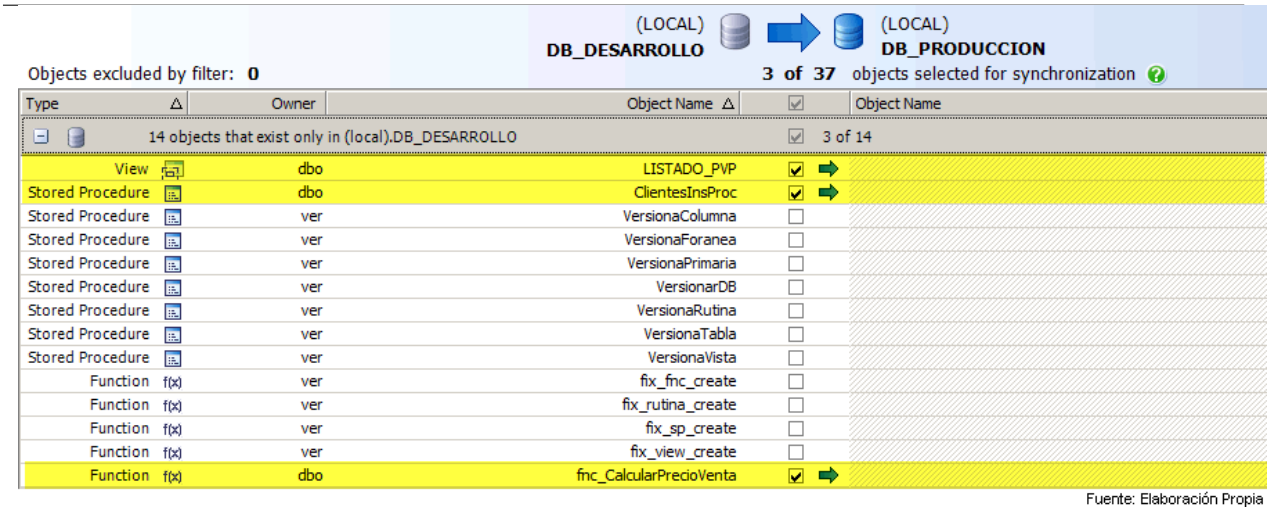
IMG_08_01_Resultado_Comparacion1RedGate



Fuente: Elaboración Propia

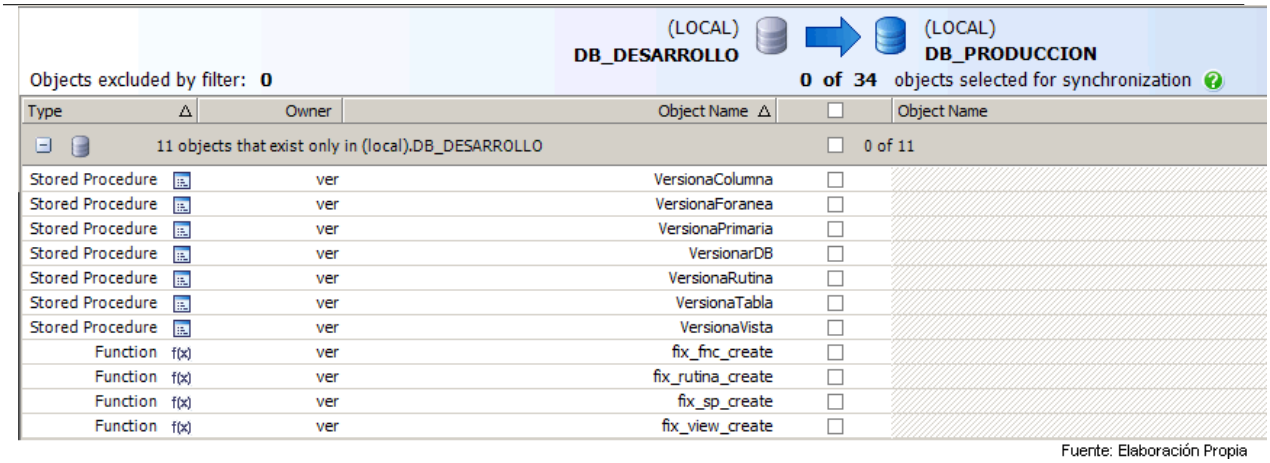
En esta ilustración se puede observar los elementos bajo el esquema *dbo* que se encuentran en **DB_DESARROLLO** y que no están en **DB_PRODUCCION**. Un total de 5 objetos.

textbfIMG_08_02_Resultado_Comparacion2RedGate



Fuente: Elaboración Propia

IMG_08_05_Resultado_Comparacion2RedGate



Fuente: Elaboración Propia

Indicaciones

Para que la base de datos nueva funcione, deberá contar con los procedimientos almacenados bajo el esquema *ver* y así poder registrar el esquema, por lo que lo primero que tenemos que hacer es incluir dichos procedimientos en la base de datos ejemplo.

Procedimiento

1. **Metadatos 1.4**: incluir la base de datos que se va a versionar en las tabla *DB* ejecutando el siguiente Script 5.24.

Script 5.24: 09 01 Agregar DB a Metadatos

```
use [Metadatos1.4]


INSERT INTO DB VALUES ('FacturacionElectronica')
```

2. Ejecutar el Script (*ScriptIncluirVersionador.sql*) en la base de datos que vamos a versionar (En nuestro ejemplo, DB: *FacturacionElectronica*).

3. **FacturacionElectronica**: ejecute *VersionarDB* tal como se observa en el Script B.2.

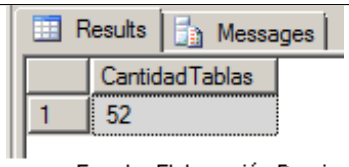
Script 5.25: 09 02 VersionarDB Facturación Electrónica

```
use [FacturacionElectronica]


EXEC ver.VersionarDB 'FacturacionElectronica'
```

4. **Metadatos1.4**: Para verificar las tablas registradas en nuestra segunda base de datos ejecute el script lst:ScriptPruebas0903Count sobre la base de datos *Metadatos1.4*.

Tabla 5.9: Resultado del registro de la base de datos 'FacturacionElectronica'

IMG_09_01_CantidadTablas

Fuente: Elaboración Propia

Script 5.26: 09 03 Contar tablas encontradas en Base de datos Facturación Electrónica

```
USE [Metadatos1.4]


SELECT COUNT(1) as CantidadTablas
FROM Tabla WHERE BaseDatosID = 2
```

Resultados

Tal como se obesrva en la Tabla 5.9 los metadasos registraron 52 tablas en la base de datos de *FacturacionElectronica*.

5.3.2. Prueba 10: Migrar una base de datos vacía a la última versión de un sistema dado

Para esta prueba se utilzó la base de datos versionada durante la prueba *09 Versionar una base de datos de un sistema dado*, para migrar una nueva base de datos vacía a la última versión de la base de datos resultante en dicha prueba.

A continuación se describe el procedimiento a seguir para ejecutar esta prueba:

Indicaciones

Durante esta aprueba, se hará uso de *SQL Compare de RedGate* para verificar el funcionamiento de los SMO creados.

Procedimiento

1. Cree una base de datos nueva

Script 5.27: 10 01 Crear DB Nueva

```
CREATE DATABASE [FE_Nueva]
GO
```

2. Ejecute el script *ScriptsDeploymenṫsql*.

3. Ejecute *SQL Compare* con las bases de datos: **FacturacionElectronica** y **FE_Nueva**. Vease resultado: *IMG_10_02_Resultado_Comparacion1RedGate*

4. **FE_Nueva**: Dado que, la base de datos de **FacturacionElectronica** fue versionada en la prueba *09 Versionar una base de datos de un sistema dado*, ejecute el procedimiento almaceando para migrar a la última versión de la siguiente manera:

Script 5.28: 10 02 ActualizarDB

```
EXEC ver.ActualizarDB 2,1
```

5. **RedGate**: Ejecuta nuevamente *SQL Compare* para observar las diferencias entre las bases de datos. Vease resultado: *IMG_10_04_ResultadoActualizarDB*.

Resultados

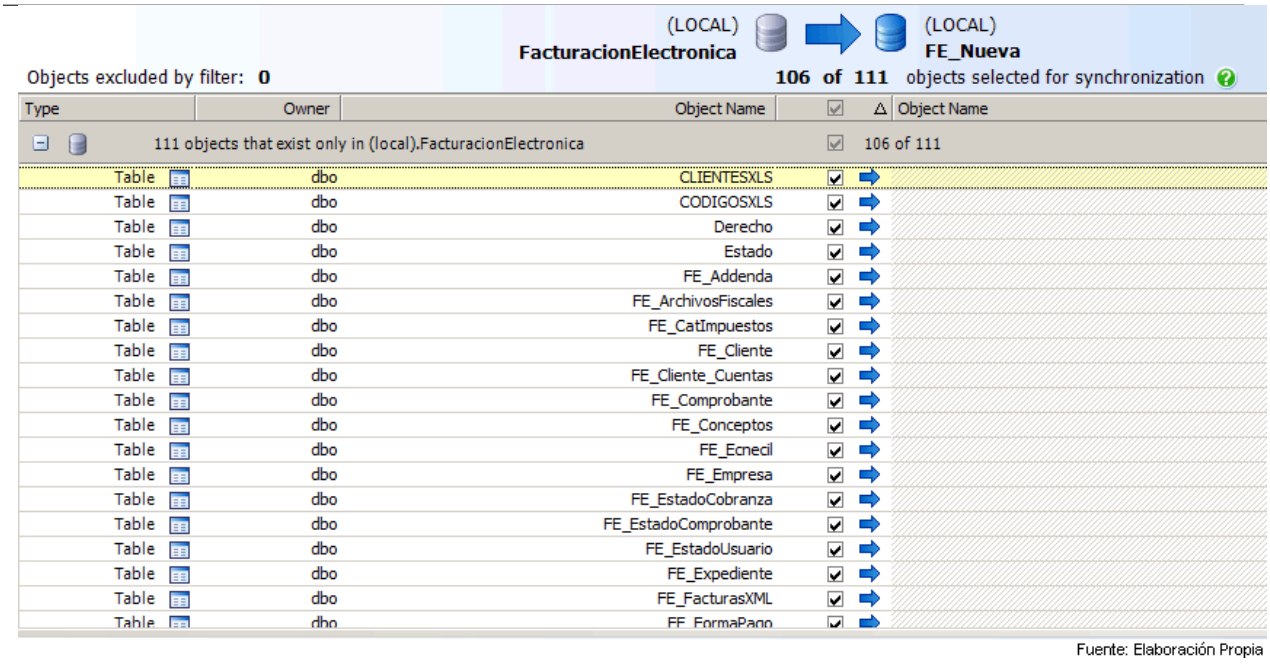
En la Tabla **??** se pueden observar los resultados de la prueba Migrar una base de datos vacía a la última versión de un sistema dado. Tal como se puede observar en la primera imagen, *SQL Compare* de *RedGate*, registra que existen diferencias entre la base de datos de producción y la base de datos nueva.

Una vez ejecutado el procedimiento almacenado que se encarga de realizar la actualización, y ejecutar nuevamente la comparación con la herramienta se puede observar el resultado de la segunda imagen de la Tabla **??**, la cual encuentra ahora 128 objetos idénticos entre las bases de datos, que corresponden a la estructura de producción y algunas tablas del versionador.

5.4. Análisis de resultados

En la Tabla 5.11 se enlistan los resultados finales de cada una de las pruebas realizadas a la metodología de versionamiento basada en SMO en Microsoft SQL Server 2008 R2, y la duración expresada en minutos.

IMG_10_02_Resultado_Comparacion1RedGate



Fuente: Elaboración Propia

RedGate indica que existen 106 objetos existentes en **FacturacionElectronica** bajo el esquema *dbo* que no están en **FE_Nueva**.

textbfIMG_10_04_ResultadoActualizarDB

Fuente: Elaboración Propia

Como podrá usted observar, *RedGate* encontró las diferencias en **FacturacionElectronica** que corresponden únicamente a los SMO bajo el esquema ver, y en **FE_Nueva** los procedimientos para versionar, mientras que las tablas que pertenecen propiamente al sistema de facturación se encuentran en la lista como objetos iguales.

Tabla 5.10: Tabla de resultados prueba 10: Migrar una base de datos vacía a la última versión

Tabla 5.11: Resultado de las pruebas realizadas

Nombre de la prueba	Duración	Resultado	Sección
Preparación del entorno de pruebas	02:00.74	Exitoso	5.2.1
Creación de tablas	00:57.60	Exitoso	5.2.2
Verificar Columnas	00:19.85	Exitoso	5.2.3
Manipular Columnas	01:07.49	Exitoso	5.2.4
Crear Vistas	01:32.33	Exitoso	5.2.5
Crear Funciones	00:53.61	Exitoso	5.2.6
Crear Procedimientos Almacenados	01:17.24	Exitoso	5.2.7
Migrar base de datos vacía a una versión	1:16.39	Exitoso	5.2.8
Versionar una base de datos de un sistema dado	2:07.29	Exitoso	5.3.1
Migrar una base de datos vacía a la última versión de un sistema dado	1:02.14	Exitoso	5.3.2

Capítulo 6

Conclusiones y trabajo futuro

A continuación se presentan algunas conclusiones a las que se llegó con la experiencia de la aplicación y el desarrollo de la metodología propuesta. También se hacen recomendaciones para su correcto funcionamiento y se exponen algunas oportunidades de mejora como trabajo futuro.

6.1. Conclusiones

Al término del desarrollo de esta metodología se crearon 7 procedimientos almacenados para registrar una versión, estos proveen la funcionalidad de versionar directamente la base de datos durante el proceso de desarrollo de los sistemas de software, lo cual permite que se realice dentro del mismo ciclo de vida del mismo, los cuales se pueden visualizar en la Tabla **??** del capítulo 3.

Mismos que son contenidos en la base de datos de desarrollo y que a su vez, crean una serie de reglas de actualización utilizando los 22 *SMO* que fueron creados para interpretar los cambios que se registran en una nueva versión de base de datos:

Tabla 6.1: SMO y tablas relacionadas.

Tabla 4.1	SMO para tablas
Tabla 4.2	SMO para columnas
Tabla 4.3	SMO para llaves primarias e índices
Tabla 4.4	SMO para llaves foráneas
Tabla 4.5	SMO para vistas
Tabla 4.6	SMO para rutinas

Estos procedimientos almacenados proveen la posibilidad de controlar las versiones de una base de datos y a su vez automatizar el proceso de migración de versiones.

Y por último se creó una estructura de base de datos capaz de contener la información generada por los operadores diseñados para registrar una versión y los operadores SMO, en la base de datos que se denominó *Metadatos1.4*.

Una vez concluido con el desarrollo se realizaron 10 pruebas cubriendo distintos escenarios posibles para migrar una base de datos de una versión a otra. Tanto bases de datos nuevas, como también bases de datos que ya se encuentran en producción usando como caso de estudio la empresa *SDPoint*.

Las pruebas que se realizaron se enumeran en la Tabla 5.11 del capítulo 4.

6.2. Recomendaciones

Se recomienda utilizar esta metodología de versionamiento, en bases de datos nuevas, esto con la finalidad de mantener las versiones de la base de datos desde su creación hasta las últimas versiones liberadas de la misma.

En caso contrario, para bases de datos maduras y en producción, las versiones podrán controlarse a partir de que se utiliza por primera vez el versionador en dicha base de datos, y debido a los problemas en los metadatos de las versiones de SQL Server 2008 y anteriores,

es recomendable utilizarlo en versiones más nuevas. [29]

6.3. Trabajo futuro

Entre algunas de las cosas que puede realizarse con esta metodología se destacan:

Incluir en los instaladores *ClickOnce* la revisión de version de base de datos.

De esta forma, al instalarse una versión del sistema, se podría realizar la comparación de las bases de datos para mantenerla en la versión compatible con la aplicación que se está instalando.

Sincronización de datos para tablas de sistema.

Esto permitiría poder migrar la información de las tablas que se utilizan para control de sistema, tales como catálogos de tipos y estados.

Migrar la metodología a otro motor de base de datos.

Dado que muchos sistemas web y de desarrollo OpenSoruce se dan bajo una arquitectura abierta de base de datos, se podría migrar de *Microsoft SQL Server* a *MySQL*.

Bibliografía

[1] Carlo a. Curino, Hyun J. Moon, MyungWon Ham, and Carlo Zaniolo. The PRISM Workwench: Database Schema Evolution without Tears. *2009 IEEE 25th International Conference on Data Engineering*, pages 1523–1526, March 2009.

[2] Han-chieh Wei and Ramez Elmasri. Study and Comparison of Schema Versioning and Database Conversion Techniques for Bi-temporal Databases Han-Chieh Wei and Ramez Elmasri Department of Computer Science and Engineering The University of Texas at Arlington. pages 1–11.

[3] Carlo A Curino, Politecnico Milano, and Carlo Zaniolo. Graceful Database Schema Evolution : the PRISM Workbench. 2008.

[4] John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37:383–393, 1995.

[5] James F Terwilliger, Philip A Bernstein, and Adi Unnithan. Worry-Free Database Upgrades : Automated Model-Driven Evolution of Schemas and Complex Mappings. pages 1191–1194, 2010.

[6] RedGate. SQL Compare.

[7] RedGate. SQL Source Control, 2013.

[8] Neeraj Sharma, Liviu Perniu, Raul F Chong, Abhishek Iyer, Chaitali Nandan, Adicristina Mitea, Mallarswami Nonvinkere, and Mirela Danubianu. *Database Fundamentals*. IBM Corporation 2010, Armonk, N.Y., first edition, 2010.

[9] Rafael Paré Camps. Introducción a las bases de datos.

[10] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. Number c. The McGraw-Hill, 4 edition, 2004.

[11] Joseph Sack. *SQL Server 2005 T-SQL recipes*. Apress, 2005.

[12] Yan-nei Law, Haixun Wang, and Carlo Zaniolo. Query Languages and Data Models for Database Sequences and Data Streams. *VLDB '04 Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, 2004.

[13] William Sprouse. *Structured Query Language (SQL) Tutorial*. Number December. Colorado State Unversity, Foft Collins, Co, 2004.

[14] Romero Martínez. Lenguajes de bases de. *Tesis de Maestría*, pages 1–8, 1999.

[15] Microsoft Corporation. *Microsoft SQL Server 2012 Transact-SQL DML Reference*. 2012.

[16] Microsoft Corporation. *Transact-SQL Data Definition Language (DDL) Reference*. Microsoft Corporation, 2012.

[17] Alex Kriegel and Boris M. Trukhnov. *SQL Bible*. Wiley Publishing, Inc., Indianapolis, IN, second edition, 2008.

