UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

GESTIÓN DEL TIEMPO EN SISTEMAS DATA WAREHOUSE SOBRE BASES DE DATOS RELACIONALES

AUTOR: LAURA GONZÁLEZ MACHO
TUTOR: JUAN PEDRO CARAÇA-VALENTE
Septiembre - 2009

Índice

Índice de Figuras

1 INTRODUCCIÓN

1.1 Introducción

La Sociedad del Conocimiento se caracteriza por la utilización de la información para generar conocimiento, con el fin de mejorar los procesos de cualquier organización. La información es un bien cada vez menos restringido, más compartido y la ventaja competitiva de las organizaciones radica en interpretarla y convertirla en un elemento diferencial, en un activo productivo y rentable.

Los antiguos sistemas de información a la Dirección, que convertían datos operacionales en indicadores de gestión (la mayor parte de las veces de naturaleza económico-financiera), se han visto absorbidos y superados por un nuevo concepto del tratamiento de la información para la toma decisiones que, bajo el nombre de Inteligencia de Negocio (*Business Intelligence*), evoluciona con fuerza en el ámbito de las Tecnologías de la Información (TI). Este cambio ha sido propiciado por la propia evolución de las TI, que permiten un tratamiento cada vez más rápido, complejo e inmediato de los datos, la información y, en definitiva, el conocimiento.

Hoy en día, en las actividades diarias de cualquier organización, se generan datos como producto secundario, que son el resultado de todas las transacciones que se realizan en sus procesos empresariales.

La Inteligencia de Negocio tiene como objetivo que estos datos dejen de ser simples datos y pasen a convertirse, en un primer estadio, en información, y, finalmente, en conocimiento que ayude y enriquezca las decisiones de negocio.

Este objetivo se refleja en la definición habitual de Inteligencia de Negocio o Inteligencia Empresarial que suele definirse como la transformación de los datos de la compañía en conocimiento para obtener una ventaja competitiva. Desde un punto de vista más pragmático, y asociándolo directamente a las tecnologías de la información, se puede definir la Inteligencia de Negocio como el conjunto de metodologías, aplicaciones y tecnologías que permiten reunir, depurar y transformar datos de los sistemas transaccionales e información desestructurada (interna y externa a la compañía) en información estructurada, para su explotación directa o para su análisis y conversión en conocimiento soporte a la toma de decisiones sobre el negocio.

Las organizaciones desean explotar y maximizar el valor de su información para lograr tener una mayor ventaja competitiva en un entorno de creciente competitividad.

La Inteligencia de Negocio hace especial hincapié en los procesos de recolección y utilización efectiva de la información con el fin de mejorar la operación de un negocio, brindando a sus usuarios el acceso a la información clave que necesitan para llevar a cabo sus tareas habituales y más precisamente para poder tomar decisiones oportunas basadas en datos correctos y fiables.

Al contar con esta información exacta y en tiempo real es posible, por ejemplo, identificar y corregir situaciones antes de que se conviertan en problemas y en potenciales pérdidas de control de la empresa, pudiendo lograr nuevas oportunidades o readaptarse frente a la ocurrencia de sucesos inesperados.

Cuanto más relevante y útil sea la inteligencia que posea una organización sobre un negocio, sus clientes, proveedores, productos y servicios, operaciones, etc., mayor será su ventaja competitiva y se podrán tomar mejor decisiones. Por ejemplo, cuanto más se conoce a los clientes mejor se logra satisfacer sus necesidades e incrementar su fidelidad.

En la sociedad actual, es comúnmente aceptado que la información constituye un activo esencial para proporcionar beneficios significativos a cualquier organización y que, por lo tanto, representa una ventaja competitiva en el mundo de los negocios.

A fin de comprender cómo una organización puede crear Inteligencia de Negocio a partir de sus datos para, como ya se ha mencionado, proveer a los usuarios finales de un acceso a esta información de ayuda a la toma de decisiones, se describe a continuación, sucintamente, el proceso global de Inteligencia de Negocio.

- **Fase 1:** Dirigir y planificar. En esta fase inicial es dónde se deberán recolectar los requerimientos de información específicos de los diferentes usuarios, así como entender sus diversas necesidades, para que luego en conjunto con ellos se generen efectivamente las preguntas que les ayudarán a alcanzar sus objetivos.
- **Fase 2:** Recolección de información. Es en esta fase dónde se realiza el proceso de extraer desde la distintas fuentes de información de la empresa, tanto internas como externas, los datos que serán necesarios para encontrar las respuestas a las preguntas planteadas en la fase inicial.
- **Fase 3:** Procesamiento de datos. En esta fase se integran y cargan los datos en crudo en un formato utilizable para el análisis. Esta actividad puede realizarse mediante la creación de una nueva base de datos, agregando datos a una base de datos ya existente o bien consolidando la información.
- **Fase 4:** Análisis y producción. Se procederá, en esta fase, a trabajar sobre los datos extraídos e integrados, utilizando herramientas y técnicas propias de la tecnología de Inteligencia de Negocio. Como resultado final de esta fase se obtendrán las respuestas a las preguntas, mediante la creación de informes, indicadores, etc.
- **Fase 5:** Difusión. Finalmente se entregará a los usuarios las herramientas necesarias que les permitan explorar los datos de manera rápida y sencilla.

Como se desprende del ciclo expuesto, para crear una infraestructura que dé soporte a la Inteligencia de Negocio es imperativo y de vital importancia contar con una tecnología que permita gestionar, depurar e integrar datos procedentes de diversas fuentes, además

de almacenarlos en un único destino, depósito o base de datos que permita su posterior análisis y exploración.

Esta tecnología es el Data Warehouse, que básicamente se encarga de consolidar, integrar y centralizar los datos que la empresa genera en todos los ámbitos de una actividad de negocio, mediante una estructura que permite el acceso y exploración de la información requerida con un buen rendimiento, facilitando posteriormente una amplia gama de análisis que permitirá la toma de decisiones estratégicas y tácticas.