[18] Michael Coles. *Pro T-SQL 2008 Programmer's Guide*. Apress, 2008.

[19] Carlo A. Curino, Hyun J. Moon, Alin Deutsch, and Zaniolo Carlo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM. *Proceedings of the VLDB Endowment*, 4(2), 2010.

[20] ESRI. Versioning. 2004.

[21] Benchiao Jai. *Automating Software Deployment*. Ph.d., New York University, 2013.

[22] Antonio Carzaniga. A Characterization of the Software Deployment Process and a Survey of Related Technologies. 1997.

[23] Microsoft Corporation. sys.objects (Transact-SQL), 2008.

[24] Universidad Simon Bolivar. Tipo de Metodología de Sistemas. 2009.

[25] Leonardo Javier Malpica. *Sistema de información para la gestión de recursos humanos*. PhD thesis, Universidad Católica Andrés Bello, 2004.

[26] Bonilla Sanchez Silverio. Metodología Evolutiva Incremental Mediante Prototipo y Técnicas Orientada a Objeto (MEI/P-OO) para el desarrollo de aplicaciones. 2010.

[27] Luis Castellanos. Desarrollo de Sistemas de Información bajo un enfoque incremental. 2011.

[28] Hyun Jin Moon. Supporting Schema Evolution in Information Systems and Historical Databases. 2008.

[29] Sameer Verkhedkar. Sp_refreshsqlmodule changes object definition, 2011.

Apéndice A

Entidad Relación de Metadatos

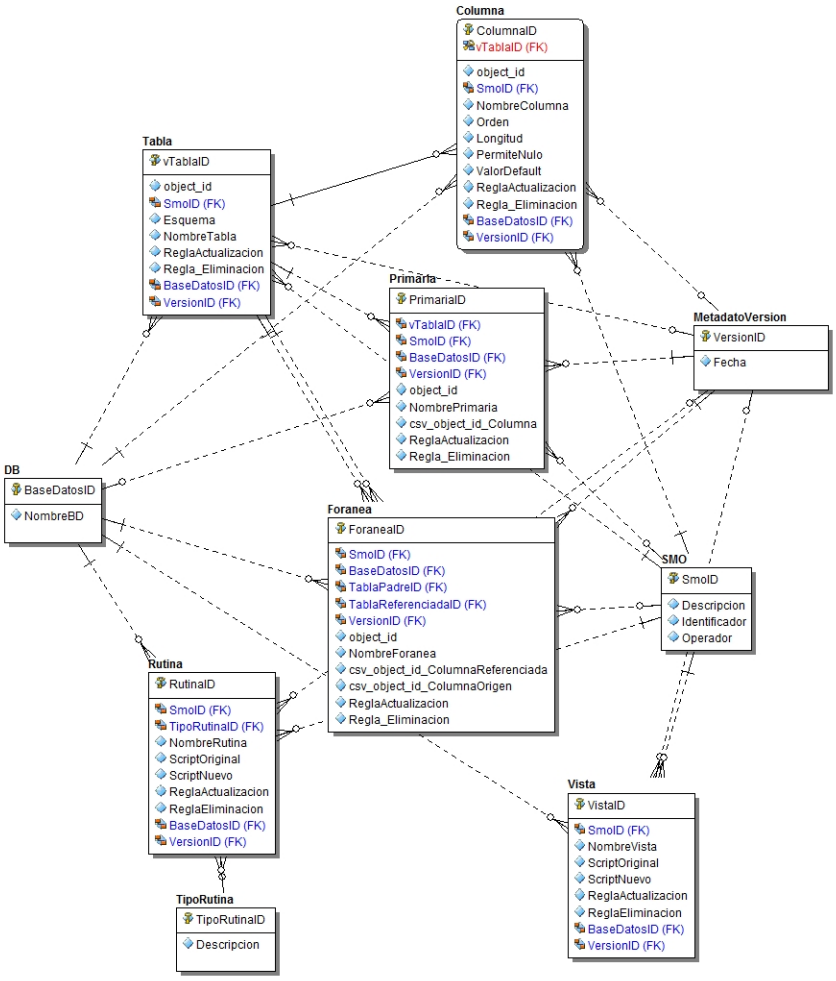


Figura A.1: Metadatos - Diseño de base de datos que almacena la información de las versiones registradas.

Apéndice B

Scripts SQL

Se anexan una serie de scripts que se requieren para replicar cada una de las pruebas realizadas a la metodología de versionamiento basada en SMO, y que además explican el funcionamiento de la misma.

B.1. ScriptsCreacionTablasMetadatos1.4.sql

Este es un ejemplo de como referenciar en el texto a un listado de código. Por ejemplo, digamos que quiero hablar del código B.1, entonces uso la etiqueta para referencia cruzada.

Script B.1: Ejemplo de caption

```
USE [Metadatos1.4]
GO
/****** Object:  Table [dbo].[DB]    Script Date: 08/29/2013
   23:12:09 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[DB](
        [BaseDatosID] [int] IDENTITY(1,1) NOT NULL,
        [NombreBD] [varchar](30) NULL,
 CONSTRAINT [PK11] PRIMARY KEY NONCLUSTERED
(
        [BaseDatosID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]



/*SCRIPT CREACION TABLAS DE Metadatos1.4*/
USE [Metadatos1.4]
GO
/****** Object:  Table [dbo].[DB]    Script Date: 08/29/2013
   23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[DB](
        [BaseDatosID] [int] IDENTITY(1,1) NOT NULL,
        [NombreBD] [varchar](30) NULL,
 CONSTRAINT [PK11] PRIMARY KEY NONCLUSTERED
(
        [BaseDatosID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[MetadatoVersion]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MetadatoVersion](
        [VersionID] [int] IDENTITY(1,1) NOT NULL,
        [Fecha] [datetime] NOT NULL,
 CONSTRAINT [PK12] PRIMARY KEY NONCLUSTERED
(
```

```
        [VersionID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[SMO]    Script Date: 08/29/2013
   23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[SMO](
        [SmoID] [int] IDENTITY(1,1) NOT NULL,
        [Descripcion] [varchar](50) NULL,
        [Identificador] [varchar](10) NULL,
        [Operador] [varchar](10) NULL,
 CONSTRAINT [PK6] PRIMARY KEY NONCLUSTERED
(
        [SmoID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Script]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Script](
        [ScriptID] [int] IDENTITY(1,1) NOT NULL,
        [Fecha] [datetime] NULL,
        [DeLaVersion] [float] NOT NULL,
        [ALaVersion] [float] NOT NULL,
        [Script] [varchar](max) NULL,
 CONSTRAINT [PK10] PRIMARY KEY NONCLUSTERED
(
        [ScriptID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
```

```
GO
/****** Object:  Table [dbo].[TipoRutina]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[TipoRutina](
        [TipoRutinaID] [int] IDENTITY(1,1) NOT NULL,
        [Descripcion] [varchar](20) NULL,
 CONSTRAINT [PK8] PRIMARY KEY NONCLUSTERED
(
        [TipoRutinaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Vista]    Script Date: 08/29/2013
    23:12:09 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Vista](
        [VistaID] [int] IDENTITY(1,1) NOT NULL,
        [SmoID] [int] NOT NULL,
        [NombreVista] [varchar](50) NOT NULL,
        [ScriptOriginal] [varchar](max) NULL,
        [ScriptNuevo] [varchar](max) NOT NULL,
        [ReglaActualizacion] [varchar](max) NOT NULL,
        [ReglaEliminacion] [varchar](max) NOT NULL,
        [BaseDatosID] [int] NOT NULL,
        [VersionID] [int] NULL,
 CONSTRAINT [PK3] PRIMARY KEY NONCLUSTERED
(
        [VistaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Tabla]    Script Date: 08/29/2013
    23:12:09 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Tabla](
        [vTablaID] [int] IDENTITY(1,1) NOT NULL,
        [object_id] [int] NULL,
        [SmoID] [int] NOT NULL,
        [Esquema] [varchar](10) NULL,
        [NombreTabla] [varchar](40) NOT NULL,
        [ReglaActualizacion] [varchar](100) NOT NULL,
        [Regla_Eliminacion] [varchar](100) NOT NULL,
        [BaseDatosID] [int] NOT NULL,
        [VersionID] [int] NULL,
 CONSTRAINT [PK1] PRIMARY KEY NONCLUSTERED
(
        [vTablaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
```

```
/****** Object:  Table [dbo].[Rutina]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Rutina](
        [RutinaID] [int] IDENTITY(1,1) NOT NULL,
        [SmoID] [int] NOT NULL,
        [TipoRutinaID] [int] NULL,
        [NombreRutina] [varchar](50) NOT NULL,
        [ScriptOriginal] [varchar](max) NULL,
        [ScriptNuevo] [varchar](max) NOT NULL,
        [ReglaActualizacion] [varchar](max) NOT NULL,
        [ReglaEliminacion] [varchar](max) NOT NULL,
        [BaseDatosID] [int] NOT NULL,
        [VersionID] [int] NULL,
 CONSTRAINT [PK4] PRIMARY KEY NONCLUSTERED
(
        [RutinaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Primaria]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Primaria](
        [PrimariaID] [int] IDENTITY(1,1) NOT NULL,
        [vTablaID] [int] NOT NULL,
        [SmoID] [int] NULL,
        [BaseDatosID] [int] NULL,
        [VersionID] [int] NOT NULL,
        [object_id] [int] NOT NULL,
        [NombrePrimaria] [varchar](100) NOT NULL,
        [csv_object_id_Columna] [varchar](max) NOT NULL,
        [ReglaActualizacion] [varchar](100) NOT NULL,
        [Regla_Eliminacion] [varchar](100) NOT NULL,
 CONSTRAINT [PK13] PRIMARY KEY NONCLUSTERED
(
        [PrimariaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
```

```
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Foranea]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Foranea](
        [ForaneaID] [int] IDENTITY(1,1) NOT NULL,
        [SmoID] [int] NULL,
        [BaseDatosID] [int] NOT NULL,
        [TablaPadreID] [int] NOT NULL,
        [TablaReferenciadaID] [int] NOT NULL,
        [VersionID] [int] NOT NULL,
        [object_id] [int] NULL,
        [NombreForanea] [varchar](100) NULL,
        [csv_object_id_ColumnaReferenciada] [varchar](max) NULL
           ,
        [csv_object_id_ColumnaOrigen] [varchar](max) NULL,
```

```
        [ReglaActualizacion] [varchar](100) NOT NULL,
        [Regla_Eliminacion] [varchar](100) NOT NULL,
 CONSTRAINT [PK14] PRIMARY KEY NONCLUSTERED
(
        [ForaneaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Columna]    Script Date:
   08/29/2013 23:12:09 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Columna](
        [ColumnaID] [int] IDENTITY(1,1) NOT NULL,
        [vTablaID] [int] NOT NULL,
        [object_id] [int] NOT NULL,
        [SmoID] [int] NOT NULL,
        [NombreColumna] [varchar](50) NOT NULL,
```

```
        [Orden] [int] NULL,
        [Longitud] [int] NULL,
        [PermiteNulo] [varchar](3) NULL,
        [ValorDefault] [varchar](50) NULL,
        [ReglaActualizacion] [varchar](100) NOT NULL,
        [Regla_Eliminacion] [varchar](100) NOT NULL,
        [BaseDatosID] [int] NOT NULL,
        [VersionID] [int] NULL,
 CONSTRAINT [PK2] PRIMARY KEY NONCLUSTERED
(
        [ColumnaID] ASC,
        [vTablaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  ForeignKey [RefDB19]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Columna]  WITH CHECK ADD  CONSTRAINT [
   RefDB19] FOREIGN KEY([BaseDatosID])
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Columna] CHECK CONSTRAINT [RefDB19]
```

```
GO
/****** Object:  ForeignKey [RefMetadatoVersion28]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Columna]  WITH CHECK ADD  CONSTRAINT [
   RefMetadatoVersion28] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Columna] CHECK CONSTRAINT [
   RefMetadatoVersion28]
GO
/****** Object:  ForeignKey [RefSMO5]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Columna]  WITH CHECK ADD  CONSTRAINT [
   RefSMO5] FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
ALTER TABLE [dbo].[Columna] CHECK CONSTRAINT [RefSMO5]
GO
/****** Object:  ForeignKey [RefTabla10]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Columna]  WITH CHECK ADD  CONSTRAINT [
   RefTabla10] FOREIGN KEY([vTablaID])
REFERENCES [dbo].[Tabla] ([vTablaID])
GO
ALTER TABLE [dbo].[Columna] CHECK CONSTRAINT [RefTabla10]
GO
```

```
/****** Object:  ForeignKey [RefDB38]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Foranea]  WITH CHECK ADD  CONSTRAINT [
   RefDB38] FOREIGN KEY([BaseDatosID])
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Foranea] CHECK CONSTRAINT [RefDB38]
GO
/****** Object:  ForeignKey [RefMetadatoVersion43]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Foranea]  WITH CHECK ADD  CONSTRAINT [
   RefMetadatoVersion43] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Foranea] CHECK CONSTRAINT [
   RefMetadatoVersion43]
GO
/****** Object:  ForeignKey [RefSMO37]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Foranea]  WITH CHECK ADD  CONSTRAINT [
   RefSMO37] FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
ALTER TABLE [dbo].[Foranea] CHECK CONSTRAINT [RefSMO37]
GO
/****** Object:  ForeignKey [RefTabla40]    Script Date:
```

```
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Foranea]  WITH CHECK ADD  CONSTRAINT [
   RefTabla40] FOREIGN KEY([TablaPadreID])
REFERENCES [dbo].[Tabla] ([vTablaID])
GO
ALTER TABLE [dbo].[Foranea] CHECK CONSTRAINT [RefTabla40]
GO
/****** Object:  ForeignKey [RefTabla41]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Foranea]  WITH CHECK ADD  CONSTRAINT [
   RefTabla41] FOREIGN KEY([TablaReferenciadaID])
REFERENCES [dbo].[Tabla] ([vTablaID])
GO
ALTER TABLE [dbo].[Foranea] CHECK CONSTRAINT [RefTabla41]
GO
/****** Object:  ForeignKey [RefDB36]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Primaria]  WITH CHECK ADD  CONSTRAINT [
   RefDB36] FOREIGN KEY([BaseDatosID])
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Primaria] CHECK CONSTRAINT [RefDB36]
GO
/****** Object:  ForeignKey [RefMetadatoVersion42]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Primaria]  WITH CHECK ADD  CONSTRAINT [
```

```
   RefMetadatoVersion42] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Primaria] CHECK CONSTRAINT [
   RefMetadatoVersion42]
GO
/****** Object:  ForeignKey [RefSMO35]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Primaria]  WITH CHECK ADD  CONSTRAINT [
   RefSMO35] FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
ALTER TABLE [dbo].[Primaria] CHECK CONSTRAINT [RefSMO35]
GO
/****** Object:  ForeignKey [RefTabla34]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Primaria]  WITH CHECK ADD  CONSTRAINT [
   RefTabla34] FOREIGN KEY([vTablaID])
REFERENCES [dbo].[Tabla] ([vTablaID])
GO
ALTER TABLE [dbo].[Primaria] CHECK CONSTRAINT [RefTabla34]
GO
/****** Object:  ForeignKey [RefDB20]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Rutina]  WITH CHECK ADD  CONSTRAINT [RefDB20
   ] FOREIGN KEY([BaseDatosID])
```

```
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Rutina] CHECK CONSTRAINT [RefDB20]
GO
/****** Object:  ForeignKey [RefMetadatoVersion30]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Rutina]  WITH CHECK ADD  CONSTRAINT [
   RefMetadatoVersion30] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Rutina] CHECK CONSTRAINT [
   RefMetadatoVersion30]
GO
/****** Object:  ForeignKey [RefSMO7]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Rutina]  WITH CHECK ADD  CONSTRAINT [RefSMO7
   ] FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
ALTER TABLE [dbo].[Rutina] CHECK CONSTRAINT [RefSMO7]
GO
/****** Object:  ForeignKey [RefTipoRutina14]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Rutina]  WITH CHECK ADD  CONSTRAINT [
   RefTipoRutina14] FOREIGN KEY([TipoRutinaID])
REFERENCES [dbo].[TipoRutina] ([TipoRutinaID])
```

```
GO
ALTER TABLE [dbo].[Rutina] CHECK CONSTRAINT [RefTipoRutina14]
GO
/****** Object:  ForeignKey [RefDB18]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Tabla]  WITH CHECK ADD  CONSTRAINT [RefDB18]
    FOREIGN KEY([BaseDatosID])
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Tabla] CHECK CONSTRAINT [RefDB18]
GO
/****** Object:  ForeignKey [RefMetadatoVersion26]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Tabla]  WITH CHECK ADD  CONSTRAINT [
   RefMetadatoVersion26] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Tabla] CHECK CONSTRAINT [
   RefMetadatoVersion26]
GO
/****** Object:  ForeignKey [RefSMO4]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Tabla]  WITH CHECK ADD  CONSTRAINT [RefSMO4]
    FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
```

```
ALTER TABLE [dbo].[Tabla] CHECK CONSTRAINT [RefSMO4]
GO
/****** Object:  ForeignKey [RefDB21]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Vista]  WITH CHECK ADD  CONSTRAINT [RefDB21]
    FOREIGN KEY([BaseDatosID])
REFERENCES [dbo].[DB] ([BaseDatosID])
GO
ALTER TABLE [dbo].[Vista] CHECK CONSTRAINT [RefDB21]
GO
/****** Object:  ForeignKey [RefMetadatoVersion29]    Script
   Date: 08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Vista]  WITH CHECK ADD  CONSTRAINT [
   RefMetadatoVersion29] FOREIGN KEY([VersionID])
REFERENCES [dbo].[MetadatoVersion] ([VersionID])
GO
ALTER TABLE [dbo].[Vista] CHECK CONSTRAINT [
   RefMetadatoVersion29]
GO
/****** Object:  ForeignKey [RefSMO8]    Script Date:
   08/29/2013 23:12:09 ******/
ALTER TABLE [dbo].[Vista]  WITH CHECK ADD  CONSTRAINT [RefSMO8]
    FOREIGN KEY([SmoID])
REFERENCES [dbo].[SMO] ([SmoID])
GO
ALTER TABLE [dbo].[Vista] CHECK CONSTRAINT [RefSMO8]
```

```
GO



SET IDENTITY_INSERT SMO ON
insert into SMO (SmoID,Descripcion,Identificador,Operador)
   values (1,Creación de tabla','UN','UN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(2,'Cambio nombre tabla','UR','UR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(3,'Creacion de columna','CN','CN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(4,'Cambio nombre Columna','CR','CR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(5,'Crea llaves primarias','PKN','PKN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(6,'Renombra llaves primarias','PKR','PKR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(7,'Cambia las columnas de la llave primaria','PKM','
   PKM');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(8,'Crea llaves foraneas','FKN','FKN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(9,'Renombra llaves foraneas','FKR','FKR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(10,'Cambia las columnas involucradas en llave foranea
   ','FKM','FKM');
```

```
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(11,'Elimina llaves primarias','PKD','PKD');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(12,'Elimina llaves foraneas','FKD','FKD');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(13,'Eliminacion de tabla','UD','UD');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(14,'Eliminacion de Columna','CD','CD');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(15,'Creacion de una vista','VN','VN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(16,'Eliminacion de una vista','VD','VN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(17,'Cambio nombre de una vista','VR','VR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(18,'Cambio contenido de una vista','VM','VM');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(19,'Creacion de una Rutina','RN','RN');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(20,'Eliminacion de una Rutina','RD','RD');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(21,'Cambio nombre de una Rutina','RR','RR');
insert␣into␣SMO␣(SmoID,Descripcion,Identificador,Operador)␣
   values␣(22,'Cambio contenido de una Rutina','RM','RM');
SET␣IDENTITY_INSERT␣SMO␣OFF
```

```
SET␣IDENTITY_INSERT␣TipoRutina␣ON
INSERT␣INTO␣TipoRutina␣(TipoRutinaID,␣Descripcion)␣VALUES␣(1,␣'
   Fn')
INSERT␣INTO␣TipoRutina␣(TipoRutinaID,␣Descripcion)␣VALUES␣(2,␣'
   P')
SET␣IDENTITY_INSERT␣TipoRutina␣OFF

GO

SET␣ANSI_NULLS␣ON
GO
SET␣QUOTED_IDENTIFIER␣ON
GO
SET␣ANSI_PADDING␣ON
GO
CREATE␣TABLE␣[dbo].[DependenciaSmoVista](
[DependenciaVistaID]␣[int]␣IDENTITY(1,1)␣NOT␣NULL,
[VistaID]␣[int]␣NOT␣NULL,
[Dependencia]␣[varchar](max)␣NOT␣NULL,
␣CONSTRAINT␣[PK_DependenciaSmoVista]␣PRIMARY␣KEY␣CLUSTERED
(
␣␣␣␣␣␣␣␣[DependenciaVistaID]␣ASC
)WITH␣(PAD_INDEX␣␣=␣OFF,␣STATISTICS_NORECOMPUTE␣␣=␣OFF,␣
   IGNORE_DUP_KEY␣=␣OFF,␣ALLOW_ROW_LOCKS␣␣=␣ON,␣ALLOW_PAGE_LOCKS
   ␣␣=␣ON)␣ON␣[PRIMARY]
)␣ON␣[PRIMARY]
```

```
GO
SET␣ANSI_PADDING␣OFF
GO
```

B.2. ScriptIncluirVersionador.sql

Script B.2: Incluir versionador en base de daots de desarrollo

```
USE [DB_DESARROLLO] --Cambie esta linea si su base de datos es
   otra.


GO
/****** Object:  Schema [ver]    Script Date: 08/29/2013
   23:06:35 ******/
CREATE SCHEMA [ver] AUTHORIZATION [dbo]
GO
/****** Object:  UserDefinedFunction [ver].[fix_view_create]
      Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_view_create]
(
        @view_name varchar(max),
        @view_code varchar(max)
```

```
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
        declare @antes as varchar(max)
        declare @entre as varchar(max)
        declare @luego as varchar(max)
        declare @posicion_nombre as integer;

        set @a = @view_code
        set @antes=''
        set @entre=''
        set @luego=''

        select @antes=substring(@a,1,patindex('%CREATE␣VIEW%',
           @a)+11)
        select @luego=substring(@a,patindex('%CREATE␣VIEW%',@a)
           +11,LEN(@a))
        select @luego=substring(@luego,patindex('%select%',
           @luego),LEN(@luego))
        select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣VIEW',
            ''), @luego, '');

        select @entre=REPLACE(@entre, '[', '*');
```

```
        select @entre=REPLACE(@entre, ']', '*');
        select @entre=REPLACE(@entre, '.', '*');
        select @entre=REPLACE(@entre, '␣', '*');
        select @entre=REPLACE(@entre, '␣', '*');
        select @entre=REPLACE(@entre, '(', '*');
        select @entre=REPLACE(@entre, 'as', '*');

        set @posicion_nombre=0
        select @posicion_nombre = patindex('%*'+@view_name+'*%'
           ,@entre)

        set @ResultVar = @view_code
        if @posicion_nombre=0
        begin
                set @ResultVar=@antes+'␣'+ @view_name +'␣as␣'+
                   @luego
        end

        RETURN @ResultVar
END
GO
/****** Object:  UserDefinedFunction [ver].[fix_sp_create]
   Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
CREATE FUNCTION [ver].[fix_sp_create]
(
        @sp_name varchar(max),
        @sp_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
        declare @antes as varchar(max)
        declare @entre as varchar(max)
        declare @luego as varchar(max)
        declare @posicion_nombre as integer;

        set @a = @sp_code
        set @antes=''
        set @entre=''
        set @luego=''

        select @antes=substring(@a,1,patindex('%CREATE␣
           PROCEDURE%',@a)+16)
        select @luego=substring(@a,patindex('%CREATE␣PROCEDURE%
           ',@a)+16,LEN(@a))
        DECLARE @luego_tmp varchar(max)
```

```
        SET @luego_tmp = REPLACE(@luego, '@', '(')
        select @luego=substring(@luego,patindex('%(%',
           @luego_tmp),LEN(@luego))
        select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣
           PROCEDURE', ''), @luego, '');

        select @entre=REPLACE(@entre, '[', '*');
        select @entre=REPLACE(@entre, ']', '*');
        select @entre=REPLACE(@entre, '.', '*');
        select @entre=REPLACE(@entre, '␣', '*');
        select @entre=REPLACE(@entre, '␣', '*');
        select @entre=REPLACE(@entre, '(', '*');

        set @posicion_nombre=0
        select @posicion_nombre = patindex('%*'+@sp_name+'*%',
           @entre)

        set @ResultVar = @sp_code
        if @posicion_nombre=0
        begin
                set @ResultVar=@antes+'␣'+ @sp_name +'␣'+@luego
        end

        RETURN @ResultVar
END
GO
```

```
/****** Object:  UserDefinedFunction [ver].[fix_fnc_create]
   Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_fnc_create]
(
        @fnc_name varchar(max),
        @fnc_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
        declare @antes as varchar(max)
        declare @entre as varchar(max)
        declare @luego as varchar(max)
        declare @posicion_nombre as integer;

        set @a = @fnc_code
        set @antes=''
        set @entre=''
        set @luego=''
```

```
select @antes=substring(@a,1,patindex('%CREATE␣FUNCTION
   %',@a)+15)
select @luego=substring(@a,patindex('%CREATE␣FUNCTION%'
   ,@a)+15,LEN(@a))
select @luego=substring(@luego,patindex('%(%',@luego),
   LEN(@luego))
select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣
   FUNCTION', ''), @luego, '');

select @entre=REPLACE(@entre, '[', '*');
select @entre=REPLACE(@entre, ']', '*');
select @entre=REPLACE(@entre, '.', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '(', '*');

set @posicion_nombre=0
select @posicion_nombre = patindex('%*'+@fnc_name+'*%',
   @entre)

set @ResultVar = @fnc_code
if @posicion_nombre=0
begin
        set @ResultVar=@antes+'␣'+ @fnc_name +'␣'+
           @luego
end
```

```
        RETURN @ResultVar
END
GO
/****** Object:  StoredProcedure [ver].[VersionaPrimaria]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaPrimaria]
( @Version int,
  @DB_id INT
)
AS
BEGIN

--LAS PK QUE FUERON ELIMINADAS DEL ESQUEMA
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
```

```
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        mt.vTablaID,
        11 as SmoID, -- "borra PK"
        mt.BaseDatosID,
        @Version,
        mp.object_id, -- deleted object id
        '',
        '',
        'ver.PKD '''+mt.NombreTabla+''','''+mp.NombrePrimaria+'
           ''' ,
        'ver.PKN '''+mt.NombreTabla+''','''+mp.NombrePrimaria+'
           '','''+mp.csv_object_id_Columna+''''
        from [Metadatos1.4].DBO.Primaria mp
        inner join [Metadatos1.4].dbo.Tabla mt on mt.vTablaID=
           mp.vTablaID and mt.versionID = (select MAX(VersionID)
            from [Metadatos1.4].dbo.Tabla metatv where metatv.
           object_id = mt.object_id)
        left join sys.key_constraints kc on kc.object_id=mp.
           object_id and kc.parent_object_id = mt.object_id and
           mt.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatv where metatv.object_id
            = mt.object_id)
        where kc.object_id is null
```

```
        and mp.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Primaria metapv where metapv.
           object_id = mp.object_id)
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);


--LAS PK QUE NO EXISTEN EN LOS METADATOS
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        5 as SmoID, -- "crea PK"
        mt.BaseDatosID,
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
```

```
        object_id=i.object_id and ic.index_id=i.index_id
        ORDER BY ic.key_ordinal FOR XML PATH (''')), 1, 1, ''')
        ) as csv_object_id_Columna,
    'ver.PKN␣'''+mt.NombreTabla+''',''+kc.name+''',''+(
        SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
        varchar(45)) FROM sys.index_columns ic WHERE ic.
        object_id=i.object_id and ic.index_id=i.index_id
        ORDER BY ic.key_ordinal FOR XML PATH (''')), 1, 1, ''')
        )+'''',
    'ver.PKD␣'''+mt.NombreTabla+''',''+kc.name+''''
    from sys.key_constraints kc
    inner join  sys.indexes i on kc.parent_object_id=i.
        object_id and kc.name=i.name
inner join [Metadatos1.4].dbo.Tabla mt on kc.
    parent_object_id = mt.object_id and mt.versionID = (
    select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
    metatv where metatv.object_id = mt.object_id)
     LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
        object_id = kc.object_id and mp.versionID = (select
        MAX(VersionID) from [Metadatos1.4].dbo.Primaria
        metapv where metapv.object_id = mp.object_id)
     where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
     and SCHEMA_NAME(kc.schema_id) != 'ver'
     AND kc.type='PK'
     and (mp.PrimariaID is null OR mp.SmoID not in
        (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));
```

```
--LAS PK QUE EXISTEN PERO CAMBIARON DE NOMBRE
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        6 as SmoID, -- "renombra PK"
        mt.BaseDatosID,
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           ) as csv_object_id_Columna,
        'ver.PKR '''+mt.NombreTabla+''','''+mp.NombrePrimaria+'
           '','''+kc.name+'''',
```

```
        'ver.PKR ''+mt.NombreTabla+''',''+kc.name+''',''+mp.
           NombrePrimaria+'''' --rollback
        from sys.key_constraints kc
        inner join  sys.indexes i on kc.parent_object_id=i.
           object_id and kc.name=i.name
    inner join [Metadatos1.4].dbo.Tabla mt on kc.
       parent_object_id = mt.object_id and mt.versionID = (
       select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
       metatv where metatv.object_id = mt.object_id)
        LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
           object_id = kc.object_id and mp.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Primaria
           metapv where metapv.object_id = mp.object_id)
        where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(kc.schema_id) != 'ver'
        AND kc.type='PK'
        and mp.object_id is not null
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        and mp.NombrePrimaria!=kc.name;


--LAS PK QUE EXISTEN PERO SE LE MODIFICARON SUS COLUMNAS (creo
   q nunca pasara esto, hace delete->create)
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
```

```
            ,[BaseDatosID]
            ,[VersionID]
            ,[object_id]
            ,[NombrePrimaria]
            ,[csv_object_id_Columna]
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        mt.vTablaID,
        7 as SmoID, -- "modify columns PK"
        mt.BaseDatosID,
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           ) as csv_object_id_Columna,
        'ver.PKM␣'''+mt.NombreTabla+''','''+kc.name+''','''+(
           SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           )+'''',
```

```
        'ver.PKM␣'''+mt.NombreTabla+''','''+kc.name+''','''+mp.
           csv_object_id_Columna+'''' -- rollback
        from sys.key_constraints kc
        inner join  sys.indexes i on kc.parent_object_id=i.
           object_id and kc.name=i.name
    inner join [Metadatos1.4].dbo.Tabla mt on kc.
       parent_object_id = mt.object_id and mt.versionID = (
       select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
       metatv where metatv.object_id = mt.object_id)
        LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
           object_id = kc.object_id and mp.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Primaria
           metapv where metapv.object_id = mp.object_id)
        where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(kc.schema_id) != 'ver'
        AND kc.type='PK'
        and mp.object_id is not null
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        and mp.csv_object_id_Columna!=(SELECT STUFF((SELECT ','
            + CAST(ic.column_id AS varchar(45)) FROM sys.
           index_columns ic WHERE ic.object_id=i.object_id and
           ic.index_id=i.index_id ORDER BY ic.key_ordinal FOR
           XML PATH ('')), 1, 1, ''));


select '[VersionaPrimaria]'
```

```
END
GO
/****** Object:  StoredProcedure [ver].[VersionaForanea]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaForanea]
( @Version int,
  @DB_id INT
)
AS
BEGIN
--12 LAS FK QUE FUERON ELIMINADAS DEL ESQUEMA
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
           ,[BaseDatosID]
           ,[TablaPadreID]
           ,[TablaReferenciadaID]
           ,[VersionID]
           ,[object_id]
           ,[NombreForanea]
           ,[csv_object_id_ColumnaReferenciada]
           ,[csv_object_id_ColumnaOrigen]
           ,[ReglaActualizacion]
```

```
           ,[Regla_Eliminacion])
select
        12 as SmoID, -- "borra FK"
        mtp.BaseDatosID,
        mtp.vTablaID as TablaPadreID,
        mtr.vTablaID as TablaReferenciadaID,
        @Version,
        mf.object_id, -- deleted object id
        '',
        '',
        '',
        'ver.FKD '''+mtp.NombreTabla+''','''+mf.NombreForanea+'
           ''' ,
        'ver.FKN '''+mtp.NombreTabla+''','''+mtr.NombreTabla+''
           ','''+mf.NombreForanea+''','''+mf.
           csv_object_id_ColumnaOrigen+''','''+mf.
           csv_object_id_ColumnaReferenciada+''''
        from [Metadatos1.4].DBO.Foranea mf
        inner join [Metadatos1.4].dbo.Tabla mtp on mtp.vTablaID
           =mf.TablaPadreID and mtp.versionID = (select MAX(
           VersionID) from [Metadatos1.4].dbo.Tabla metatvp
           where metatvp.object_id = mtp.object_id)
        inner join [Metadatos1.4].dbo.Tabla mtr on mtr.vTablaID
           =mf.TablaReferenciadaID and mtr.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Tabla metatvr
           where metatvr.object_id = mtr.object_id)
```

```
        left join sys.foreign_keys fk on fk.object_id=mf.
           object_id and fk.parent_object_id = mtp.object_id and
            mtp.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatvp where metatvp.
           object_id = mtp.object_id)
        where fk.object_id is null
        and mf.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Foranea metafv where metafv.
           object_id = mf.object_id)
        AND mf.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);
--8 LAS FK QUE NO EXISTEN EN LOS METADATOS
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
           ,[BaseDatosID]
           ,[TablaPadreID]
           ,[TablaReferenciadaID]
           ,[VersionID]
           ,[object_id]
           ,[NombreForanea]
           ,[csv_object_id_ColumnaReferenciada]
           ,[csv_object_id_ColumnaOrigen]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        8 as SmoID, -- "crea FK"
```

```
mtp.BaseDatosID,
mtp.vTablaID,
mtr.vTablaID,
@Version,
fk.object_id,
fk.name,
(SELECT STUFF((SELECT ',' + CAST(fkc.
   referenced_column_id AS varchar(45)) from sys.
   foreign_key_columns fkc where fkc.
   constraint_object_id=fk.object_id order by fkc.
   constraint_column_id FOR XML PATH ('')), 1, 1, ''))
   as [csv_object_id_ColumnaReferenciada],
(SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
   AS varchar(45)) from sys.foreign_key_columns fkc
   where fkc.constraint_object_id=fk.object_id order by
   fkc.constraint_column_id FOR XML PATH ('')), 1, 1, ''
   )) as [csv_object_id_ColumnaOrigen],
'ver.FKN␣'''+mtp.NombreTabla+''','''+mtr.NombreTabla+''
   ','''+fk.name+''','''+(SELECT STUFF((SELECT ',' +
   CAST(fkc.parent_column_id AS varchar(45)) from sys.
   foreign_key_columns fkc where fkc.
   constraint_object_id=fk.object_id order by fkc.
   constraint_column_id FOR XML PATH ('')), 1, 1, ''))+'
   '','''+(SELECT STUFF((SELECT ',' + CAST(fkc.
   referenced_column_id AS varchar(45)) from sys.
   foreign_key_columns fkc where fkc.
```

```
            constraint_object_id=fk.object_id order by fkc.
            constraint_column_id FOR XML PATH (''')), 1, 1, ''))+'
            ''',
         'ver.FKD␣'''+mtp.NombreTabla+''',''+fk.name+''''
         from sys.foreign_keys fk
    inner join [Metadatos1.4].dbo.Tabla mtp on fk.
       parent_object_id = mtp.object_id and mtp.versionID = (
       select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
       metatvp where metatvp.object_id = mtp.object_id)
    inner join [Metadatos1.4].dbo.Tabla mtr on fk.
       referenced_object_id = mtr.object_id and mtr.versionID =
       (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
       metatvr where metatvr.object_id = mtr.object_id)
        LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
           object_id = fk.object_id and mf.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
            where metafv.object_id = mf.object_id)
         where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
         and SCHEMA_NAME(fk.schema_id) != 'ver'
         AND fk.type='F'
         and (mf.ForaneaID is null OR mf.SmoID not in
            (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));

--9 LAS FK QUE EXISTEN PERO CAMBIARON DE NOMBRE
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
            ([SmoID]
```

```
            ,[BaseDatosID]
            ,[TablaPadreID]
            ,[TablaReferenciadaID]
            ,[VersionID]
            ,[object_id]
            ,[NombreForanea]
            ,[csv_object_id_ColumnaReferenciada]
            ,[csv_object_id_ColumnaOrigen]
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        9 as SmoID, -- "rename FK"
        mtp.BaseDatosID,
        mtp.vTablaID,
        mtr.vTablaID,
        @Version,
        fk.object_id,
        fk.name,
        (SELECT STUFF((SELECT ',' + CAST(fkc.
           referenced_column_id AS varchar(45)) from sys.
           foreign_key_columns fkc where fkc.
           constraint_object_id=fk.object_id order by fkc.
           constraint_column_id FOR XML PATH ('')), 1, 1, ''))
           as [csv_object_id_ColumnaReferenciada],
        (SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
           AS varchar(45)) from sys.foreign_key_columns fkc
```

```
        where fkc.constraint_object_id=fk.object_id order by
        fkc.constraint_column_id FOR XML PATH (''')), 1, 1, '''
        )) as [csv_object_id_ColumnaOrigen],
    'ver.FKR␣'''+mtp.NombreTabla+''',''+mf.NombreForanea+'
        '',''+fk.name+''',
    'ver.FKR␣'''+mtp.NombreTabla+''',''+fk.name+''',''+mf
        .NombreForanea+''' --rollback
    from sys.foreign_keys fk
inner join [Metadatos1.4].dbo.Tabla mtp on fk.
   parent_object_id = mtp.object_id and mtp.versionID = (
   select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvp where metatvp.object_id = mtp.object_id)
inner join [Metadatos1.4].dbo.Tabla mtr on fk.
   referenced_object_id = mtr.object_id and mtr.versionID =
   (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvr where metatvr.object_id = mtr.object_id)
    LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
        object_id = fk.object_id and mf.versionID = (select
        MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
         where metafv.object_id = mf.object_id)
    where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
    and SCHEMA_NAME(fk.schema_id) != 'ver'
    AND fk.type='F'
    and mf.object_id is not null
    and mf.NombreForanea!=fk.name
```

```
        AND mf.SmoID in
            (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);
--10 LAS FK QUE EXISTEN PERO SE LE MODIFICARON SUS COLUMNAS (
   creo q nunca pasara esto, hace delete->create)
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
            ([SmoID]
            ,[BaseDatosID]
            ,[TablaPadreID]
            ,[TablaReferenciadaID]
            ,[VersionID]
            ,[object_id]
            ,[NombreForanea]
            ,[csv_object_id_ColumnaReferenciada]
            ,[csv_object_id_ColumnaOrigen]
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        10 as SmoID, -- "modify FK"
        mtp.BaseDatosID,
        mtp.vTablaID,
        mtr.vTablaID,
        @Version,
        fk.object_id,
        fk.name,
        (SELECT STUFF((SELECT ',' + CAST(fkc.
            referenced_column_id AS varchar(45)) from sys.
```

```
    foreign_key_columns fkc where fkc.
    constraint_object_id=fk.object_id order by fkc.
    constraint_column_id FOR XML PATH (''')), 1, 1, ''')
    as [csv_object_id_ColumnaReferenciada],
(SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
    AS varchar(45)) from sys.foreign_key_columns fkc
    where fkc.constraint_object_id=fk.object_id order by
    fkc.constraint_column_id FOR XML PATH (''')), 1, 1, ''
    )) as [csv_object_id_ColumnaOrigen],
'ver.FKM ''''+mtp.NombreTabla+''''','''+fk.name+''''','''+(
    SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
    AS varchar(45)) from sys.foreign_key_columns fkc