No es fácil realizar una definición de lo que es un Data Warehouse. Data Warehouse significa cosas diferentes para personas diferentes. Algunas definiciones se limitan al ámbito de los datos, otras hacen referencia al conjunto global de procesos, software, herramientas y a los propios datos. Aunque la definición exacta difiere, entre las distintas variantes casi siempre existe un nexo común. La definición más ampliamente usada y que se presenta aquí corresponde a W.H.Inmon –considerado el padre del Data Warehouse–, quien lo define como:

> *El Data Warehouse es una colección de datos temática, integrada, no volátil y variante en el tiempo diseñada para ayudar en la toma de decisiones.*

Acudiendo a una descripción más prosaica, un Data Warehouse es una arquitectura de gran almacén. Un enorme almacén que contiene una clase especial de producto: DATOS. Datos procedentes de cualquier lugar, normalmente de naturaleza heterogénea, en grandes volúmenes y con unas características especiales.

Los datos deben estar bien definidos, ser consistentes y de naturaleza no volátil. Adicionalmente, deben ser capaces de proporcionar información histórica. El Data Warehouse es una pieza -no la única, pero con seguridad una de las más importantes- del enfoque utilizado para atender las necesidades de información y análisis de la empresa.

Un Data Warehouse es ***temático*** porque la información se clasifica entorno a los aspectos o temas que son de interés para la empresa. Así organizado, los conceptos de negocio relacionados están asociados entre sí dentro del Data Warehouse. Por ejemplo, en una empresa de telecomunicaciones, se organizará alrededor de entidades de alto nivel como: clientes, líneas telefónicas, proveedores, etc. Y a su vez, conceptos como cliente estarán relacionados con otros como área geográfica, productos y servicios y valoración del cliente. En contraposición, las aplicaciones de los sistemas operacionales tradicionales se construyen alrededor de las funciones que necesita la organización para llevar a cabo sus actividades diarias. Siguiendo con el ejemplo, la información operacional estará asociada entorno a procesos como: mediación, tarifación, facturación o marketing. Cada una de las funciones está relacionada con uno o más conceptos de

negocio. Se trata de dos enfoques diferentes y, por tanto, para la creación del Data Warehouse es necesario reestructurar y alinear, alrededor de los conceptos de negocio de la empresa, la información procedente de los sistemas operacionales.

La característica de ***integración*** se refiere a la transformación de los datos desde la visión de las aplicaciones hacia una percepción integrada dentro del Data Warehouse. Cada aplicación tiene su forma particular de "ver" los datos -según y cómo sea el proceso que los trate-. Pero cuando los datos se introducen en el Data Warehouse, las distintas visiones de las aplicaciones deben entrelazarse y fusionarse en una sola. Esto implica que en el Data Warehouse habrá una única estructura -donde antes existían muchas, una por cada aplicación- para los conceptos de negocio citados previamente, como cliente, productos y servicios, área geográfica, etc. Esta integración exige, necesariamente, alcanzar el consenso en la definición de los conceptos, de manera que, en el ámbito de empresa, los términos de negocio tengan un único significado y éste sea conocido por todos.

La ***no volatilidad*** de la información hace referencia a que los datos, una vez introducidos en el Data Warehouse, no están, usualmente, sujetos a cambios. La información es útil para el análisis y la toma de decisiones sólo cuando es estable. Cualquiera que haga uso del Data Warehouse ha de tener la confianza de que una consulta producirá siempre el mismo resultado independientemente de cuando se realice. Las bases de datos son extremadamente volátiles en el sentido de que están siempre cambiando. Es bastante improbable que una consulta produzca los mismos resultados si se realiza dos veces en instantes diferentes (si las tablas a las que accede se actualizan con frecuencia). Sin embargo, una de las asunciones implícitas en un Data Warehouse es que, una vez que se ha registrado un evento, éste nunca se modifica, de manera que la única operación de escritura posible consiste en añadir nuevos eventos a medida que éstos ocurren. Aunque esta asunción es aceptable para una amplia variedad de dominios, alguna aplicaciones exigen un comportamiento diferente. En particular, los valores de una o más medidas de un evento puede cambiar a lo largo de un periodo de tiempo para ser consolidadas sólo después de que el evento haya sido registrado por primera vez en el Data Warehouse. En este contexto, si la situación actual debe ser presentada de manera precisa a los usuarios, los eventos pasados deben ser actualizados para reflejar la nueva información.

La necesidad de actualizaciones surge típicamente cuando las medidas iniciales de los eventos pueden estar sujetas a error (ej. La cantidad de una factura puede ser corregida después de que la factura se haya registrado) o cuando los eventos evolucionan inherentemente en el tiempo (ej: las notificaciones de las inscripciones en la universidad puede ser recibidas y registradas días después de que fueron realizadas). Desafortunadamente, si las actualizaciones se realizaran mediante la sobreescritura física de los eventos pasados, surgirían varios problemas:

- la justificación y la trazabilidad de la información requieren la capacidad de preservar la información exacta sobre la que el analista basó su decisión. Si los antiguos eventos son reemplazados por sus “nuevas” versiones, las decisiones pasadas ya no pueden justificarse.
- En algunas aplicaciones tener acceso sólo a la versión más actual de la información no es suficiente para asegurar la calidad del análisis. Un caso típico es el de las consultas que requieren comparar el progreso de un fenómeno activo con ocurrencias pasadas del mismo: puesto que los datos registrados del proceso activo todavía no han sido consolidados, su comparación con los del pasado consolidado puede no ser significativa.

Por último, la característica de “***variante en el tiempo***” o característica ***temporal*** del Data Warehouse tiene como objetivo que los cambios producidos en los sistemas operacionales que afectan a la información contenida en el Data Warehouse puedan ser registrados de manera que sea posible reconstruir su evolución en todo momento. Cada registro del Data Warehouse tiene el tiempo indisolublemente "grabado" en él. Una vez creado de manera fiable un registro en el Data Warehouse, éste no debe ser modificado (aspecto no volátil).

La manera más sencilla de comprender de qué manera los registros son dependientes en el tiempo es compararlos con los registros estándar de una base de datos. Consideremos un registro estándar de una base de datos: a medida que el entorno cambia, de la misma manera cambian los valores, la información es actualizada, borrada o insertada dentro del registro de la base de datos. Sin embargo, en el registro del Data Warehouse la información es única y, simplemente, es insertada, junto a la información temporal asociada. Se puede acceder a la información dentro del registro del Data Warehouse, pero, una vez introducida, la información no varia.