    where fkc.constraint_object_id=fk.object_id order by
    fkc.constraint_column_id FOR XML PATH (''')), 1, 1, ''
    ))+''''','''+(SELECT STUFF((SELECT ',' + CAST(fkc.
    referenced_column_id AS varchar(45)) from sys.
    foreign_key_columns fkc where fkc.
    constraint_object_id=fk.object_id order by fkc.
    constraint_column_id FOR XML PATH (''')), 1, 1, ''))+'
    ''',
'ver.FKM ''''+mtp.NombreTabla+''''','''+fk.name+''''','''+(
    SELECT STUFF((SELECT ',' + CAST(fkc.
    referenced_column_id AS varchar(45)) from sys.
    foreign_key_columns fkc where fkc.
    constraint_object_id=fk.object_id order by fkc.
    constraint_column_id FOR XML PATH (''')), 1, 1, ''))+'
```

```
        '',''')+(SELECT STUFF((SELECT ',' + CAST(fkc.
        parent_column_id AS varchar(45)) from sys.
        foreign_key_columns fkc where fkc.
        constraint_object_id=fk.object_id order by fkc.
        constraint_column_id FOR XML PATH ('')), 1, 1, ''))+'
        ''' -- rollback
    from sys.foreign_keys fk
inner join [Metadatos1.4].dbo.Tabla mtp on fk.
   parent_object_id = mtp.object_id and mtp.versionID = (
   select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvp where metatvp.object_id = mtp.object_id)
inner join [Metadatos1.4].dbo.Tabla mtr on fk.
   referenced_object_id = mtr.object_id and mtr.versionID =
   (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvr where metatvr.object_id = mtr.object_id)
    LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
       object_id = fk.object_id and mf.versionID = (select
       MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
        where metafv.object_id = mf.object_id)
    where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
    and SCHEMA_NAME(fk.schema_id) != 'ver'
    AND fk.type='F'
    and mf.object_id is not null
    AND mf.SmoID in
       (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
    and
```

```
        (  mf.csv_object_id_ColumnaOrigen!=(SELECT STUFF((
           SELECT ',' + CAST(fkc.parent_column_id AS varchar(45)
           ) from sys.foreign_key_columns fkc where fkc.
           constraint_object_id=fk.object_id order by fkc.
           constraint_column_id FOR XML PATH ('')), 1, 1, ''))
        or mf.csv_object_id_ColumnaReferenciada!=(SELECT STUFF
           ((SELECT ',' + CAST(fkc.referenced_column_id AS
           varchar(45)) from sys.foreign_key_columns fkc where
           fkc.constraint_object_id=fk.object_id order by fkc.
           constraint_column_id FOR XML PATH ('')), 1, 1, ''))
        );

select '[VersionaForanea]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionaColumna]
   Script Date: 11/13/2013 19:16:50 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [ver].[VersionaColumna]
( @Version int,
  @DB_id INT
```

```
)
AS
BEGIN
--14    Eliminacion de Columna
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT  mc.vTablaID,mc.object_id, 14,  mc.NombreColumna, mc.
   Orden, mc.Longitud, mc.PermiteNulo, mc.ValorDefault,
'ver.CD␣'+mt.NombreTabla +',␣'+ mc.NombreColumna +''
, mc.ReglaActualizacion
,@DB_id --base de datos
,@Version -- version
FROM [Metadatos1.4].DBO.Columna mc
inner join [Metadatos1.4].dbo.Tabla mt on mc.vTablaID = mt.
   vTablaID
left join sys.columns sc on mc.object_id=sc.column_id
where
mc.versionID = (select MAX(VersionID) from [Metadatos1.4].dbo.
   Columna metacv where metacv.object_id = mc.object_id)
AND mc.SmoID in (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
and sc.object_id is null;

--3     Creacion de columna
```

```
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT  mt.vTablaID,sc.column_id, 3,  C.COLUMN_NAME, C.
   ORDINAL_POSITION, C.CHARACTER_MAXIMUM_LENGTH, c.IS_NULLABLE,
   c.COLUMN_DEFAULT,
'ver.CN␣'+mt.NombreTabla +',␣'+ C.COLUMN_NAME +',␣'+ CASE WHEN
   C.CHARACTER_MAXIMUM_LENGTH IS NULL OR C.DATA_TYPE='text' OR C
   .DATA_TYPE='ntext' OR C.DATA_TYPE='image' OR C.DATA_TYPE='xml
   ' THEN C.DATA_TYPE  ELSE +''''+C.DATA_TYPE  + '(' + CASE WHEN
    C.CHARACTER_MAXIMUM_LENGTH=-1 THEN 'max' ELSE CAST(C.
   CHARACTER_MAXIMUM_LENGTH AS VARCHAR(50)) END+ ')''' END + ',␣
   '+C.IS_NULLABLE +',␣'+CAST(ISNULL(''''+C.COLUMN_DEFAULT+'''',
   'NULL') AS VARCHAR(20)) + '␣'
, 'ver.CD␣'+mt.NombreTabla +',␣'+ C.COLUMN_NAME +''
,@DB_id --base de datos
,@Version -- version
FROM INFORMATION_SCHEMA.COLUMNS c
inner join sys.columns sc on c.COLUMN_NAME = sc.name and
   OBJECT_NAME(sc.object_id)=c.TABLE_NAME
inner join [Metadatos1.4].DBO.Tabla mt on mt.object_id=sc.
   object_id and mt.versionID = (select MAX(VersionID) from [
   Metadatos1.4].dbo.Tabla metat where metat.object_id = mt.
   object_id)
```

```
LEFT JOIN [Metadatos1.4].DBO.Columna mc on mc.object_id=sc.
   column_id and mc.versionID = (select MAX(metacol.VersionID)
   from [Metadatos1.4].dbo.Columna metacol inner join [
   Metadatos1.4].dbo.Tabla metat on metat.vTablaID=metacol.
   vTablaID where metacol.object_id = mc.object_id and metat.
   object_id = mt.object_id)
where
c.TABLE_NAME!='sysdiagrams' and c.TABLE_SCHEMA != 'ver'
and (mc.ColumnaID is null OR mc.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))


--4      Cambio nombre Columna
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT mc.vTablaID, sc.column_id, 4--CR (renombrar columna)
, sc.name, null, null, null, null
, 'ver.CR␣' + mt.NombreTabla + ',␣'+ mc.NombreColumna + ',␣'+
   sc.name  regla1
,'ver.CR␣'+ mt.NombreTabla + ',␣'+ sc.name + ',␣'+ mc.
   NombreColumna  regla2
, @DB_id,@Version--1--version
FROM INFORMATION_SCHEMA.COLUMNS c
inner join sys.columns sc on c.COLUMN_NAME = sc.name and
   OBJECT_NAME(sc.object_id)=c.TABLE_NAME
```

```
inner join [Metadatos1.4].DBO.Tabla mt on mt.object_id=sc.
   object_id and mt.versionID = (select MAX(VersionID) from [
   Metadatos1.4].dbo.Tabla metat where metat.object_id = mt.
   object_id)
inner JOIN [Metadatos1.4].DBO.Columna mc on mc.object_id=sc.
   column_id and mc.versionID = (select MAX(metacol.VersionID)
   from [Metadatos1.4].dbo.Columna metacol inner join [
   Metadatos1.4].dbo.Tabla metat on metat.vTablaID=metacol.
   vTablaID where metacol.object_id = mc.object_id and metat.
   object_id = mt.object_id)
inner join [Metadatos1.4].dbo.Tabla mtc on mtc.vTablaID=mc.
   vTablaID
where  c.TABLE_NAME!= 'sysdiagrams'
and mtc.object_id=mt.object_id
and c.COLUMN_NAME != mc.NombreColumna
and c.TABLE_SCHEMA != 'ver'
AND mc.SmoID in (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)


END


GO
/****** Object:  UserDefinedFunction [ver].[fix_rutina_create]
      Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
CREATE FUNCTION [ver].[fix_rutina_create]
(
        @rut_name varchar(max),
        @rut_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @es_procedimiento as integer;

        set @es_procedimiento=0;

        select @es_procedimiento=COUNT(1) from sys.objects o
           where o.name=@rut_name and o.type='P'
        if @es_procedimiento>0 begin
                set @ResultVar = ver.fix_sp_create(@rut_name,
                   @rut_code)
        end else begin
                set @ResultVar = ver.fix_sp_create(@rut_name,
                   @rut_code)
        end

        RETURN @ResultVar
END
```

```
GO
/****** Object:  StoredProcedure [ver].[VersionaVista]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaVista]
( @Version int,
  @DB_id int    )
AS
BEGIN
        --Bug MsSQL 2008, no se puede llamar a
           sp_refreshsqlmodule luego de un sp_rename
        --http://connect.microsoft.com/SQLServer/feedback/
           details/644572/sp-refreshsqlmodule-changes-object-
           definition
        --declare @view_name varchar(max)
        --DECLARE view_cursor CURSOR FOR
        --      SELECT o.name
        --      FROM sys.objects AS o
        --      WHERE o.type = 'V'
        --      and SCHEMA_NAME(o.schema_id)!='ver'
        --OPEN view_cursor
        --FETCH NEXT FROM view_cursor INTO @view_name
        --WHILE @@FETCH_STATUS = 0
```

```
--BEGIN
--      EXEC sys.sp_refreshsqlmodule @view_name
--      FETCH NEXT FROM view_cursor INTO @view_name
--END
--CLOSE view_cursor
--DEALLOCATE view_cursor

-- 16   Eliminacion de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                    ([SmoID]
                    ,[NombreVista]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        16 as SmoID,
        mv.NombreVista, -- deleted view name
        mv.ScriptNuevo,
        '',
        'ver.VD␣'''+mv.NombreVista+'''',
        'ver.VN␣'''+mv.NombreVista+''','''+REPLACE(mv.
           ScriptNuevo,'''','''''')+'''',
        @DB_id,
```

```
        @Version
FROM [Metadatos1.4].DBO.Vista mv
LEFT JOIN sys.objects AS o ON mv.NombreVista =
   OBJECT_NAME(o.object_id)
WHERE mv.versionID = (select MAX(VersionID) from [
   Metadatos1.4].dbo.Vista metav where metav.NombreVista
    = mv.NombreVista)
AND mv.SmoID in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
AND o.object_id is null;

-- 15   Creacion de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                   ([SmoID]
                   ,[NombreVista]
                   ,[ScriptOriginal]
                   ,[ScriptNuevo]
                   ,[ReglaActualizacion]
                   ,[ReglaEliminacion]
                   ,[BaseDatosID]
                   ,[VersionID])
SELECT
        15 as SmoID,
        OBJECT_NAME(sm.object_id) as NombreVista,
        '',
```

```
        ver.fix_view_create(OBJECT_NAME(sm.object_id),
           sm.definition),
        'ver.VN␣'''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_view_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''','''''')+'''',
        'ver.VD␣'''+OBJECT_NAME(sm.object_id)+'''',
        @DB_id,
        @Version
FROM sys.sql_modules AS sm
JOIN sys.objects AS o ON sm.object_id = o.object_id
LEFT JOIN [Metadatos1.4].DBO.Vista mv on  mv.
   NombreVista = OBJECT_NAME(sm.object_id) and mv.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Vista metav where metav.NombreVista = mv.
   NombreVista)
where o.type='V'
and SCHEMA_NAME(o.schema_id) != 'ver'
and (mv.VistaID is null OR mv.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))
order by modify_date asc;

INSERT INTO [Metadatos1.4].[dbo].[DependenciaSmoVista]
   (VistaId, Dependencia)
select VistaID, CASE WHEN referenced_schema_name IS NOT
    NULL THEN referenced_schema_name+'.'ELSE '' END +
   referenced_entity_name
```

```
from [Metadatos1.4].[dbo].[Vista]
INNER JOIN sys.sql_expression_dependencies AS sed on
   sed.referencing_id = OBJECT_ID(NombreVista)
WHERE [BaseDatosID]=@DB_id AND [VersionID]=@Version

-- 17   Cambio nombre de una vista
-- (por el momento NO APLICA)

-- 18   Cambio contenido de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                    ([SmoID]
                    ,[NombreVista]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        15 as SmoID,
        OBJECT_NAME(sm.object_id) as NombreVista,
        mv.ScriptNuevo as ScriptOriginal,
        ver.fix_view_create(OBJECT_NAME(sm.object_id),
           sm.definition) as ScriptNuevo,
        'ver.VM ''+OBJECT_NAME(sm.object_id)+''',''+
           REPLACE(ver.fix_view_create(OBJECT_NAME(sm.
```

```
                object_id),sm.definition),'''',''''''')+'''',
            'ver.VM␣'''+OBJECT_NAME(sm.object_id)+''','''+
                REPLACE(mv.ScriptNuevo,'''',''''''')+'''',
            @DB_id,
            @Version
    FROM sys.sql_modules AS sm
    JOIN sys.objects AS o ON sm.object_id = o.object_id
    LEFT JOIN [Metadatos1.4].DBO.Vista mv on  mv.
       NombreVista = OBJECT_NAME(sm.object_id) and mv.
       versionID = (select MAX(VersionID) from [Metadatos1
       .4].dbo.Vista metav where metav.NombreVista = mv.
       NombreVista)
    where o.type='V'
    and SCHEMA_NAME(o.schema_id) != 'ver'
    and mv.VistaID is not null
    AND mv.SmoID in
       (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
    and ver.fix_view_create(OBJECT_NAME(sm.object_id),sm.
       definition)!=mv.ScriptNuevo;

    select '[VersionaVista]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionaTabla]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaTabla]
( @Version int,
  @DB_id int    )
AS
BEGIN
Declare @schema_id int
select @schema_id = schema_id from sys.schemas where name ='ver
   '


------------13  Eliminacion de tabla
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 13, mt.NombreTabla, 'ver.UD␣'+mt.
           NombreTabla+'␣', 'ver.UN␣'+mt.NombreTabla+'␣',@DB_id,
           @Version
        from [Metadatos1.4].dbo.Tabla mt
        left join sys.objects t on t.object_id = mt.object_id
        where mt.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        AND mt.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
```

```
        AND t.object_id is null


------------1   Creación de tabla
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 1, t.name, 'ver.UN␣'+t.name+'␣', '
           ver.UD␣'+t.name+'␣',@DB_id,@Version
        from sys.objects t
        LEFT JOIN
        [Metadatos1.4].dbo.Tabla mt on t.object_id = mt.
           object_id and mt.versionID = (select MAX(VersionID)
           from [Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        where t.type ='U' and t.is_ms_shipped = 0 and t.name !=
            'sysdiagrams'
        AND (mt.vTablaID is null OR mt.SmoID not in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))
        and t.schema_id != @schema_id
------------LAS TABLAS QUE FUERON RENOMBRADAS
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 2,t.name,'ver.UR␣'+mt.NombreTabla+'
           ,' + t.name,  'ver.UR␣'+t.name+',' + mt.NombreTabla,
           @DB_id,@Version
```

```
        from sys.objects t
        LEFT JOIN
        [Metadatos1.4].dbo.Tabla mt on t.object_id = mt.
           object_id and mt.versionID = (select MAX(VersionID)
           from [Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        where t.type ='U' and t.is_ms_shipped = 0 and t.name !=
            'sysdiagrams'
        AND mt.object_id IS NOT NULL and t.name!=mt.NombreTabla
            and t.schema_id != @schema_id
        AND mt.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        END
GO
/****** Object:  StoredProcedure [ver].[VersionaRutina]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaRutina]
( @Version int,
  @DB_id int    )
AS
BEGIN
```

```
--Bug MsSQL 2008, no se puede llamar a
   sp_refreshsqlmodule luego de un sp_rename
--http://connect.microsoft.com/SQLServer/feedback/
   details/644572/sp-refreshsqlmodule-changes-object-
   definition
--declare @sp_name varchar(max)
--DECLARE sp_cursor CURSOR FOR
--      SELECT o.name
--      FROM sys.objects AS o
--      WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS
   ', N'FT')
--      and SCHEMA_NAME(o.schema_id)!='ver'
--      and LEFT(o.name,3) not in ('fn_','sp_')
--OPEN sp_cursor
--FETCH NEXT FROM sp_cursor INTO @sp_name
--WHILE @@FETCH_STATUS = 0
--BEGIN
--      EXEC sys.sp_refreshsqlmodule @sp_name
--      FETCH NEXT FROM sp_cursor INTO @sp_name
--END
--CLOSE sp_cursor
--DEALLOCATE sp_cursor

-- 20   Eliminacion de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                    ([SmoID]
```

```
                    ,[TipoRutinaID]
                    ,[NombreRutina]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        20 as SmoID,
        mr.TipoRutinaID,
        mr.NombreRutina, -- deleted routine name
        mr.ScriptNuevo,
        '',
        'ver.RD␣'''+mr.NombreRutina+'''',
        'ver.RN␣'''+mr.NombreRutina+''','''+REPLACE(mr.
           ScriptNuevo,'''','''''')+'''',
        @DB_id,
        @Version
FROM [Metadatos1.4].DBO.Rutina mr
LEFT JOIN sys.objects AS o ON mr.NombreRutina =
   OBJECT_NAME(o.object_id)
WHERE
mr.versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Rutina metar where metar.NombreRutina = mr.
   NombreRutina)
```

```
AND mr.SmoID in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
AND o.object_id is null;

-- 19   Creacion de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                    ([SmoID]
                    ,[TipoRutinaID]
                    ,[NombreRutina]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        19 as SmoID,
        (CASE o.type WHEN 'P' THEN 2 ELSE 1 END) as
           TipoRutinaID,
        o.name,
        '',
        ver.fix_rutina_create(OBJECT_NAME(sm.object_id)
           ,sm.definition),
        'ver.RN␣'''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_rutina_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''','''''')+'''',
```

```
        'ver.RD ''''+OBJECT_NAME(sm.object_id)+'''',
        @DB_id,
        @Version
FROM sys.objects AS o
JOIN sys.sql_modules AS sm ON sm.object_id = o.
   object_id
LEFT JOIN [Metadatos1.4].DBO.Rutina mr on  mr.
   NombreRutina = OBJECT_NAME(sm.object_id) and mr.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Rutina metar where metar.NombreRutina = mr.
   NombreRutina)
WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS', N'FT')
and SCHEMA_NAME(o.schema_id)!='ver'
and LEFT(o.name,3) not in ('fn_','sp_')
and (mr.RutinaID is null OR mr.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))
order by modify_date asc;

-- 21   Cambio nombre de una Rutina
-- (por el momento NO APLICA)

-- 22   Cambio contenido de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                    ([SmoID]
                    ,[TipoRutinaID]
                    ,[NombreRutina]
```

```
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        22 as SmoID,
        (CASE o.type WHEN 'P' THEN 2 ELSE 1 END) as
           TipoRutinaID,
        o.name,
        mr.ScriptNuevo as ScriptOriginal,
        ver.fix_rutina_create(OBJECT_NAME(sm.object_id)
           ,sm.definition) as ScriptNuevo,
        'ver.RM '''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_rutina_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''',''''')+'''',
        'ver.RM '''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(mr.ScriptNuevo,'''',''''')+'''',
        @DB_id,
        @Version
FROM sys.objects AS o
JOIN sys.sql_modules AS sm ON sm.object_id = o.
   object_id
LEFT JOIN [Metadatos1.4].DBO.Rutina mr on  mr.
   NombreRutina = OBJECT_NAME(sm.object_id) and mr.
```

```
            versionID = (select MAX(VersionID) from [Metadatos1
            .4].dbo.Rutina metar where metar.NombreRutina = mr.
            NombreRutina)
         WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS', N'FT')
         and SCHEMA_NAME(o.schema_id)!='ver'
         and LEFT(o.name,3) not in ('fn_','sp_')
         and mr.RutinaID is not null
         AND mr.SmoID in
            (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
         and ver.fix_rutina_create(OBJECT_NAME(sm.object_id),sm.
            definition)!=mr.ScriptNuevo;

         select '[VersionaRutina]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionarDB]    Script
   Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionarDB]
( @DB varchar(100) )
AS
BEGIN
         DECLARE @Version INT--- OBTENER LA SIGUIENTE VERSION
```

```
        INSERT INTO [Metadatos1.4].dbo.MetadatoVersion (Fecha)
           VALUES (GETDATE())
        SELECT @Version = @@IDENTITY --1

        DECLARE @DB_id int --- OBTENER EL ID DE LA BASE DE
           DATOS QUE SE VA A VERSIONAR
        SELECT @DB_id = BaseDatosID FROM [Metadatos1.4].DBO.DB

        EXEC ver.VersionaTabla @Version, @DB_id
        exec ver.VersionaColumna @Version, @DB_id
        exec ver.VersionaPrimaria @Version, @DB_id
        exec ver.VersionaForanea @Version, @DB_id
        exec ver.VersionaVista @Version, @DB_id
        exec ver.VersionaRutina @Version, @DB_id

END
GO
```

B.3. ScriptsDeployment.sql

Script B.3: Base de datos 'base' para producción

```
/* SCRIPTS DE DEPLOYMENT, INSTALAR EN PRODUCCION */
USE [DB_PRODUCION]
GO
```

```
/****** Object:  Schema [ver]    Script Date: 08/29/2013
   23:08:27 ******/
CREATE SCHEMA [ver] AUTHORIZATION [dbo]
GO
/****** Object:  StoredProcedure [ver].[PKD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO


CREATE PROCEDURE [ver].[PKD]
(
 @TablaNombre varchar(50),
 @NombrePrimaria varchar(50),
 @vid integer
)
AS
BEGIN
        declare @sql varchar(max)
        set @sql = 'IF  EXISTS (SELECT * FROM sys.indexes WHERE
           object_id = OBJECT_ID(N''[dbo].['+@TablaNombre+']'')
           AND name = N'''+@NombrePrimaria+''') ';
        set @sql = @sql + 'ALTER TABLE [dbo].['+@TablaNombre+']
           DROP CONSTRAINT ['+@NombrePrimaria+'];';
        select @sql;
```

```
        exec (@sql);


END
GO
/****** Object:  StoredProcedure [ver].[FKR]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[FKR]
(
 @TablaNombre varchar(50),
 @OldNombreForanea varchar(50),
 @NewNombreForanea varchar(50),
 @vid integer
)
AS
BEGIN
        declare @sql varchar(max)
        set @sql = 'sp_rename ''dbo.'/*+@TablaNombre+'.'*/+
           @OldNombreForanea+''',''+@NewNombreForanea+''''
        select @sql;
        exec (@sql);


END
```

```
GO
/****** Object:  StoredProcedure [ver].[FKD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[FKD]
(
 @TablaNombre varchar(50),
 @NombreForanea varchar(50),
 @vid integer
)
AS
BEGIN
        declare @sql varchar(max)

        set @sql = 'IF  EXISTS (SELECT * FROM sys.foreign_keys
           WHERE object_id = OBJECT_ID(N''[dbo].['+
           @NombreForanea+']'') AND parent_object_id = OBJECT_ID
           (N''[dbo].['+@TablaNombre+']''))'
        set @sql = @sql + 'ALTER TABLE [dbo].['+@TablaNombre+']
           DROP CONSTRAINT ['+@NombreForanea+'];';
        select @sql;
        exec (@sql);
```

```
END
GO
/****** Object:  StoredProcedure [ver].[CR]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[CR]
(
        @Tabla varchar(50),
        @ColumnaOriginal varchar(50),
        @ColumnaNueva varchar(50),
        @vid integer
)
as
BEGIN
        declare @variable varchar(200)
        set @variable = 'sp_rename ''dbo.'+ @Tabla+'.' +
           @ColumnaOriginal+''','''+ @ColumnaNueva +''', ''
           COLUMN'''
        exec(@variable)
        --EXEC ()
END
GO
```

```
/****** Object:  StoredProcedure [ver].[CN]    Script Date:
   11/13/2013 19:19:43 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [ver].[CN]
(
 @TablaNombre varchar(50),
 @ColumnaNombre varchar(50),
 @Tipo varchar(50),
 @null varchar(50) = null,
 @default varchar(50) = null,
 @vid integer
)
AS
BEGIN
                IF @null = 'YES' BEGIN
                        SET @null = 'NULL'
                END
                ELSE
                BEGIN
                        SET @null = 'NOT␣NULL'
                END
```

```
                IF @default IS NOT NULL BEGIN
                        SET @default = 'DEFAULT␣'+ @default
                END
                EXEC('ALTER␣TABLE␣'+@TablaNombre+'␣ADD␣'+
                   @ColumnaNombre+'␣' +@Tipo+'␣'+@null+'␣'+
                   @default)
                -- delete column Dummy
                IF @ColumnaNombre!='DUMMY_COLUMN' AND EXISTS (
                  SELECT *
                  FROM   sys.columns
                  WHERE  object_id = OBJECT_ID(@TablaNombre)
                                AND name = 'DUMMY_COLUMN')
                BEGIN
                        exec('ALTER␣TABLE␣'+ @TablaNombre+'␣
                           DROP␣COLUMN␣DUMMY_COLUMN');
                END
END

GO
/****** Object:  StoredProcedure [ver].[CD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[CD]
```

```
(
 @TablaNombre varchar(50),
 @ColumnaNombre varchar(50),
 @vid integer
)
AS
BEGIN
        IF (SELECT COUNT(COLUMN_NAME)FROM INFORMATION_SCHEMA.
           COLUMNS where TABLE_NAME = @TablaNombre) =1
        BEGIN
                EXEC('␣ALTER␣TABLE␣' + @TablaNombre     +'␣ADD␣
                   DUMMY_COLUMN␣INT')
        END
        EXEC ('ALTER␣TABLE␣' + @TablaNombre +'␣DROP␣COLUMN␣'+
           @ColumnaNombre )
END
GO
/****** Object:  StoredProcedure [ver].[VN]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VN]
(
 @NombreVista varchar(50),
```

```
 @Script varchar(MAX),
 @vid integer
)
AS
BEGIN
        exec (@Script)
END
GO
/****** Object:  StoredProcedure [ver].[VM]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VM]
(
 @NombreVista varchar(50),
 @Script varchar(MAX),
 @vid integer
)
AS
BEGIN
        declare @smoD  as varchar(max)
        declare @smoN  as varchar(max)
        select @Script = REPLACE(@Script,'''','''''')
```

```
        set @smoD='[ver].[VD] '''+@NombreVista+''','+CONVERT(
           varchar(10),@vid)
        set @smoN='[ver].[VN] '''+@NombreVista+''','''+@Script+
           ''','+CONVERT(varchar(10),@vid)
        exec (@smoD)
        exec (@smoN)
END
GO
/****** Object:  Table [dbo].[Version]    Script Date:
   08/29/2013 23:08:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Version](
        [VersionID] [int] IDENTITY(1,1) NOT NULL,
        [Fecha] [date] NULL,
        [VersionDB] [float] NULL,
        [VersionApp] [float] NULL,
        [VersionServicio] [float] NULL,
        [VersionMapping] [int] NULL,
 CONSTRAINT [PK_VERSION_EB4AE92811F148CBB1C6FCE4283E424E]
    PRIMARY KEY NONCLUSTERED
(
        [VersionID] ASC
```

```
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS
     = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  StoredProcedure [ver].[VD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VD]
(
 @NombreVista varchar(50),
 @vid integer
)
AS
BEGIN
        exec ('DROP␣VIEW␣['+@NombreVista+']')
END
GO
/****** Object:  StoredProcedure [ver].[UR]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
CREATE PROCEDURE [ver].[UR]
(
        @NombreOriginal varchar(50),
        @NuevoNombre    varchar(50),
        @vid integer
)
AS
BEGIN
declare @variable varchar(200)
        set @variable = 'sp_rename ''dbo.'+ @NombreOriginal+'''
           ,'''+ @NuevoNombre +''''
        exec(@variable)
END
GO
/****** Object:  StoredProcedure [ver].[UN]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[UN]
(
        @TablaNombre varchar(50),
        @vid integer
)
```

```
AS
BEGIN

IF NOT EXISTS( SELECT * FROM INFORMATION_SCHEMA.TABLES where
   TABLE_NAME = @TablaNombre)
BEGIN
        EXEC('CREATE␣TABLE␣' + @TablaNombre +'(␣DUMMY_COLUMN␣
           int)')
END

END
GO
/****** Object:  StoredProcedure [ver].[UD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[UD]
(
        @TablaNombre varchar(50),
        @vid integer
)
AS
BEGIN
```

```
        EXEC ('DROP TABLE '+ @TablaNombre)
END
GO
/****** Object:  UserDefinedFunction [ver].[
   SplitCsvIntoIDsTempTable]    Script Date: 08/29/2013 23:08:27
    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[SplitCsvIntoIDsTempTable](@a varchar
   (8000))
returns @temptable TABLE (items integer)
as
begin
        set @a=@a+',';
        while len(@a) > 1
        begin
        insert into @temptable
        select substring(@a,1,patindex('%,%',@a)-1);
        set @a = substring(@a,patindex('%,%',@a)+1,len(@a))
        end;
        return
end
GO
```

```
/****** Object:  StoredProcedure [ver].[RN]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[RN]
(
 @NombreRutina varchar(50),
 @Script varchar(MAX),
 @vid integer
)
AS
BEGIN
        exec (@Script)
END
GO
/****** Object:  StoredProcedure [ver].[RM]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[RM]
(
 @NombreRutina varchar(50),
```

```
 @Script varchar(MAX),
 @vid integer
)
AS
BEGIN
        declare @smoD  as varchar(max)
        declare @smoN  as varchar(max)
        select @Script = REPLACE(@Script,'''',''''''')
        set @smoD='[ver].[RD] '''+@NombreRutina+''',CONVERT(
           varchar(10),@vid)
        set @smoN='[ver].[RN] '''+@NombreRutina+''','''+@Script
           +''',CONVERT(varchar(10),@vid)
        exec (@smoD)
        exec (@smoN)
END
GO
/****** Object:  StoredProcedure [ver].[RD]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[RD]
(
 @NombreRutina varchar(50),
 @vid integer
```

```
)
AS
BEGIN
        IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
            OBJECT_ID(N'['+@NombreRutina+']') AND type in (N'FN'
           , N'IF', N'TF', N'FS', N'FT')) BEGIN
                exec ('DROP␣FUNCION␣['+@NombreRutina+']')
        END
        IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
            OBJECT_ID(N'['+@NombreRutina+']') AND type in (N'P')
          ) BEGIN
                exec ('DROP␣PROCEDURE␣['+@NombreRutina+']')
        END
END
GO
/****** Object:  StoredProcedure [ver].[PKR]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[PKR]
(
 @TablaNombre varchar(50),
 @OldNombrePrimaria varchar(50),
 @NewNombrePrimaria varchar(50),
```

```
 @vid integer
)
AS
BEGIN
        declare @sql varchar(max)
        set @sql = 'sp_rename ''dbo.'/*+@TablaNombre+'.'*/+
           @OldNombrePrimaria+''','''+@NewNombrePrimaria+''''
        select @sql;
        exec (@sql);

END
GO
/****** Object:  StoredProcedure [ver].[PKN]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[PKN]
(
 @TablaNombre varchar(50),
 @NombrePrimaria varchar(50),
 @csv_object_id_Columna varchar(50),
 @vid integer
)
AS
```

```
BEGIN
        declare @temptable_object_id_Columna TABLE (object_id
           integer);
        declare @sql varchar(max)
        declare @column_object_id integer;
        declare @column_number integer;
        declare @column_name varchar(50);

        insert into @temptable_object_id_Columna select * from
           ver.SplitCsvIntoIDsTempTable(@csv_object_id_Columna);

        set @sql = 'ALTER␣TABLE␣'+@TablaNombre+'␣';
        set @sql = @sql + '␣ADD␣␣CONSTRAINT␣['+@NombrePrimaria+
           ']␣PRIMARY␣KEY␣CLUSTERED␣(␣';

        DECLARE cursor_columnas CURSOR FOR select * from
           @temptable_object_id_Columna
        OPEN cursor_columnas
        FETCH cursor_columnas INTO @column_object_id
        set @column_number=0;
        WHILE (@@FETCH_STATUS = 0)
        BEGIN
                set @column_number = @column_number+1;
                set @column_name='';

                select @column_name=mc.NombreColumna
```

```
from [Metadatos1.4].dbo
   .Columna mc
inner join [Metadatos1
   .4].dbo.Tabla mtc on
   mc.vTablaID=mtc.
   vTablaID
inner join [Metadatos1
   .4].dbo.Tabla mt on
   mt.object_id=mtc.
   object_id and mt.
   versionID = (select
   MAX(VersionID) from [
   Metadatos1.4].dbo.
   Tabla metat where
   VersionID<=@vid and
   metat.object_id = mt.
   object_id)
where
mc.object_id=
   @column_object_id
and mt.NombreTabla=
   @TablaNombre
and mc.versionID = (
   select MAX(VersionID)
    from [Metadatos1.4].
   dbo.Columna metacol
```

```
                                        where VersionID<=@vid
                                         and metacol.
                                        object_id = mc.
                                        object_id and metacol
                                        .vTablaID = mc.
                                        vTablaID)

        if (@column_name!='') begin
                if (@column_number>1) begin
                        set @sql=@sql+','
                end
                set @sql = @sql + ' [' +@column_name+'] '
                   ASC ';
        end

        FETCH cursor_columnas INTO @column_object_id
END
CLOSE cursor_columnas
DEALLOCATE cursor_columnas

set @sql = @sql + ' )WITH (PAD_INDEX  = OFF, '
   STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF, '
   IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  '
   = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY] ';

select @sql;
```

```
        exec(@sql);
END
GO

/****** Object:  StoredProcedure [ver].[ActualizarDB]    Script
    Date: 11/13/2013 21:03:22 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [ver].[ActualizarDB]
(
   @ALaVersion INT,
   @Ejecuta bit=0
)
AS
BEGIN

        DECLARE @DeLaVersion INT
        DECLARE @smo varchar(max)
        DECLARE @step integer
        DECLARE @vid integer
        DECLARE @smo_id integer

        SET @DeLaVersion = 0;
```

```
SELECT @DeLaVersion=v.VersionDB from dbo.Version v
   where v.VersionID=(select MAX(VersionID) from dbo.
   Version);

IF @DeLaVersion < @ALaVersion
BEGIN
        DECLARE Cursor_Actualiza CURSOR
        FOR
        SELECT ReglaActualizacion, 1 as step, VersionID
            as vid, 0  from [Metadatos1.4].dbo.Tabla
           WHERE VersionID > @DeLaVersion and VersionID
           <= @ALaVersion
        UNION ALL
        SELECT ReglaActualizacion, 2 as step, VersionID
            as vid, 0  from [Metadatos1.4].dbo.Columna
           WHERE VersionID > @DeLaVersion and VersionID
           <= @ALaVersion
        UNION ALL
        SELECT ReglaActualizacion, 3 as step, VersionID
            as vid, 0  from [Metadatos1.4].dbo.Primaria
           WHERE VersionID > @DeLaVersion and VersionID
           <= @ALaVersion
        UNION ALL
        SELECT ReglaActualizacion, 4 as step, VersionID
            as vid, 0  from [Metadatos1.4].dbo.Foranea
           WHERE VersionID > @DeLaVersion and VersionID
```

```
                <= @ALaVersion
            UNION ALL
            SELECT ReglaActualizacion, 5 as step, VersionID
                as vid, VistaID  from [Metadatos1.4].dbo.
                Vista WHERE VersionID > @DeLaVersion and
                VersionID <= @ALaVersion
            UNION ALL
            SELECT ReglaActualizacion, 6 as step, VersionID
                as vid, 0  from [Metadatos1.4].dbo.Rutina
                WHERE VersionID > @DeLaVersion and VersionID
                <= @ALaVersion
            ORDER BY vid asc, step asc
    END
    IF @DeLaVersion > @ALaVersion BEGIN
            DECLARE Cursor_Actualiza CURSOR
            FOR
            SELECT ReglaEliminacion as smo, 1 as step,
                VersionID as vid, 0 from [Metadatos1.4].dbo.
                Rutina WHERE VersionID <= @DeLaVersion and
                VersionID > @ALaVersion
            UNION ALL
            SELECT ReglaEliminacion as smo, 2 as step,