Mientras que los sistemas operacionales están diseñados –entre otras características– para proporcionar tiempos de respuesta específicos (usualmente lo más cortos posible), el foco de los Data Warehouse es, generalmente, el análisis estratégico de información integrada a partir de sistemas heterogéneos [Inmon, 95]. El proceso de integración de la información puede ser muy complejo y cubre la adquisición, extracción, transformación y carga de la información en el Data Warehouse. Tradicionalmente no existe una conexión en tiempo real entre un Data Warehouse y sus fuentes de información, porque el enfoque de escribir-una-vez-leer-muchas (write-once read-many) del Data Warehouse entraría en conflicto con la carga continúa de información desde los sistemas operacionales y daría lugar a tiempos de respuesta no aceptables. La carga de datos en el Data Warehouse suele realizarse como procesos batch con periodicidades establecidas (por ejemplo cada noche, una vez a la semana) durante ventanas de tiempo que no afecten a la capacidad de análisis del sistema.

1.2 Estructura del proyecto

A continuación se describe brevemente la organización de este documento con el contenido de cada uno de sus capítulos:

- El Capítulo 2 describe la situación actual de la gestión temporal, combinando la visión práctica utilizada en los Data Warehouse y la visión teórica de las bases de datos temporales.
- El Capítulo 3 define como objetivo de este trabajo la definición de un modelo de gestión del tiempo con unas características determinadas.
- En el Capítulo 4 se presenta la definición del modelo bitemporal, comenzando desde un modelo básico, prácticamente sin gestión del tiempo, y construyendo modelos cada vez más completos añadiendo conceptos de forma gradual hasta llegar a este modelo final.
- En el Capítulo 5 se exponen los resultados de este trabajo, añadiendo a la definición del modelo de gestión del tiempo presentado en el capítulo anterior, un checklist de aplicación a los proyectos que deban incorporar gestión del tiempo.
- En el Capítulo 6 se resumen las conclusiones de este trabajo y se describe el cumplimiento de los objetivos marcados al inicio del mismo.
- Por último, en el Capítulo 7 se presentan posibles alternativas para la continuación de este trabajo y se identifican varias líneas de investigación activas actualmente.

2 ESTADO DE LA CUESTIÓN

En este capítulo se expone brevemente la teoría perteneciente a las diferentes áreas de conocimiento consideradas para la realización de este trabajo, prestando especial atención a su situación actual desde el punto de vista de su aplicación en el mundo empresarial.

En primer lugar se describe la problemática de la gestión del tiempo en los sistemas Data Warehouse tal como fue presentada y resuelta originalmente por uno de los autores más prolíficos en lo que a metodología de Data Warehouse se refiere, Ralph Kimball. La teoría por él expuesta sigue vigente actualmente y en ella se basan gran parte de los sistemas Data Warehouse que se construyen hoy en día.

Una vez presentada la solución propuesta originalmente al problema de la gestión del tiempo en sistemas Data Warehouse, se presenta la teoría de las bases de datos temporales, de las cuales se exponen sus principales conceptos y situación actual.

El hecho es que a día de hoy las bases de datos temporales siguen siendo todavía proyectos “de laboratorio” que no han llegado a materializarse en productos comerciales. Sin embargo, los fabricantes de SGBDs parece que han empezado a darse cuenta de que la necesidad de funcionalidad temporal se está convirtiendo en un requisito en la mayoría de los sistemas y es demandada cada vez con mayor fuerza. Y el camino por el que parece que se va a avanzar es la incorporación de funcionalidades temporales dentro de los motores de bases de datos relacionales existentes actualmente.

Por ello, para finalizar, se describen las dos implementaciones que más destacan actualmente de SGBDs con soporte temporal.

2.1 Gestión del tiempo en sistemas Data Warehouse

Los sistemas Data Warehouse surgieron a mediados de la década de 1990. Éstos integran y consolidan gran cantidad de información proveniente de diferentes fuentes de una organización.

Ralph Kimball, considerado el “gurú” del Data Warehouse, expone en [Kimball, 96] la problemática de lo que denomina “slowly changing dimensions” (dimensiones que cambian lentamente) proponiendo tres tipos de tratamientos (denominados “tipo 1”, “tipo 2” y “tipo 3”) para la gestión del tiempo en bases de datos relacionales que, varios años después, se han visto ampliados con la inclusión de algunos nuevos tipos.

2.1.1 Slowly changing dimensions

En el diseño inicial de una base de datos dimensional para el Data Warehouse, se asume que cada una de las dimensiones es lógicamente independiente del resto de dimensiones. En particular, dimensiones como producto o cliente se asumen independientes del tiempo. Pero en el mundo real esto no es estrictamente cierto, ya que

a lo largo del tiempo la descripción y la composición de los productos reales evolucionan constantemente. Los clientes en particular cambian de nombre, se casan y se divorcian, tienen más hijos y cambian de residencia. También las fuerzas de ventas periódicamente cambian la configuración de sus distritos y regiones. Como diseñadores de bases de datos es preciso tomar una decisión sobre cómo manejar estas situaciones ya que una de las principales responsabilidades del Data Warehouse es representar correctamente la historia.

La respuesta no es colocar todo en la tabla de hechos o hacer todas las dimensiones dependientes del tiempo. Esto daría lugar a un diagrama entidad relación explosivo con consecuencias desastrosas de pérdida de inteligibilidad y rendimiento. En lugar de eso, se debe tener en cuenta el hecho de que la mayoría de las dimensiones son *casi* constantes a lo largo del tiempo y es posible preservar la estructura dimensional realizando aditivos relativamente menores para capturar esta naturaleza cambiante. Se denomina a estas dimensiones *casi* constantes *slowly changing dimensions* (SCD) y para su tratamiento Kimball identificó originalmente tres tratamientos diferentes, a los que denominó “Tipo 1”, “Tipo 2” y “Tipo 3”. Esta lista se ha ampliado de manera que a fecha de hoy son seis los tipos considerados.

- **Tipo 0:** Es el enfoque pasivo de mantenimiento de cambios de valores, en el cual no se hace nada. Los valores permanecen tal como fueron introducidos originalmente. En ciertas circunstancias puede ser válido, pero es más frecuente utilizar alguno de los niveles siguientes para garantizar la preservación de la historia.
- **Tipo 1:** Se sobrescribe el dato antiguo con el nuevo, y por tanto no se registra la historia. Se considera apropiado para corrección de ciertos tipos de errores, como por ejemplo errores tipográficos en un nombre (asumiendo que nunca interesará saber cómo estaba el dato cuando era incorrecto). La ventaja de este tipo es que es muy sencillo de implementar.
- **Tipo 2:** Registra datos históricos mediante la creación de múltiples registros con claves diferentes. Con el tipo 2 se preserva toda la historia mediante la inserción de un nuevo registro cada vez que se hace un cambio.

  En este caso se añade a la dimensión un nuevo atributo booleano que tomará valor “1” para el último valor y “0” para todos los anteriores.

  La utilización de SCDs de tipo 2 segmenta la historia. Se puede utilizar el primer registro hasta una fecha concreta en los registros de las tablas de hechos creados para él. A partir de esa fecha, se utiliza el segundo registro de la dimensión en la tabla de hechos. Si se seleccionan ambos registros de la dimensión y se cruzan con la tabla de hechos, el primer registro cruzará únicamente con los registros de la tabla de hechos anteriores al cambio, y el segundo registro cruzará únicamente con los registros de la tabla de hechos posteriores al cambio.

El hecho de no tener que filtrar por fechas en las SCDs de tipo 2 tiene como objetivo evitar complejidad adicional a las aplicaciones reduciendo el uso de filtros innecesarios.

- **Tipo 3:** Registra los cambios utilizando columnas separadas. Mientras que el tipo 2 puede registrar un número ilimitado de cambios, el tipo 3 está limitado al número de columnas dedicadas para mantenimiento de datos históricos. Por ejemplo, si se desea registrar la historia de un atributo "residencia" la dimensión tendría los atributos "residencia original" y "residencia actual", limitando a dos el número de valores posibles a guardar. El Tipo 3 es útil en los casos en que se quiere ver "cómo habría sido el pasado si esto hubiera sido así desde siempre". Por ejemplo, esto ocurre frecuentemente cuando se describen cambios en fuerzas de ventas. Hay un momento del tiempo en el que se modifica la composición de todos los distritos de ventas. Durante unos pocos meses, existe el deseo de analizar la historia pasada en base a la nueva composición, y al contrario, analizar nueva historia en base a la antigua composición. El objetivo es ser capaz de comparar el rendimiento a través de la transición.
- **Tipo 4:** Se suele denominar como el método de las "tablas de histórico", donde una tabla almacena los datos actuales y se utiliza una tabla adicional para almacenar los cambios de valores.
- **Tipo 6:** Este método combina los anteriores 1, 2 y 3 (1 + 2 + 3 = 6), de ahí su nombre. No se utiliza con frecuencia porque es complicado para el acceso de usuario, pero presenta ciertas ventajas sobre el resto, especialmente si se emplean técnicas para mitigar esa complejidad de cara al usuario. Se trata de utilizar el Tipo 1 (actualizaciones) junto con el Tipo 2 (añadir filas) y el Tipo 3 (añadir columnas), pero añadiendo una pareja adicional de columnas para indicar el rango de fechas al cual aplica cada fila en particular.
- **Tipo 6 (implementación alternativa):** Uno de los inconvenientes de la implementación anterior es que existe una relación N:N entre la dimensión y la tabla de hechos que únicamente se puede resolver en tiempo de ejecución, a nivel de informe, cuando un usuario introduce un valor para el parámetro "Fecha". La consecuencia es que no se puede forzar la integridad referencial del SGBD entre la tabla de dimensión y la tabla de hechos. Existe una variante de este Tipo 6 que conserva todas sus ventajas y suprime este inconveniente. Se trata de definir una clave principal en la tabla de la dimensión formada por una clave subrogada y un número de versión. El número de versión del registro actual en la dimensión será siempre 0. Antes de registrar un cambio, se copia la versión 0 como versión n+1 y así la versión 0 se puede actualizar con los nuevos valores.

En la tabla de hechos, cuando se añade un hecho, sólo se puede hacer utilizando el registro actual de la dimensión, consecuentemente, todos los registros de la tabla de hechos tendrán un número de versión igual a 0.

La ventaja de esta implementación es que se resuelve mediante el modelo la relación N:N, dando prioridad a la versión actual (versión número 0).

Tal como ya se ha adelantado, la teoría de las SCDs sigue vigente en la actualidad y las soluciones propuestas por Kimball se siguen utilizando en la mayor parte de los sistemas Data Warehouse que se construyen actualmente.

2.2 Bases de datos temporales

En contraposición al origen práctico del Data Warehouse se encuentran las bases de datos temporales, que surgen como objeto de estudio e investigación en los años 70 y evolucionan en un entorno totalmente académico.

Una base de datos temporal es un Sistema de Gestión de Bases de Datos (SGBD) que implementa y trata con especial énfasis aspectos temporales, teniendo un modelo de datos temporal y una versión temporal del lenguaje SQL.

Una base de datos temporal fue presentada por Snodgrass [Snodgrass, 00] como una base de datos que soporta “tiempo de validez” o “tiempo de transacción” o ambos:

- Tiempo de validez: indica el periodo de tiempo en el cual un hecho es verdad en el mundo real.
- Tiempo de transacción: indica el periodo de tiempo en el cual un hecho está guardado en la base de datos.
- Dato bitemporal: es la combinación del tiempo de validez y el tiempo de transacción.

Los tiempos de transacción y de validez de una BD no tienen porqué coincidir para un mismo hecho:

- El tiempo de transacción viene marcado por el reloj interno del sistema.
- El tiempo de validez de un hecho puede ser:
    - Posterior al tiempo de transacción, lo que se llama actividad proactiva.
    - Anterior al tiempo de transacción, lo que se llama actividad retroactiva.
    - Simultáneo al tiempo de transacción, se llama actividad simultánea.

Se han propuesto muchos modelos de datos temporales, pero casi ninguno de ellos se ha llegado a implementar, y no existe en la actualidad ningún SGBD temporal comercial, sino únicamente algunos prototipos.