                VersionID as vid, VistaID from [Metadatos1
                .4].dbo.Vista WHERE VersionID <= @DeLaVersion
                and VersionID > @ALaVersion
            UNION ALL
```

```
            SELECT Regla_Eliminacion as smo, 3 as step,
               VersionID as vid, 0 from [Metadatos1.4].dbo.
               Foranea WHERE VersionID <= @DeLaVersion and
               VersionID > @ALaVersion
            UNION ALL
            SELECT Regla_Eliminacion as smo, 4 as step,
               VersionID as vid, 0 from [Metadatos1.4].dbo.
               Primaria WHERE VersionID <= @DeLaVersion and
               VersionID > @ALaVersion
            UNION ALL
            SELECT Regla_Eliminacion as smo, 5 as step,
               VersionID as vid, 0 from [Metadatos1.4].dbo.
               Columna WHERE VersionID <= @DeLaVersion and
               VersionID > @ALaVersion
            UNION ALL
            SELECT Regla_Eliminacion as smo, 6 as step,
               VersionID as vid, 0 from [Metadatos1.4].dbo.
               Tabla WHERE VersionID <= @DeLaVersion and
               VersionID > @ALaVersion
            ORDER BY vid desc, step asc
      END

      SET XACT_ABORT ON;
      BEGIN TRANSACTION;
      DECLARE @VistasPendientes as Table(smo varchar(max),
         terminado bit, smo_id integer)
```

```
DECLARE @CantidadDependenciasFaltantes as integer
IF @DeLaVersion != @ALaVersion BEGIN
        OPEN Cursor_Actualiza
        FETCH Cursor_Actualiza INTO @smo, @step, @vid,
           @smo_id
        WHILE (@@FETCH_STATUS = 0)
        BEGIN
                set @smo = ''+@smo+','+CONVERT(varchar
                   (10),@vid)
                PRINT (@smo)
                if @Ejecuta=1 begin
                        IF (@DeLaVersion < @ALaVersion
                           AND @step=5)
                                OR (@DeLaVersion >
                                   @ALaVersion AND @step
                                   =2) BEGIN
                                SELECT
                                   @CantidadDependenciasFaltantes
                                   =COUNT(1)
                                FROM [Metadatos1.4].dbo
                                   .DependenciaSmoVista
                                LEFT JOIN sys.objects
                                   ON object_id =
                                   OBJECT_ID(Dependencia
                                   )
```

```
                        WHERE VistaID=@smo_id
                           AND object_id is null
                        IF
                           @CantidadDependenciasFaltantes
                           >0 BEGIN
                                 INSERT INTO
                                    @VistasPendientes
                                     (smo,
                                    terminado,
                                    smo_id)
                                    values (@smo,
                                     0, @smo_id)
                        END ELSE BEGIN
                                 EXEC (@smo)
                        END
                        PRINT 'FALTANTES:␣'+
                           CAST(
                           @CantidadDependenciasFaltantes
                            as varchar(max))+'␣'
                           +@smo
                  END ELSE BEGIN
                           EXEC (@smo)
                  END
            end
            FETCH Cursor_Actualiza INTO @smo, @step
               , @vid, @smo_id
```

```
END
CLOSE Cursor_Actualiza
DEALLOCATE Cursor_Actualiza

if @Ejecuta=1 begin
        DECLARE @CantidadIntentosRestantes as
           integer
        SELECT @CantidadIntentosRestantes=COUNT
           (1) FROM @VistasPendientes
        WHILE @CantidadIntentosRestantes>0
           BEGIN
                SET @CantidadIntentosRestantes=
                    @CantidadIntentosRestantes-1
                DECLARE
                   Cursor_ActualizaVistasPendientes
                    CURSOR FOR SELECT smo,
                   smo_id FROM @VistasPendientes
                    where terminado=0
                OPEN
                   Cursor_ActualizaVistasPendientes

                FETCH
                   Cursor_ActualizaVistasPendientes
                    INTO @smo, @smo_id
                WHILE (@@FETCH_STATUS = 0)
                BEGIN
```

```
                    SELECT
                       @CantidadDependenciasFaltantes
                       =COUNT(1)
                    FROM [Metadatos1.4].dbo
                       .DependenciaSmoVista
                    LEFT JOIN sys.objects
                       ON object_id =
                       OBJECT_ID(Dependencia
                       )
                    WHERE VistaID=@smo_id
                       AND object_id is null
                    IF
                       @CantidadDependenciasFaltantes
                       =0 BEGIN
                            EXEC (@smo)
                            UPDATE
                               @VistasPendientes
                                SET
                               terminado=1
                               WHERE smo_id=
                               @smo_id
                    END
                    FETCH
                       Cursor_ActualizaVistasPendiente
                        INTO @smo, @smo_id
                END
```

```
                    CLOSE
                       Cursor_ActualizaVistasPendientes

                    DEALLOCATE
                       Cursor_ActualizaVistasPendientes

            END
        end

        if @Ejecuta=1 begin
            IF @DeLaVersion < @ALaVersion BEGIN
                INSERT INTO [dbo].[Version]
                           ([Fecha]
                           ,[VersionDB]
                           ,[VersionApp]
                           ,[VersionServicio]
                           ,[VersionMapping])
                     VALUES
                           (GETDATE()
                           ,@vid
                           ,null
                           ,null
                           ,null)
            end
            IF @DeLaVersion > @ALaVersion BEGIN
                INSERT INTO [dbo].[Version]
```

```
                                                ([Fecha]
                                                ,[VersionDB]
                                                ,[VersionApp]
                                                ,[VersionServicio]
                                                ,[VersionMapping])
                                        VALUES
                                                (GETDATE()
                                                ,@ALaVersion
                                                ,null
                                                ,null
                                                ,null)
                                end
                        end

        END

        COMMIT TRANSACTION;

END
GO
/****** Object:  StoredProcedure [ver].[FKN]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE PROCEDURE [ver].[FKN]
(
 @TablaPadreNombre varchar(50),
 @TablaReferenciadaNombre varchar(50),
 @NombreForanea varchar(50),
 @csv_object_id_ColumnaOrigen varchar(50),
 @csv_object_id_ColumnaReferenciada varchar(50),
 @vid integer
)
AS
BEGIN
        declare @temptable_object_id_ColumnaOrigen TABLE (
           object_id integer);
        declare @temptable_object_id_ColumnaReferenciada TABLE
           (object_id integer);
        declare @sql varchar(max)
        declare @column_object_id integer;
        declare @column_number integer;
        declare @column_name varchar(50);

        insert into @temptable_object_id_ColumnaOrigen select *
            from ver.SplitCsvIntoIDsTempTable(
           @csv_object_id_ColumnaOrigen);
        insert into @temptable_object_id_ColumnaReferenciada
           select * from ver.SplitCsvIntoIDsTempTable(
           @csv_object_id_ColumnaReferenciada);
```

```
set @sql = 'ALTER TABLE ['+@TablaPadreNombre+'] WITH
   CHECK ADD CONSTRAINT ['+@NombreForanea+'] FOREIGN
   KEY( '

DECLARE cursor_columnas CURSOR FOR select * from
   @temptable_object_id_ColumnaOrigen
OPEN cursor_columnas
FETCH cursor_columnas INTO @column_object_id
set @column_number=0;
WHILE (@@FETCH_STATUS = 0)
BEGIN
        set @column_number = @column_number+1;
        set @column_name='';

        select @column_name=mc.NombreColumna
                from [Metadatos1.4].dbo.Columna mc
                inner join [Metadatos1.4].dbo.Tabla mtc
                    on mc.vTablaID=mtc.vTablaID
                inner join [Metadatos1.4].dbo.Tabla mt
                   on mt.object_id=mtc.object_id and mt.
                   versionID = (select MAX(VersionID)
                   from [Metadatos1.4].dbo.Tabla metat
                   where VersionID<=@vid and metat.
                   object_id = mt.object_id)
                where
```

```
                mc.object_id=@column_object_id
                and mt.NombreTabla=@TablaPadreNombre
                and mc.versionID = (select MAX(
                   VersionID) from [Metadatos1.4].dbo.
                   Columna metacol where VersionID<=@vid
                    and metacol.object_id = mc.object_id
                    and metacol.vTablaID = mc.vTablaID)

        if (@column_name!='') begin
                if (@column_number>1) begin
                        set @sql=@sql+','
                end
                set @sql = @sql + ' [' +@column_name+'] '
                   ';
        end
        FETCH cursor_columnas INTO @column_object_id
END
CLOSE cursor_columnas
DEALLOCATE cursor_columnas

set @sql=@sql+' )REFERENCES [' +@TablaReferenciadaNombre
   +'] ( '

DECLARE cursor_columnas CURSOR FOR select * from
   @temptable_object_id_ColumnaReferenciada
OPEN cursor_columnas
```

```
FETCH cursor_columnas INTO @column_object_id
set @column_number=0;
WHILE (@@FETCH_STATUS = 0)
BEGIN
        set @column_number = @column_number+1;
        set @column_name='';
        select @column_name=mc.NombreColumna
                from [Metadatos1.4].dbo.Columna mc
                inner join [Metadatos1.4].dbo.Tabla mtc
                    on mc.vTablaID=mtc.vTablaID
                inner join [Metadatos1.4].dbo.Tabla mt
                   on mt.object_id=mtc.object_id and mt.
                   versionID = (select MAX(VersionID)
                   from [Metadatos1.4].dbo.Tabla metat
                   where VersionID<=@vid and metat.
                   object_id = mt.object_id)
                where
                mc.object_id=@column_object_id
                and mt.NombreTabla=
                   @TablaReferenciadaNombre
                and mc.versionID = (select MAX(
                   VersionID) from [Metadatos1.4].dbo.
                   Columna metacol where VersionID<=@vid
                    and metacol.object_id = mc.object_id
                    and metacol.vTablaID = mc.vTablaID)
```

```
                if (@column_name!='') begin
                        if (@column_number>1) begin
                                set @sql=@sql+','
                        end
                        set @sql = @sql + ' [' +@column_name+'] '
                           ';
                end
                FETCH cursor_columnas INTO @column_object_id
        END
        CLOSE cursor_columnas
        DEALLOCATE cursor_columnas

        set @sql=@sql+' ) '

        select @sql;
        exec (@sql)
END
GO
/****** Object:  StoredProcedure [ver].[FKM]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[FKM]
(
```

```
 @TablaPadreNombre varchar(50),
 @TablaReferenciadaNombre varchar(50),
 @NombreForanea varchar(50),
 @csv_object_id_ColumnaOrigen varchar(50),
 @csv_object_id_ColumnaReferenciada varchar(50),
 @vid integer
)
AS
BEGIN
        -- To modify a FOREIGN KEY constraint, you must first
           delete the existing FOREIGN KEY constraint and then
           re-create it with the new definition.
        -- http://msdn.microsoft.com/en-us/library/ms177463(v=
           sql.105).aspx
        exec ver.FKD @TablaPadreNombre, @NombreForanea, @vid
        exec ver.FKN @TablaPadreNombre,
           @TablaReferenciadaNombre, @NombreForanea,
           @csv_object_id_ColumnaOrigen,
           @csv_object_id_ColumnaReferenciada, @vid
END
GO
/****** Object:  StoredProcedure [ver].[PKM]    Script Date:
   08/29/2013 23:08:24 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
CREATE PROCEDURE [ver].[PKM]
(
 @TablaNombre varchar(50),
 @NombrePrimaria varchar(50),
 @csv_object_id_Columna varchar(50),
 @vid integer
)
AS
BEGIN
        -- To modify a PRIMARY KEY constraint, you must first
           delete the existing PRIMARY KEY constraint and then
           re-create it with the new definition.
        -- http://msdn.microsoft.com/en-us/library/ms181043(v=
           sql.105).aspx
        exec ver.PKD @TablaNombre, @NombrePrimaria, @vid
        exec ver.PKN @TablaNombre, @NombrePrimaria,
           @csv_object_id_Columna, @vid
END
GO
```

B.4. Script 06 01 CrearFuncion

Script B.4: Script para los escenarios de pruebas; Crear una función

```
USE DB_DESARROLLO
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION fnc_CalcularPrecioVenta
(
        @ProductoID float
)
RETURNS float
AS
BEGIN
        DECLARE @PrecioVenta FLOAT
        DECLARE @PrecioCompra FLOAT
        DECLARE @MaxPrecioVenta FLOAT
        DECLARE @MinPrecioVenta FLOAT

        SELECT @PrecioCompra = PrecioCompra, @MinPrecioVenta =
           MinPrecioVenta, @MaxPrecioVenta = MaxPrecioVenta
        FROM Producto WHERE ProductoID = @ProductoID

        IF @PrecioCompra *1.30 > @MaxPrecioVenta
        BEGIN
                SET @PrecioVenta = @MaxPrecioVenta
        END
        ELSE
```

```
        BEGIN
                IF @PrecioCompra *1.30 < @MinPrecioVenta
                BEGIN
                        SET @PrecioVenta = @MinPrecioVenta
                END
                ELSE
                        BEGIN
                        SET @PrecioVenta = @PrecioCompra * 1.30
                        END
        END

        RETURN @PrecioVenta

END
GO
```