Un SGBD temporal debería soportar:

1. Un lenguaje de definición de datos temporales.
2. Un lenguaje de manipulación (consulta y modificación) de datos temporales.
3. Unas restricciones temporales.

La historia de las bases de datos temporales corre paralela a la historia de las bases de datos. Con el desarrollo del SQL y su utilización en aplicaciones empresariales, inmediatamente se vio la utilidad de añadir una fecha a la clave principal de una tabla para registrar el histórico de cambios. Sin embargo, este aspecto no fue tenido en cuenta en el estándar SQL92.

En 1992 Richard Snodgrass propuso que una comunidad dedicada a las bases de datos temporales desarrollase una extensión temporal para SQL. En respuesta a esta propuesta, en 1993 se constituyó un comité para diseñar esta extensión, que fue desarrollada a lo largo de ese año y que es conocida como TSQL2. A finales de 1993, Snodgrass presentó este trabajo al comité técnico de ANSI responsable del lenguaje SQL, apareciendo la especificación preliminar de TSQL2 en el ACM SIGMOD Record de Marzo de 1994 [Snodgrass, 00]. En base a comentarios y respuestas a esa especificación preliminar, se hicieron cambios al lenguaje, y se publicó la versión definitiva de la Especificación del Lenguaje TSQL2 en Septiembre de 1994.

Algunas partes de TSQL2 se incluyeron en un nuevo subestándar de SQL3 (SQL:1999), denominado SQL/Temporal. Esta nueva parte fue aceptada en la reunión de Ottawa en Enero de 1995 como Parte 7 de la especificación de SQL3. Entonces comenzó el debate sobre la ampliación de SQL/Temporal para incorporar soporte para tiempo de validez y tiempo de transacción y, debido a desacuerdos dentro del comité ISO sobre cómo debería incluirse este soporte, el proyecto responsable del soporte temporal fue cancelado hacia 2001. Desde entonces, el borrador de trabajo “SQL/Temporal, Parte 7” está en suspense.

Los motivos de este estancamiento son tres. En primer lugar, hasta ese momento no parecía haber demasiado entusiasmo por parte de los SGBDs comerciales (o más bien, por parte de sus clientes) por disponer de soporte temporal en las bases de datos. En segundo lugar, los grupos de estándares SQL se vieron forzados a concentrarse en completar el estándar SQL3 (en parte porque estaba siendo más complejo y llevando más tiempo de lo esperado) y tuvieron que dejar de lado el trabajo en un proyecto que aparentemente tenía poca demanda de mercado. Y finalmente, existe un desacuerdo fundamental entre dos líneas de trabajo sobre qué funcionalidades debería proporcionar el SQL/Temporal y cómo debería hacerlo, es decir, desavenencias tanto sobre la sintaxis como sobre la semántica.

Una vez que se finalizó el desarrollo de SQL3, dando como resultado la publicación de SQL:1999, los participantes se dieron cuenta de que el mercado estaba mostrando un creciente interés sobre los beneficios del soporte temporal y al menos un SGBD

comercial mostraba interés por el tema. Por este motivo el desarrollo sobre el SQL/Temporal ha vuelto a revivir recientemente por lo que se espera llegar a la ver su publicación en breve.

Como resultado de toda esta situación, los profesionales de las TI sólo disponen por el momento del estándar SQL para manipular sus datos temporales.

No obstante, las ideas y conceptos descritos en la especificación de TSQL2, como tiempo de validez, tiempo de transacción y tablas bitemporales, se han afianzado desde entonces en la literatura sobre bases de datos temporales. En el año 2002 Chris Date, Hugh Darwen y Nikos Lorentzos presentaron en su libro [Date, 02] el tratamiento para este tema incluyendo muchos de los términos presentados por TSQL2.

Los SGBDs comerciales como Oracle, Sybase, Informix son SGBDs no temporales. Pueden incorporar tipos de datos para almacenar datos como eventos, pero no proporcionan un lenguaje de consulta y manipulación de datos temporales ni un lenguaje de definición de datos temporales ni restricciones temporales.

La única alternativa existente en la actualidad para manejar datos temporales consiste en implementar funcionalidad temporal utilizando los mecanismos que los SGBDs relacionales ponen a nuestra disposición. En esta línea, se describen a continuación dos implementaciones diferentes basadas en la extensión de un modelo de datos no temporal a un modelo de datos temporal. Se trata de TimeDB, con difusión en el entorno académico, y Oracle Workspace Manager, el primer intento de incorporación de funcionalidad temporal en un SGBD relacional comercial.

2.3 TimeDB, la implementación de laboratorio

Existen algunos proyectos de implementación de funcionalidad temporal sobre SGBDs relacionales y orientados a objetos, como TimeDB [Steiner, 98], OODAPLEX [Wuu, 93], y JTemporal [jtemporal], todas ellas consideradas “proyectos” sin ninguna aplicación en el mundo empresarial. Entre todos ellos destaca la utilización de TimeDB en el mundo académico, principalmente en el área de la docencia para la realización de prácticas y el estudio de las bases de datos temporales.

Se trata de un SGBD *relacional temporal* basado en SQL. No se puede considerar un SGBD temporal ya que traduce las sentencias de un SQL temporal (ATSQL2, que se trata de una adaptación propia de TSQL2) en sentencias de SQL estándar que después son ejecutadas en un SGBD comercial como Oracle, Sybase, etc.

Incluye un lenguaje de consulta, un lenguaje de manipulación de datos y un lenguaje de definición de datos y restricciones.

Fue desarrollada por Andreas Steiner como parte de su tesis doctoral [Steiner, 98], en la que se encuentran también definiciones, conceptos y descripciones de implementación de un modelo de datos completo.

La primera versión de TimeDB fue escrita utilizando SICStus Prolog. La siguiente, ya está basada en Java, utiliza JDBC, dispone de un API y ofrece mayor funcionalidad.

La ventaja de esta solución es que una base de datos existente se puede migrar a una base de datos temporal mientras las aplicaciones legadas siguen funcionando sin necesidad de cambios.

TimeDB es capaz de almacenar la historia de los datos con tiempo de validez y tiempo de transacción.

Una tabla en el DBMS relacional "bitemporal" TimeDB puede ser:

- Una tabla snapshot (almacena sólo datos actuales)
- Una tabla con tiempo de validez (almacena cuándo los datos son válidos en el mundo real)
- Una tabla con tiempo de transacción (almacena cuándo los datos son almacenados en la base de datos)
- Una tabla bitemporal (almacena los tiempos de validez y transacción)

Una versión extendida de SQL permite especificar el tipo de tabla en el momento de su creación y también hace posible alterar tablas existentes.

Además, soporta consultas temporales, sentencias de modificación temporales y restricciones temporales, mediante la implementación de un lenguaje de definición de datos temporales, un lenguaje de manipulación (consulta y modificación) de datos temporales y restricciones temporales (integridad referencial temporal).

2.4 Oracle Workspace Manager, la implementación comercial

Por su parte Oracle, fiel a su agresiva política de incorporación de nuevas funcionalidades, es el primer SGBD comercial que ha anunciado la implementación de funcionalidad temporal sobre su SGBD relacional.

Recientemente, Oracle ha presentado un nuevo componente llamado "Workspace Manager" que permite gestionar versiones actuales, históricas y futuras de los datos en la misma base de datos. La última versión cumple con TSQL2 (o SQL/Temporal).

Se trata de la primer (y único, por el momento) SGBD comercial que ha anunciado soporte para este tipo de funcionalidad.

En el "ANEXO A: Workspace Manager en Oracle 11 G" se incluye información más detallada sobre la implementación de funcionalidad temporal de este producto.

3 PLANTEAMIENTO DEL PROBLEMA

Uno de los aspectos menos comprendido en su verdadera implicación a la hora de diseñar un Data Warehouse es el tratamiento y representación del tiempo. Como se ha mostrado en la introducción la presencia del tiempo y la dependencia del mismo es una de las características que identifica y diferencia al Data Warehouse de los sistemas operacionales tradicionales. La mayoría de las aplicaciones empresariales están preparadas para funcionar en el entorno “del presente" donde el tiempo no exige un tratamiento especial. En gran parte de los casos, las fechas no son más que meros atributos descriptivos. Sin embargo, en un Data Warehouse el tiempo afecta a la estructura misma del sistema. Las exigencias relativas al tiempo en un Data Warehouse son muy diferentes de las de un sistema operacional, aunque sea el sistema operacional quien proporcione la información sobre las modificaciones en los datos al Data Warehouse.

La inclusión del tiempo en los Data Warehouse permite disponer de información histórica y hacer consultas sobre la misma. Esto significa que los usuarios del Data Warehouse pueden ver aspectos de sus empresas en un punto específico o durante un periodo de tiempo. Esta característica permite, a su vez, la observación de patrones de comportamiento a lo largo del tiempo de manera que es posible realizar comparaciones entre periodos similares o diferentes, por ejemplo, este año respecto al anterior. Armados con esta información es posible facilitar los procesos de toma de decisión que serán más exactos y fiables cuanto más lo sean los datos de partida. Se está, en efecto, utilizando el pasado para intentar predecir el futuro.

En definitiva, el Data Warehouse tiene como objetivo describir la evolución histórica de una organización e idealmente de manera que esta descripción sea lo suficientemente precisa como para ser capaz de justificar o mantener la trazabilidad de los cambios en la propia información. Es decir, no basta con responder a “cómo estaban yendo mis ventas el año pasado” sino también a cuestiones como “por qué el informe del mes pasado mostraba ventas por 100€ y hoy muestra que las ventas fueron sólo 25€”.

Cuando se considera la información temporal en los Data Warehouse es preciso comprender cómo se refleja el tiempo en la base de datos, cómo se relaciona con la estructura de la información y cómo un cambio de estado afecta a la información existente. Hay varios enfoques al respecto:

- Información transitoria: la característica principal de la información transitoria es que la alteración y borrado de los registros existentes destruyen físicamente el contenido previo de la información. Este tipo de información se encuentra habitualmente en los entornos operacionales.
- Información periódica: una vez que el registro se ha añadido a la base de datos, éste nunca se borra físicamente ni su contenido es modificado. En su lugar,

siempre se añaden nuevos registros para reflejar actualizaciones o incluso borrados. La información periódica, por tanto, contiene un completo registro de los cambios que han ocurrido. Los Data Warehouse son de naturaleza periódica.

- Información semi-periódica: Esta clase de información típicamente se encuentra en la información de tiempo de real de los sistemas operacionales donde los estados previos son importantes (sistemas de cuentas bancarias, sistemas de seguros, etc.). Sin embargo, casi todos los sistemas operacionales mantienen sólo un pequeña historia de los cambios realizados en la información debido a restricciones de rendimiento o almacenamiento. De ahí que este tipo de información pueda ser definida como semi-periódica.
- Información tipo “snapshot”: Este tipo de información representa una vista estable de la información tal y como existe en un momento dado del tiempo. Es un tipo especial de información periódica. Generalmente los “snapshots” representan la información en un momento concreto del pasado, y mediante una serie de “snapshots” tomadas en distintos puntos del tiempo se puede proporcionar una vista de la historia de una organización.

El enfoque habitual para almacenar información periódica (la que se encuentra normalmente en los Data Warehouse) es emplear registros de estado con fecha (timestamps) y eventos.

El enfoque de utilizar un único “timestamp” para guardar sólo la fecha de inicio de cuando un registro se considera válido es aplicable a los eventos, pero presenta serias deficiencias en el contexto de los Data Warehouse, dónde en general se almacena información de estado.

Existen dos tipos habituales de consultas en un entorno Data Warehouse que explican el problema:

- Una consulta que necesita acceder a la información actual. En el esquema de un único timestamp, la única manera de identificar los registros actuales es buscar el último timestamp, lo cual puede constituir un proceso computacionalmente exigente.
- Una consulta que reconstruye una vista de la información en un momento particular del pasado. Para dar soporte a esta clase de consulta es necesario conocer el periodo de validez de cada registro con el objeto de compararlo con el periodo de tiempo requerido. Con el enfoque de único timestamp, el final del periodo de validez sólo se puede obtener a partir del siguiente registro en la secuencia. Por lo general, también se trata de un proceso computacionalmente exigente.