B.5. ScriptBaseDesarrollo

Script B.5: Script Base para desarrollo

```
/* SCRIPTS DE VERSIONADOR, INSTALAR EN DESARROLLO */
USE [DB_DESARROLLO]
GO
/****** Object:  Schema [ver]    Script Date: 08/29/2013
   23:06:35 ******/
CREATE SCHEMA [ver] AUTHORIZATION [dbo]
```

```
GO
/****** Object:  UserDefinedFunction [ver].[fix_view_create]
      Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_view_create]
(
        @view_name varchar(max),
        @view_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
        declare @antes as varchar(max)
        declare @entre as varchar(max)
        declare @luego as varchar(max)
        declare @posicion_nombre as integer;

        set @a = @view_code
        set @antes=''
        set @entre=''
        set @luego=''
```

```
select @antes=substring(@a,1,patindex('%CREATE␣VIEW%',
   @a)+11)
select @luego=substring(@a,patindex('%CREATE␣VIEW%',@a)
   +11,LEN(@a))
select @luego=substring(@luego,patindex('%select␣%',
   @luego),LEN(@luego))
select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣VIEW',
    ''), @luego, '');

select @entre=REPLACE(@entre, '[', '*');
select @entre=REPLACE(@entre, ']', '*');
select @entre=REPLACE(@entre, '.', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '(', '*');
select @entre=REPLACE(@entre, 'as', '*');

set @posicion_nombre=0
select @posicion_nombre = patindex('%*'+@view_name+'*%'
   ,@entre)

set @ResultVar = @view_code
if @posicion_nombre=0
begin
```

```
                set @ResultVar=@antes+' '+ @view_name +' as '+
                   @luego
        end

        RETURN @ResultVar
END
GO
/****** Object:  UserDefinedFunction [ver].[fix_sp_create]
   Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_sp_create]
(
        @sp_name varchar(max),
        @sp_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
        declare @antes as varchar(max)
        declare @entre as varchar(max)
        declare @luego as varchar(max)
```

```
declare @posicion_nombre as integer;

set @a = @sp_code
set @antes=''
set @entre=''
set @luego=''

select @antes=substring(@a,1,patindex('%CREATE␣
   PROCEDURE%',@a)+16)
select @luego=substring(@a,patindex('%CREATE␣PROCEDURE%
   ',@a)+16,LEN(@a))
select @luego=substring(@luego,patindex('%(%',@luego),
   LEN(@luego))
select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣
   PROCEDURE', ''), @luego, '');

select @entre=REPLACE(@entre, '[', '*');
select @entre=REPLACE(@entre, ']', '*');
select @entre=REPLACE(@entre, '.', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '(', '*');

set @posicion_nombre=0
select @posicion_nombre = patindex('%*'+@sp_name+'*%',
   @entre)
```

```
        set @ResultVar = @sp_code
        if @posicion_nombre=0
        begin
                set @ResultVar=@antes+' '+ @sp_name +' '+@luego
        end

        RETURN @ResultVar
END
GO
/****** Object:  UserDefinedFunction [ver].[fix_fnc_create]
   Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_fnc_create]
(
        @fnc_name varchar(max),
        @fnc_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @a as varchar(max)
```

```
declare @antes as varchar(max)
declare @entre as varchar(max)
declare @luego as varchar(max)
declare @posicion_nombre as integer;

set @a = @fnc_code
set @antes=''
set @entre=''
set @luego=''

select @antes=substring(@a,1,patindex('%CREATE␣FUNCTION
   %',@a)+15)
select @luego=substring(@a,patindex('%CREATE␣FUNCTION%'
   ,@a)+15,LEN(@a))
select @luego=substring(@luego,patindex('%(%',@luego),
   LEN(@luego))
select @entre=REPLACE(REPLACE(@a, @antes+'CREATE␣
   FUNCTION', ''), @luego, '');

select @entre=REPLACE(@entre, '[', '*');
select @entre=REPLACE(@entre, ']', '*');
select @entre=REPLACE(@entre, '.', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '␣', '*');
select @entre=REPLACE(@entre, '(', '*');
```

```
        set @posicion_nombre=0
        select @posicion_nombre = patindex('%*'+@fnc_name+'*%',
           @entre)

        set @ResultVar = @fnc_code
        if @posicion_nombre=0
        begin
                set @ResultVar=@antes+' '+ @fnc_name +' '+
                   @luego
        end

        RETURN @ResultVar
END
GO
/****** Object:  StoredProcedure [ver].[VersionaPrimaria]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaPrimaria]
( @Version int,
  @DB_id INT
)
AS
BEGIN
```

```
--LAS PK QUE FUERON ELIMINADAS DEL ESQUEMA
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        11 as SmoID, -- "borra PK"
        mt.BaseDatosID,
        @Version,
        mp.object_id, -- deleted object id
        '',
        '',
        'ver.PKD ''+mt.NombreTabla+''',''+mp.NombrePrimaria+'
           ''' ,
        'ver.PKN ''+mt.NombreTabla+''',''+mp.NombrePrimaria+'
           '',''+mp.csv_object_id_Columna+''''
        from [Metadatos1.4].DBO.Primaria mp
```

```
        inner join [Metadatos1.4].dbo.Tabla mt on mt.vTablaID=
           mp.vTablaID and mt.versionID = (select MAX(VersionID)
            from [Metadatos1.4].dbo.Tabla metatv where metatv.
           object_id = mt.object_id)
        left join sys.key_constraints kc on kc.object_id=mp.
           object_id and kc.parent_object_id = mt.object_id and
           mt.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatv where metatv.object_id
            = mt.object_id)
        where kc.object_id is null
        and mp.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Primaria metapv where metapv.
           object_id = mp.object_id)
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);

--LAS PK QUE NO EXISTEN EN LOS METADATOS
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
```

```
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        5 as SmoID, -- "crea PK"
        mt.BaseDatosID,
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           ) as csv_object_id_Columna,
        'ver.PKN ''+mt.NombreTabla+''',''+kc.name+''',''+(
           SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           )+'''',
        'ver.PKD ''+mt.NombreTabla+''',''+kc.name+''''
        from sys.key_constraints kc
        inner join  sys.indexes i on kc.parent_object_id=i.
           object_id and kc.name=i.name
    inner join [Metadatos1.4].dbo.Tabla mt on kc.
       parent_object_id = mt.object_id and mt.versionID = (
       select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
```

```
       metatv where metatv.object_id = mt.object_id)
        LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
           object_id = kc.object_id and mp.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Primaria
           metapv where metapv.object_id = mp.object_id)
        where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(kc.schema_id) != 'ver'
        AND kc.type='PK'
        and (mp.PrimariaID is null OR mp.SmoID not in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));

--LAS PK QUE EXISTEN PERO CAMBIARON DE NOMBRE
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        6 as SmoID, -- "renombra PK"
        mt.BaseDatosID,
```

```
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
           object_id=i.object_id and ic.index_id=i.index_id
           ORDER BY ic.key_ordinal FOR XML PATH ('')), 1, 1, '')
           ) as csv_object_id_Columna,
        'ver.PKR␣'''+mt.NombreTabla+''','''+mp.NombrePrimaria+'
           '','''+kc.name+''''',
        'ver.PKR␣'''+mt.NombreTabla+''','''+kc.name+''','''+mp.
           NombrePrimaria+'''' --rollback
        from sys.key_constraints kc
        inner join  sys.indexes i on kc.parent_object_id=i.
           object_id and kc.name=i.name
    inner join [Metadatos1.4].dbo.Tabla mt on kc.
       parent_object_id = mt.object_id and mt.versionID = (
       select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
       metatv where metatv.object_id = mt.object_id)
        LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
           object_id = kc.object_id and mp.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Primaria
           metapv where metapv.object_id = mp.object_id)
        where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(kc.schema_id) != 'ver'
        AND kc.type='PK'
```

```
        and mp.object_id is not null
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        and mp.NombrePrimaria!=kc.name;

--LAS PK QUE EXISTEN PERO SE LE MODIFICARON SUS COLUMNAS (creo
   q nunca pasara esto, hace delete->create)
INSERT INTO [Metadatos1.4].[dbo].[Primaria]
           ([vTablaID]
           ,[SmoID]
           ,[BaseDatosID]
           ,[VersionID]
           ,[object_id]
           ,[NombrePrimaria]
           ,[csv_object_id_Columna]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        mt.vTablaID,
        7 as SmoID, -- "modify columns PK"
        mt.BaseDatosID,
        @Version,
        kc.object_id,
        kc.name,
        (SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
           varchar(45)) FROM sys.index_columns ic WHERE ic.
```

```
        object_id=i.object_id and ic.index_id=i.index_id
        ORDER BY ic.key_ordinal FOR XML PATH (''')), 1, 1, ''')
        ) as csv_object_id_Columna,
    'ver.PKM␣'''+mt.NombreTabla+''','''+kc.name+''','''+(
        SELECT STUFF((SELECT ',' + CAST(ic.column_id AS
        varchar(45)) FROM sys.index_columns ic WHERE ic.
        object_id=i.object_id and ic.index_id=i.index_id
        ORDER BY ic.key_ordinal FOR XML PATH (''')), 1, 1, ''')
        )+'''',
    'ver.PKM␣'''+mt.NombreTabla+''','''+kc.name+''','''+mp.
        csv_object_id_Columna+'''' -- rollback
    from sys.key_constraints kc
    inner join  sys.indexes i on kc.parent_object_id=i.
        object_id and kc.name=i.name
inner join [Metadatos1.4].dbo.Tabla mt on kc.
    parent_object_id = mt.object_id and mt.versionID = (
    select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
    metatv where metatv.object_id = mt.object_id)
     LEFT JOIN [Metadatos1.4].DBO.Primaria mp on  mp.
        object_id = kc.object_id and mp.versionID = (select
        MAX(VersionID) from [Metadatos1.4].dbo.Primaria
        metapv where metapv.object_id = mp.object_id)
     where OBJECT_NAME(kc.parent_object_id)!='sysdiagrams'
     and SCHEMA_NAME(kc.schema_id) != 'ver'
     AND kc.type='PK'
     and mp.object_id is not null
```

```
        AND mp.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        and mp.csv_object_id_Columna!=(SELECT STUFF((SELECT ','
            + CAST(ic.column_id AS varchar(45)) FROM sys.
           index_columns ic WHERE ic.object_id=i.object_id and
           ic.index_id=i.index_id ORDER BY ic.key_ordinal FOR
           XML PATH (''')), 1, 1, '''));

select '[VersionaPrimaria]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionaForanea]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaForanea]
( @Version int,
  @DB_id INT
)
AS
BEGIN
--12 LAS FK QUE FUERON ELIMINADAS DEL ESQUEMA
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
```

```
            ,[BaseDatosID]
            ,[TablaPadreID]
            ,[TablaReferenciadaID]
            ,[VersionID]
            ,[object_id]
            ,[NombreForanea]
            ,[csv_object_id_ColumnaReferenciada]
            ,[csv_object_id_ColumnaOrigen]
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        12 as SmoID, -- "borra FK"
        mtp.BaseDatosID,
        mtp.vTablaID as TablaPadreID,
        mtr.vTablaID as TablaReferenciadaID,
        @Version,
        mf.object_id, -- deleted object id
        '',
        '',
        '',
        'ver.FKD '''+mtp.NombreTabla+''','''+mf.NombreForanea+'
           ''' ,
        'ver.FKN '''+mtp.NombreTabla+''','''+mtr.NombreTabla+''
           ','''+mf.NombreForanea+''','''+mf.
           csv_object_id_ColumnaOrigen+''','''+mf.
           csv_object_id_ColumnaReferenciada+''''
```

```
        from [Metadatos1.4].DBO.Foranea mf
        inner join [Metadatos1.4].dbo.Tabla mtp on mtp.vTablaID
           =mf.TablaPadreID and mtp.versionID = (select MAX(
           VersionID) from [Metadatos1.4].dbo.Tabla metatvp
           where metatvp.object_id = mtp.object_id)
        inner join [Metadatos1.4].dbo.Tabla mtr on mtr.vTablaID
           =mf.TablaReferenciadaID and mtr.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Tabla metatvr
           where metatvr.object_id = mtr.object_id)
        left join sys.foreign_keys fk on fk.object_id=mf.
           object_id and fk.parent_object_id = mtp.object_id and
            mtp.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatvp where metatvp.
           object_id = mtp.object_id)
        where fk.object_id is null
        and mf.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Foranea metafv where metafv.
           object_id = mf.object_id)
        AND mf.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);
--8 LAS FK QUE NO EXISTEN EN LOS METADATOS
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
           ,[BaseDatosID]
           ,[TablaPadreID]
           ,[TablaReferenciadaID]
```

```
            ,[VersionID]
            ,[object_id]
            ,[NombreForanea]
            ,[csv_object_id_ColumnaReferenciada]
            ,[csv_object_id_ColumnaOrigen]
            ,[ReglaActualizacion]
            ,[Regla_Eliminacion])
select
        8 as SmoID, -- "crea FK"
        mtp.BaseDatosID,
        mtp.vTablaID,
        mtr.vTablaID,
        @Version,
        fk.object_id,
        fk.name,
        (SELECT STUFF((SELECT ',' + CAST(fkc.
           referenced_column_id AS varchar(45)) from sys.
           foreign_key_columns fkc where fkc.
           constraint_object_id=fk.object_id order by fkc.
           constraint_column_id FOR XML PATH ('')), 1, 1, ''))
           as [csv_object_id_ColumnaReferenciada],
        (SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
           AS varchar(45)) from sys.foreign_key_columns fkc
           where fkc.constraint_object_id=fk.object_id order by
           fkc.constraint_column_id FOR XML PATH ('')), 1, 1, ''
           )) as [csv_object_id_ColumnaOrigen],
```

```
    'ver.FKN ''''+mtp.NombreTabla+''',''''+mtr.NombreTabla+''
      ',''''+fk.name+''',''''+(SELECT STUFF((SELECT ',' +
      CAST(fkc.parent_column_id AS varchar(45)) from sys.
      foreign_key_columns fkc where fkc.
      constraint_object_id=fk.object_id order by fkc.
      constraint_column_id FOR XML PATH (''')), 1, 1, ''))+'
      '',''''+(SELECT STUFF((SELECT ',' + CAST(fkc.
      referenced_column_id AS varchar(45)) from sys.
      foreign_key_columns fkc where fkc.
      constraint_object_id=fk.object_id order by fkc.
      constraint_column_id FOR XML PATH (''')), 1, 1, ''))+'
      ''',
    'ver.FKD ''''+mtp.NombreTabla+''',''''+fk.name+''''
    from sys.foreign_keys fk
inner join [Metadatos1.4].dbo.Tabla mtp on fk.
   parent_object_id = mtp.object_id and mtp.versionID = (
   select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvp where metatvp.object_id = mtp.object_id)
inner join [Metadatos1.4].dbo.Tabla mtr on fk.
   referenced_object_id = mtr.object_id and mtr.versionID =
   (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvr where metatvr.object_id = mtr.object_id)
    LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
      object_id = fk.object_id and mf.versionID = (select
      MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
       where metafv.object_id = mf.object_id)
```

```
        where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(fk.schema_id) != 'ver'
        AND fk.type='F'
        and (mf.ForaneaID is null OR mf.SmoID not in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));


--9 LAS FK QUE EXISTEN PERO CAMBIARON DE NOMBRE
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
           ,[BaseDatosID]
           ,[TablaPadreID]
           ,[TablaReferenciadaID]
           ,[VersionID]
           ,[object_id]
           ,[NombreForanea]
           ,[csv_object_id_ColumnaReferenciada]
           ,[csv_object_id_ColumnaOrigen]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
        9 as SmoID, -- "rename FK"
        mtp.BaseDatosID,
        mtp.vTablaID,
        mtr.vTablaID,
        @Version,
        fk.object_id,
```

```
    fk.name,
    (SELECT STUFF((SELECT ',' + CAST(fkc.
      referenced_column_id AS varchar(45)) from sys.
      foreign_key_columns fkc where fkc.
      constraint_object_id=fk.object_id order by fkc.
      constraint_column_id FOR XML PATH ('')), 1, 1, ''))
      as [csv_object_id_ColumnaReferenciada],
    (SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
      AS varchar(45)) from sys.foreign_key_columns fkc
      where fkc.constraint_object_id=fk.object_id order by
      fkc.constraint_column_id FOR XML PATH ('')), 1, 1, ''
      )) as [csv_object_id_ColumnaOrigen],
    'ver.FKR␣'''+mtp.NombreTabla+''','''+mf.NombreForanea+'
      '','''+fk.name+'''',
    'ver.FKR␣'''+mtp.NombreTabla+''','''+fk.name+''','''+mf
      .NombreForanea+'''' --rollback
    from sys.foreign_keys fk
inner join [Metadatos1.4].dbo.Tabla mtp on fk.
  parent_object_id = mtp.object_id and mtp.versionID = (
  select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
  metatvp where metatvp.object_id = mtp.object_id)
inner join [Metadatos1.4].dbo.Tabla mtr on fk.
  referenced_object_id = mtr.object_id and mtr.versionID =
  (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
  metatvr where metatvr.object_id = mtr.object_id)
```

```
        LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
           object_id = fk.object_id and mf.versionID = (select
           MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
            where metafv.object_id = mf.object_id)
        where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
        and SCHEMA_NAME(fk.schema_id) != 'ver'
        AND fk.type='F'
        and mf.object_id is not null
        and mf.NombreForanea!=fk.name
        AND mf.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22);
--10 LAS FK QUE EXISTEN PERO SE LE MODIFICARON SUS COLUMNAS (
   creo q nunca pasara esto, hace delete->create)
INSERT INTO [Metadatos1.4].[dbo].[Foranea]
           ([SmoID]
           ,[BaseDatosID]
           ,[TablaPadreID]
           ,[TablaReferenciadaID]
           ,[VersionID]
           ,[object_id]
           ,[NombreForanea]
           ,[csv_object_id_ColumnaReferenciada]
           ,[csv_object_id_ColumnaOrigen]
           ,[ReglaActualizacion]
           ,[Regla_Eliminacion])
select
```

```
10 as SmoID, -- "modify FK"
mtp.BaseDatosID,
mtp.vTablaID,
mtr.vTablaID,
@Version,
fk.object_id,
fk.name,
(SELECT STUFF((SELECT ',' + CAST(fkc.
   referenced_column_id AS varchar(45)) from sys.
   foreign_key_columns fkc where fkc.
   constraint_object_id=fk.object_id order by fkc.
   constraint_column_id FOR XML PATH ('')), 1, 1, ''))
   as [csv_object_id_ColumnaReferenciada],
(SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
   AS varchar(45)) from sys.foreign_key_columns fkc
   where fkc.constraint_object_id=fk.object_id order by
   fkc.constraint_column_id FOR XML PATH ('')), 1, 1, ''
   )) as [csv_object_id_ColumnaOrigen],
'ver.FKM ''''+mtp.NombreTabla+''',''''+fk.name+''',''''+(
   SELECT STUFF((SELECT ',' + CAST(fkc.parent_column_id
   AS varchar(45)) from sys.foreign_key_columns fkc
   where fkc.constraint_object_id=fk.object_id order by
   fkc.constraint_column_id FOR XML PATH ('')), 1, 1, ''
   ))+''',''''+(SELECT STUFF((SELECT ',' + CAST(fkc.
   referenced_column_id AS varchar(45)) from sys.
   foreign_key_columns fkc where fkc.
```

```
        constraint_object_id=fk.object_id order by fkc.
        constraint_column_id FOR XML PATH (''')), 1, 1, ''')+'
        ''',
    'ver.FKM␣'''+mtp.NombreTabla+''',''+fk.name+''',''+(
        SELECT STUFF((SELECT ',' + CAST(fkc.
        referenced_column_id AS varchar(45)) from sys.
        foreign_key_columns fkc where fkc.
        constraint_object_id=fk.object_id order by fkc.
        constraint_column_id FOR XML PATH (''')), 1, 1, ''')+'
        '',''+(SELECT STUFF((SELECT ',' + CAST(fkc.
        parent_column_id AS varchar(45)) from sys.
        foreign_key_columns fkc where fkc.
        constraint_object_id=fk.object_id order by fkc.
        constraint_column_id FOR XML PATH (''')), 1, 1, ''')+'
        ''' -- rollback
    from sys.foreign_keys fk
inner join [Metadatos1.4].dbo.Tabla mtp on fk.
   parent_object_id = mtp.object_id and mtp.versionID = (
   select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvp where metatvp.object_id = mtp.object_id)
inner join [Metadatos1.4].dbo.Tabla mtr on fk.
   referenced_object_id = mtr.object_id and mtr.versionID =
   (select MAX(VersionID) from [Metadatos1.4].dbo.Tabla
   metatvr where metatvr.object_id = mtr.object_id)
    LEFT JOIN [Metadatos1.4].DBO.Foranea mf on  mf.
        object_id = fk.object_id and mf.versionID = (select
```

```
            MAX(VersionID) from [Metadatos1.4].dbo.Foranea metafv
             where metafv.object_id = mf.object_id)
          where OBJECT_NAME(fk.parent_object_id)!='sysdiagrams'
          and SCHEMA_NAME(fk.schema_id) != 'ver'
          AND fk.type='F'
          and mf.object_id is not null
          AND mf.SmoID in
            (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
          and
          (  mf.csv_object_id_ColumnaOrigen!=(SELECT STUFF((
            SELECT ',' + CAST(fkc.parent_column_id AS varchar(45)
            ) from sys.foreign_key_columns fkc where fkc.
            constraint_object_id=fk.object_id order by fkc.
            constraint_column_id FOR XML PATH ('')), 1, 1, ''))
          or mf.csv_object_id_ColumnaReferenciada!=(SELECT STUFF
            ((SELECT ',' + CAST(fkc.referenced_column_id AS
            varchar(45)) from sys.foreign_key_columns fkc where
            fkc.constraint_object_id=fk.object_id order by fkc.
            constraint_column_id FOR XML PATH ('')), 1, 1, ''))
          );

select '[VersionaForanea]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionaColumna]
   Script Date: 08/29/2013 23:06:34 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaColumna]
( @Version int,
  @DB_id INT
)
AS
BEGIN
--14    Eliminacion de Columna
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT  mc.vTablaID,mc.object_id, 14,  mc.NombreColumna, mc.
   Orden, mc.Longitud, mc.PermiteNulo, mc.ValorDefault,
'ver.CD␣'+mt.NombreTabla +',␣'+ mc.NombreColumna +''
, mc.ReglaActualizacion
,@DB_id --base de datos
,@Version -- version
FROM [Metadatos1.4].DBO.Columna mc
inner join [Metadatos1.4].dbo.Tabla mt on mc.vTablaID = mt.
   vTablaID
left join sys.columns sc on mc.object_id=sc.column_id
where
```

```
mc.versionID = (select MAX(VersionID) from [Metadatos1.4].dbo.
   Columna metacv where metacv.object_id = mc.object_id)
AND mc.SmoID in (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
and sc.object_id is null;

--3     Creacion de columna
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT  mt.vTablaID,sc.column_id, 3,  C.COLUMN_NAME, C.
   ORDINAL_POSITION, C.CHARACTER_MAXIMUM_LENGTH, c.IS_NULLABLE,
   c.COLUMN_DEFAULT,
'ver.CN␣'+mt.NombreTabla +',␣'+ C.COLUMN_NAME +',␣'+ CASE WHEN
   C.CHARACTER_MAXIMUM_LENGTH IS NULL THEN C.DATA_TYPE  ELSE +''
   ''+C.DATA_TYPE  + '(' + CAST(C.CHARACTER_MAXIMUM_LENGTH AS
   VARCHAR(50))+ ')''' END + ',␣'+C.IS_NULLABLE +',␣'+CAST(
   ISNULL(C.COLUMN_DEFAULT,'NULL') AS VARCHAR(20)) + '␣'
, 'ver.CD␣'+mt.NombreTabla +',␣'+ C.COLUMN_NAME +''
,@DB_id --base de datos
,@Version -- version
FROM INFORMATION_SCHEMA.COLUMNS c
inner join sys.columns sc on c.COLUMN_NAME = sc.name and
   OBJECT_NAME(sc.object_id)=c.TABLE_NAME
inner join [Metadatos1.4].DBO.Tabla mt on mt.object_id=sc.
   object_id and mt.versionID = (select MAX(VersionID) from [
```

```
   Metadatos1.4].dbo.Tabla metat where metat.object_id = mt.
   object_id)
LEFT JOIN [Metadatos1.4].DBO.Columna mc on mc.object_id=sc.
   column_id and mc.versionID = (select MAX(metacol.VersionID)
   from [Metadatos1.4].dbo.Columna metacol inner join [
   Metadatos1.4].dbo.Tabla metat on metat.vTablaID=metacol.
   vTablaID where metacol.object_id = mc.object_id and metat.
   object_id = mt.object_id)
where
c.TABLE_NAME!='sysdiagrams' and c.TABLE_SCHEMA != 'ver'
and (mc.ColumnaID is null OR mc.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))


--4      Cambio nombre Columna
INSERT INTO [Metadatos1.4].dbo.Columna (vTablaID, object_id,
   SmoID, NombreColumna, Orden, Longitud, PermiteNulo,
   ValorDefault, ReglaActualizacion, Regla_Eliminacion,
   BaseDatosID, VersionID)
SELECT mc.vTablaID, sc.column_id, 4--CR (renombrar columna)
, sc.name, null, null, null, null
, 'ver.CR␣' + mt.NombreTabla + ',␣'+ mc.NombreColumna + ',␣'+
   sc.name  regla1
,'ver.CR␣'+ mt.NombreTabla + ',␣'+ sc.name + ',␣'+ mc.
   NombreColumna  regla2
, @DB_id,@Version--1--version
FROM INFORMATION_SCHEMA.COLUMNS c
```

```
inner join sys.columns sc on c.COLUMN_NAME = sc.name and
   OBJECT_NAME(sc.object_id)=c.TABLE_NAME
inner join [Metadatos1.4].DBO.Tabla mt on mt.object_id=sc.
   object_id and mt.versionID = (select MAX(VersionID) from [
   Metadatos1.4].dbo.Tabla metat where metat.object_id = mt.
   object_id)
inner JOIN [Metadatos1.4].DBO.Columna mc on mc.object_id=sc.
   column_id and mc.versionID = (select MAX(metacol.VersionID)
   from [Metadatos1.4].dbo.Columna metacol inner join [
   Metadatos1.4].dbo.Tabla metat on metat.vTablaID=metacol.
   vTablaID where metacol.object_id = mc.object_id and metat.
   object_id = mt.object_id)
inner join [Metadatos1.4].dbo.Tabla mtc on mtc.vTablaID=mc.
   vTablaID
where  c.TABLE_NAME!= 'sysdiagrams'
and mtc.object_id=mt.object_id
and c.COLUMN_NAME != mc.NombreColumna
and c.TABLE_SCHEMA != 'ver'
AND mc.SmoID in (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)


END
GO
/****** Object:  UserDefinedFunction [ver].[fix_rutina_create]
      Script Date: 08/29/2013 23:06:35 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [ver].[fix_rutina_create]
(
        @rut_name varchar(max),
        @rut_code varchar(max)
)
RETURNS varchar(max)
AS
BEGIN
        DECLARE @ResultVar varchar(max)
        declare @es_procedimiento as integer;

        set @es_procedimiento=0;

        select @es_procedimiento=COUNT(1) from sys.objects o
           where o.name=@rut_name and o.type='P'
        if @es_procedimiento>0 begin
                set @ResultVar = ver.fix_sp_create(@rut_name,
                   @rut_code)
        end else begin
                set @ResultVar = ver.fix_sp_create(@rut_name,
                   @rut_code)
        end

        RETURN @ResultVar
```

```
END
GO
/****** Object:  StoredProcedure [ver].[VersionaVista]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaVista]
( @Version int,
  @DB_id int    )
AS
BEGIN
        --Bug MsSQL 2008, no se puede llamar a
           sp_refreshsqlmodule luego de un sp_rename
        --http://connect.microsoft.com/SQLServer/feedback/
           details/644572/sp-refreshsqlmodule-changes-object-
           definition
        --declare @view_name varchar(max)
        --DECLARE view_cursor CURSOR FOR
        --      SELECT o.name
        --      FROM sys.objects AS o
        --      WHERE o.type = 'V'
        --      and SCHEMA_NAME(o.schema_id)!='ver'
        --OPEN view_cursor
        --FETCH NEXT FROM view_cursor INTO @view_name
```

```
--WHILE @@FETCH_STATUS = 0
--BEGIN
--      EXEC sys.sp_refreshsqlmodule @view_name
--      FETCH NEXT FROM view_cursor INTO @view_name
--END
--CLOSE view_cursor
--DEALLOCATE view_cursor

-- 16   Eliminacion de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                    ([SmoID]
                    ,[NombreVista]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        16 as SmoID,
        mv.NombreVista, -- deleted view name
        mv.ScriptNuevo,
        '',
        'ver.VD␣'''+mv.NombreVista+'''',
        'ver.VN␣'''+mv.NombreVista+''',''+REPLACE(mv.
           ScriptNuevo,'''','''''')+'''',
```

```
        @DB_id,
        @Version
FROM [Metadatos1.4].DBO.Vista mv
LEFT JOIN sys.objects AS o ON mv.NombreVista =
   OBJECT_NAME(o.object_id)
WHERE mv.versionID = (select MAX(VersionID) from [
   Metadatos1.4].dbo.Vista metav where metav.NombreVista
    = mv.NombreVista)
AND mv.SmoID in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
AND o.object_id is null;

-- 15   Creacion de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                   ([SmoID]
                   ,[NombreVista]
                   ,[ScriptOriginal]
                   ,[ScriptNuevo]
                   ,[ReglaActualizacion]
                   ,[ReglaEliminacion]
                   ,[BaseDatosID]
                   ,[VersionID])
SELECT
        15 as SmoID,
        OBJECT_NAME(sm.object_id) as NombreVista,
        '',
```

```
        ver.fix_view_create(OBJECT_NAME(sm.object_id),
           sm.definition),
        'ver.VN␣'''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_view_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''',''''''')+'''',
        'ver.VD␣'''+OBJECT_NAME(sm.object_id)+'''',
        @DB_id,
        @Version
FROM sys.sql_modules AS sm
JOIN sys.objects AS o ON sm.object_id = o.object_id
LEFT JOIN [Metadatos1.4].DBO.Vista mv on  mv.
   NombreVista = OBJECT_NAME(sm.object_id) and mv.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Vista metav where metav.NombreVista = mv.
   NombreVista)
where o.type='V'
and SCHEMA_NAME(o.schema_id) != 'ver'
and (mv.VistaID is null OR mv.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));

-- 17   Cambio nombre de una vista
-- (por el momento NO APLICA)

-- 18   Cambio contenido de una vista
INSERT INTO [Metadatos1.4].[dbo].[Vista]
                    ([SmoID]
```

```
                    ,[NombreVista]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        15 as SmoID,
        OBJECT_NAME(sm.object_id) as NombreVista,
        mv.ScriptNuevo as ScriptOriginal,
        ver.fix_view_create(OBJECT_NAME(sm.object_id),
           sm.definition) as ScriptNuevo,
        'ver.VM␣'''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_view_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''','''''')+'''',
        'ver.VM␣'''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(mv.ScriptNuevo,'''','''''')+'''',
        @DB_id,
        @Version
FROM sys.sql_modules AS sm
JOIN sys.objects AS o ON sm.object_id = o.object_id
LEFT JOIN [Metadatos1.4].DBO.Vista mv on  mv.
   NombreVista = OBJECT_NAME(sm.object_id) and mv.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Vista metav where metav.NombreVista = mv.
```

```
           NombreVista)
        where o.type='V'
        and SCHEMA_NAME(o.schema_id) != 'ver'
        and mv.VistaID is not null
        AND mv.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        and ver.fix_view_create(OBJECT_NAME(sm.object_id),sm.
           definition)!=mv.ScriptNuevo;

        select '[VersionaVista]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionaTabla]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaTabla]
( @Version int,
  @DB_id int    )
AS
BEGIN
Declare @schema_id int
select @schema_id = schema_id from sys.schemas where name ='ver
   '
```

```
------------13  Eliminacion de tabla
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 13, mt.NombreTabla, 'ver.UD␣'+mt.
           NombreTabla+'␣', 'ver.UN␣'+mt.NombreTabla+'␣',@DB_id,
           @Version
        from [Metadatos1.4].dbo.Tabla mt
        left join sys.objects t on t.object_id = mt.object_id
        where mt.versionID = (select MAX(VersionID) from [
           Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        AND mt.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        AND t.object_id is null

------------1   Creación de tabla
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 1, t.name, 'ver.UN␣'+t.name+'␣', '
           ver.UD␣'+t.name+'␣',@DB_id,@Version
        from sys.objects t
        LEFT JOIN
```

```
        [Metadatos1.4].dbo.Tabla mt on t.object_id = mt.
           object_id and mt.versionID = (select MAX(VersionID)
           from [Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        where t.type ='U' and t.is_ms_shipped = 0 and t.name !=
            'sysdiagrams'
        AND (mt.vTablaID is null OR mt.SmoID not in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22))
        and t.schema_id != @schema_id
------------LAS TABLAS QUE FUERON RENOMBRADAS
        INSERT INTO [Metadatos1.4].dbo.Tabla([object_id], SMOID
           , NombreTabla, ReglaActualizacion, Regla_Eliminacion,
            BaseDatosID, VersionID)
        select t.object_id, 2,t.name,'ver.UR␣'+mt.NombreTabla+'
           ,' + t.name,  'ver.UR␣'+t.name+',' + mt.NombreTabla,
           @DB_id,@Version
        from sys.objects t
        LEFT JOIN
        [Metadatos1.4].dbo.Tabla mt on t.object_id = mt.
           object_id and mt.versionID = (select MAX(VersionID)
           from [Metadatos1.4].dbo.Tabla metatbl where metatbl.
           object_id = mt.object_id)
        where t.type ='U' and t.is_ms_shipped = 0 and t.name !=
            'sysdiagrams'
        AND mt.object_id IS NOT NULL and t.name!=mt.NombreTabla
            and t.schema_id != @schema_id
```

```
        AND mt.SmoID in
           (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
        END
GO
/****** Object:  StoredProcedure [ver].[VersionaRutina]
   Script Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionaRutina]
( @Version int,
  @DB_id int    )
AS
BEGIN
        --Bug MsSQL 2008, no se puede llamar a
           sp_refreshsqlmodule luego de un sp_rename
        --http://connect.microsoft.com/SQLServer/feedback/
           details/644572/sp-refreshsqlmodule-changes-object-
           definition
        --declare @sp_name varchar(max)
        --DECLARE sp_cursor CURSOR FOR
        --      SELECT o.name
        --      FROM sys.objects AS o
        --      WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS
           ', N'FT')
```

```
--      and SCHEMA_NAME(o.schema_id)!='ver'
--      and LEFT(o.name,3) not in ('fn_','sp_')
--OPEN sp_cursor
--FETCH NEXT FROM sp_cursor INTO @sp_name
--WHILE @@FETCH_STATUS = 0
--BEGIN
--      EXEC sys.sp_refreshsqlmodule @sp_name
--      FETCH NEXT FROM sp_cursor INTO @sp_name
--END
--CLOSE sp_cursor
--DEALLOCATE sp_cursor

-- 20   Eliminacion de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                    ([SmoID]
                    ,[TipoRutinaID]
                    ,[NombreRutina]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        20 as SmoID,
        mr.TipoRutinaID,
```

```
        mr.NombreRutina, -- deleted routine name
        mr.ScriptNuevo,
        '',
        'ver.RD␣'''+mr.NombreRutina+'''',
        'ver.RN␣'''+mr.NombreRutina+''','''+REPLACE(mr.
           ScriptNuevo,'''','''''')+'''',
        @DB_id,
        @Version
FROM [Metadatos1.4].DBO.Rutina mr
LEFT JOIN sys.objects AS o ON mr.NombreRutina =
   OBJECT_NAME(o.object_id)
WHERE
mr.versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Rutina metar where metar.NombreRutina = mr.
   NombreRutina)
AND mr.SmoID in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
AND o.object_id is null;

-- 19   Creacion de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                    ([SmoID]
                    ,[TipoRutinaID]
                    ,[NombreRutina]
                    ,[ScriptOriginal]
                    ,[ScriptNuevo]
```

```
                    ,[ReglaActualizacion]
                    ,[ReglaEliminacion]
                    ,[BaseDatosID]
                    ,[VersionID])
SELECT
        19 as SmoID,
        (CASE o.type WHEN 'P' THEN 2 ELSE 1 END) as
           TipoRutinaID,
        o.name,
        '',
        ver.fix_rutina_create(OBJECT_NAME(sm.object_id)
           ,sm.definition),
        'ver.RN '''+OBJECT_NAME(sm.object_id)+''','''+
           REPLACE(ver.fix_rutina_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''','''''')+'''',
        'ver.RD '''+OBJECT_NAME(sm.object_id)+'''',
        @DB_id,
        @Version
FROM sys.objects AS o
JOIN sys.sql_modules AS sm ON sm.object_id = o.
   object_id
LEFT JOIN [Metadatos1.4].DBO.Rutina mr on  mr.
   NombreRutina = OBJECT_NAME(sm.object_id) and mr.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Rutina metar where metar.NombreRutina = mr.
   NombreRutina)
```

```
WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS', N'FT')
and SCHEMA_NAME(o.schema_id)!='ver'
and LEFT(o.name,3) not in ('fn_','sp_')
and (mr.RutinaID is null OR mr.SmoID not in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22));

-- 21   Cambio nombre de una Rutina
-- (por el momento NO APLICA)

-- 22   Cambio contenido de una Rutina
INSERT INTO [Metadatos1.4].[dbo].[Rutina]
                   ([SmoID]
                   ,[TipoRutinaID]
                   ,[NombreRutina]
                   ,[ScriptOriginal]
                   ,[ScriptNuevo]
                   ,[ReglaActualizacion]
                   ,[ReglaEliminacion]
                   ,[BaseDatosID]
                   ,[VersionID])
SELECT
        22 as SmoID,
        (CASE o.type WHEN 'P' THEN 2 ELSE 1 END) as
           TipoRutinaID,
        o.name,
        mr.ScriptNuevo as ScriptOriginal,
```

```
        ver.fix_rutina_create(OBJECT_NAME(sm.object_id)
           ,sm.definition) as ScriptNuevo,
        'ver.RM␣'''+OBJECT_NAME(sm.object_id)+''',''''+
           REPLACE(ver.fix_rutina_create(OBJECT_NAME(sm.
           object_id),sm.definition),'''','''''')+'''',
        'ver.RM␣'''+OBJECT_NAME(sm.object_id)+''',''''+
           REPLACE(mr.ScriptNuevo,'''','''''')+'''',
        @DB_id,
        @Version
FROM sys.objects AS o
JOIN sys.sql_modules AS sm ON sm.object_id = o.
   object_id
LEFT JOIN [Metadatos1.4].DBO.Rutina mr on  mr.
   NombreRutina = OBJECT_NAME(sm.object_id) and mr.
   versionID = (select MAX(VersionID) from [Metadatos1
   .4].dbo.Rutina metar where metar.NombreRutina = mr.
   NombreRutina)
WHERE o.type in ('P',N'FN', N'IF', N'TF', N'FS', N'FT')
and SCHEMA_NAME(o.schema_id)!='ver'
and LEFT(o.name,3) not in ('fn_','sp_')
and mr.RutinaID is not null
AND mr.SmoID in
   (1,2,3,4,5,6,7,8,9,10,15,17,18,19,21,22)
and ver.fix_rutina_create(OBJECT_NAME(sm.object_id),sm.
   definition)!=mr.ScriptNuevo;
```

```
        select '[VersionaRutina]'
END
GO
/****** Object:  StoredProcedure [ver].[VersionarDB]    Script
   Date: 08/29/2013 23:06:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [ver].[VersionarDB]
( @DB varchar(100) )
AS
BEGIN
        DECLARE @Version INT--- OBTENER LA SIGUIENTE VERSION
        INSERT INTO [Metadatos1.4].dbo.MetadatoVersion (Fecha)
           VALUES (GETDATE())
        SELECT @Version = @@IDENTITY --1

        DECLARE @DB_id int --- OBTENER EL ID DE LA BASE DE
           DATOS QUE SE VA A VERSIONAR
        SELECT @DB_id = BaseDatosID FROM [Metadatos1.4].DBO.DB

        EXEC ver.VersionaTabla @Version, @DB_id

        exec ver.VersionaColumna @Version, @DB_id
```

```
        exec ver.VersionaPrimaria @Version, @DB_id

        exec ver.VersionaForanea @Version, @DB_id

        exec ver.VersionaVista @Version, @DB_id

        exec ver.VersionaRutina @Version, @DB_id

END
GO
```