Con el objeto de resolver el primer problema, se añade un segundo timestamp (fecha fin) a cada hecho. Este timestamp identifica el final del periodo de validez. De esta

manera se mejora el rendimiento en la recuperación de información. Sin embargo, no es suficiente para resolver totalmente el segundo problema, porque el periodo de validez puede cambiar a lo largo del tiempo debido a la posterior incorporación de nueva información en el Data Warehouse.

Una vista consistente desde el punto de vista temporal (similar a la información snapshots) durante el proceso analítico requiere guardar ambos periodos de validez (el antiguo y el nuevo). Por lo tanto, los modelos de datos del Data Warehouse se ven mejorados mediante el uso de las siguientes dimensiones temporales:

- dimensión de tiempo de validez –validez del conocimiento-.
- dimensión de tiempo de la transacción. Describe el punto en el tiempo cuando la información fue integrada en la base de datos.

La afinidad entre los conceptos de Data Warehouse y bases de datos temporales puede no ser obvia. Sin embargo, las referencias al tiempo son esenciales en las decisiones de negocio. Como ya se ha indicado Inmon define un Data Warehouse como una arquitectura que soporta la gestión de información "temática", "integrada", "dependiente del tiempo" y "no volátil". Por otro lado, una base de datos temporal se presenta como una base de datos que soporta "tiempo de validez" –el tiempo cuando el hecho es efectivo en la realidad-, o "tiempo de transacción" –el tiempo cuando el hecho es almacenado en la base de datos-, o ambos tiempos. Esta definición excluye "tiempo definido por el usuario" puesto que se trata de un dominio de atributos de tiempo no interpretado gestionado directamente por el usuario y no por la base de datos.

Teniendo en cuenta la definición de Data Warehouse de Inmon, ésta puede ser reescrita en términos de base de datos temporal. En primer lugar, la característica de "dependiente del tiempo" simplemente específica que cada registro en el Data Warehouse es "exacto" respecto a algún momento en el tiempo. Por otro lado, la definición del tiempo de validez expresa cuando un hecho es considerado verdadero en la realidad modelada. Por tanto, ambos conceptos destacan la importancia de mostrar cuando la información es correcta y corresponde exactamente a la realidad. Adicionalmente, la "no volatilidad" hace referencia al hecho de que los cambios en el Data Warehouse son capturados en la forma de "snapshots temporales". En lugar de actualizaciones, se añade un nuevo snapshot al Data Warehouse para reflejar los cambios. Este concepto puede ser claramente asimilado con el de tiempo de transacción, definido como el momento en el que el hecho es actual en la base de datos. Por lo tanto, al definir una base de datos bitemporal como aquella que soporta tanto el tiempo de validez como el de transacción, un Data Warehouse puede considerarse como una base de datos bitemporal conteniendo información integrada y temática con el objeto de ayudar en los procesos de toma de decisión.

Idealmente, por tanto, podría deducirse que el entorno más adecuado para la implantación de un Data Warehouse correspondería a un SGBD temporal. En un

sistema de estas características el soporte al tiempo estaría implícito en el propio SGBD y el lenguaje de consultas contendría funciones específicas temporales para simplificar la manipulación del tiempo.

Sin embargo, la realidad indica que en la actualidad dichos sistemas gestores de base de datos temporales todavía no han dado el salto desde el ámbito investigador al mundo comercial y que por lo tanto, mientras tales tipos de sistemas no estén disponibles y preparados para el entorno empresarial, los Data Warehouse tienen que ser diseñados y construidos en base a los actuales SGBDs, en los cuales, el soporte para el tiempo tiene que ser construido explícitamente en las estructuras de datos y usando el lenguaje de consulta habitual.

En este sentido, la gestión de información temporal puede ser muy complicada utilizando modelos de datos y lenguajes de consulta convencionales. Acomodar la naturaleza time-varying (variable en el tiempo) de la empresa es, en la mayoría de las ocasiones, dejado al albur de los desarrolladores de aplicaciones de base de datos, dando lugar a ineficientes y poco efectivas soluciones ad-hoc que deben ser reinventadas cada vez que se desarrolla una nueva aplicación. El resultado es que esta gestión de la información básicamente se ha convertido en una actividad del tipo prueba-error.

3.1 Objetivos

El objetivo de este trabajo consiste en definir un modelo de gestión del tiempo aplicable a sistemas Data Warehouse que pueda ser desarrollado utilizando las capacidades de los SGBDs relacionales y el SQL disponibles actualmente. El modelo definido debe cumplir los siguientes requisitos:

[REQ-1] **Genérico:** Debe ser un modelo genérico, es decir, totalmente independiente del dominio de aplicación que se esté modelando. El modelo definido no debe ser dependiente o estar diseñado para problemáticas de sistemas o negocios específicos. Lo que se persigue es definir una arquitectura de gestión del tiempo que pueda ser utilizada por cualquier sistema de cualquier tipo de negocio, de la misma forma que cualquier sistema puede utilizar el resto de mecanismos proporcionados por un SGBD, como pueden ser las claves principales, las claves ajenas o la integridad referencial.

Tampoco debe restringir a la totalidad del Data Warehouse a ser temporal, sino que debe permitir la coexistencia de aspectos no temporales y temporales.

[REQ-2] **Completo:** Debe ser capaz de mostrar el estado de cualquier objeto tal como era en cualquier momento de su vida. El modelo definido debe poder capturar todos los cambios de estado de los objetos. La definición de qué se considera “cambio de estado” en cada caso se realizará junto con los especialistas del negocio para lo cual se les solicitará identificar cuáles son

los atributos cuyas evoluciones interesa registrar para posteriormente poder ser analizados a lo largo del tiempo.

[REQ-3] **Tiempo real:** Debe permitir dar respuesta a las preguntas en tiempo real. El modelo definido debe permitir responder las consultas de los usuarios en tiempo real, sin que sea necesaria la intervención de los departamentos de sistemas u operaciones para recuperar datos históricos a partir de sistemas de almacenamiento secundarios.

[REQ-4] **Trazable:** Debe ser capaz de registrar la trayectoria y las manipulaciones realizadas sobre los datos. El modelo definido permitirá capturar todos los cambios de estado de los objetos, tal como se recoge en un requisito anterior. Lo que indica este último requisito es que, además de la historia de los objetos, permitirá capturar la historia de los propios datos, de forma que sea posible conocer en qué momento se realizó la inserción o modificación de los mismos.

La trazabilidad requiere la capacidad de preservar la información exacta en base a la cuál el analista tomó su decisión. Si los eventos antiguos son reemplazados por sus nuevas versiones, las decisiones pasadas no pueden ser ya justificadas.

Esta característica es la que permitirá dar respuesta a la pregunta formulada anteriormente de “por qué el informe del mes pasado mostraba ventas por 100€ y hoy muestra que las ventas fueron sólo 25€”.

Estos requisitos tienen como fin que el modelo sea lo más completo posible, de manera que pueda servir como un marco de referencia para la actividad de diseño de la mayoría de los Data Warehouse.

Por último, la presentación del modelo a lo largo del presente trabajo se realiza de manera gradual mostrando el proceso de pensamiento que ha llevado a cada una de las iteraciones o refinamientos del mismo, partiendo de las propuestas más sencillas hasta las más complejas, identificando sus características e inconvenientes y justificando la transición a la siguiente iteración del modelo hasta alcanzar aquella que satisface la totalidad de los requisitos expuestos.

4 RESOLUCIÓN DEL PROBLEMA

En este capítulo se define un modelo de gestión del tiempo que tiene como objetivo atender a los requisitos indicados en el planteamiento del problema.

Como se verá a lo largo de este capítulo, el **modelo bitemporal** es el que atiende todos estos requisitos y se llegará hasta él de forma gradual, comenzando con la presentación de un modelo que permite una mínima gestión del tiempo e introduciendo sucesivamente nuevos requisitos que irán dando lugar a modelos más completos hasta llegar al modelo final que cumple todos los requisitos definidos para el mismo.

La presentación de los sucesivos modelos se realiza mediante ejemplos que se han denominado patrones, constituidos por una estructura de datos y unas restricciones semánticas que se deben forzar para que el modelo esté correctamente implementado.

Una vez alcanzado el modelo objetivo, se pasa a recopilar el resto de información necesaria para tener definido un modelo de gestión del tiempo completo, el cual debe incluir estructuras de datos, operaciones de definición, acceso y manipulación de las mismas y restricciones de integridad.

4.1 REPRESENTACIÓN DE PATRONES

En este trabajo se presenta una serie de patrones o modelos de gestión del tiempo, para cada uno de los cuales se comienza mostrando su estructura de datos, es decir, los campos necesarios para implementar dicho patrón. Estos campos se muestran sobre una tabla de ejemplo, en la que se han sombreado las columnas que constituyen la estructura de datos y se han dejado en blanco el resto de columnas que pertenecen a la tabla ejemplo y que se repiten en cada patrón. Como ejemplo se utiliza una tabla de líneas telefónicas de una empresa de telecomunicaciones con las siguientes columnas:

- **nro_linea:** es el identificador de la línea telefónica, podría ser el número de teléfono, aunque a lo largo de este trabajo se utiliza por abreviar un código más corto, de cuatro caracteres.
- **nro_cliente:** es el identificador del cliente al que pertenece la línea.
- **tipo_contrato:** es el tipo de contrato o “plan” que tiene asociada la línea y que puede cambiar a lo largo del tiempo.

A continuación, para cada uno de los patrones se describe como se comportan ante distintos escenarios de operación. Cada escenario muestra el resultado de hacer una inserción, actualización o borrado sobre una tabla construida de acuerdo a ese patrón.

Inicialmente, estas transacciones son muy simples y afectan físicamente a una única fila en una tabla. Pero gradualmente se van considerando escenarios más completos, en los

que lo que parece una transacción desde el punto de vista lógico da como resultado que se vean afectadas varias filas.

La Figura 5.1 muestra el diagrama que se utiliza para representar los escenarios:

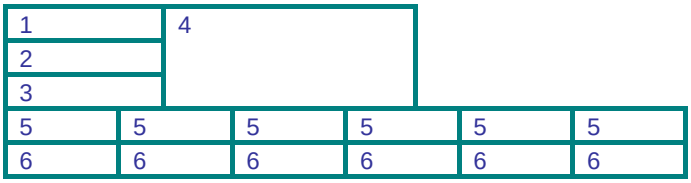


Figura 4.1: Representación de un escenario

Las celdas numeradas en el diagrama de representación contienen la información siguiente:

1. Nombre del escenario.
2. Tabla utilizada para este escenario.
3. Definición de la tabla.
4. Descripción del escenario.
5. Nombre de las columnas de la tabla.
6. Una o más filas de la tabla.

Básicamente, las celdas 1 a 4 contienen metadatos necesarios para definir el escenario, las celdas 5 son los nombres de las columnas de la tabla en cuestión y finalmente, puede haber una o más filas etiquetadas como 6, representando cada una de ellas una fila en una tabla.

La convención para distinguir diferentes tipos de columnas en estas tablas ilustrativas es la siguiente:

- Las columnas que forman la clave principal se colocan en primer lugar, en negrita y con la etiqueta (CP).
- Las columnas que son claves ajenas van a continuación, subrayadas y con la etiqueta (CA).
- A continuación se muestra el resto de columnas.
- Y por último se muestran las columnas de metadatos, como por ejemplo la fecha de creación y la de última actualización.

Finalmente, para facilitar su identificación, se muestran resaltados en negrita y con doble borde los datos afectados por cada operación.

A continuación de los escenarios se presenta una lista de restricciones semánticas que deben ser respetadas para que el patrón funcione correctamente. La mayoría de estas restricciones se pueden implementar mediante el uso de disparadores, procedimientos almacenados o código de aplicación. Una muestra de lo "especial" y compleja que es la gestión del tiempo es el hecho de que los mecanismos de integridad referencial de los SGBDs más habituales pueden hacer muy poco para forzar la semántica de estos patrones.

Se considera necesario destacar la importancia de listar explícitamente las restricciones semánticas de cada patrón puesto que junto con la estructura de datos estas restricciones constituyen la definición formal completa de cada patrón y sólo se tiene adecuadamente implementado un patrón si:

a) se incluyen las columnas definidas para el patrón en las tablas en las que se desea incluir gestión del tiempo y

b) se fuerza cada una de estas restricciones semánticas en las operaciones realizadas sobre esas tablas.

Los escenarios muestran cómo trabaja cada patrón, pero las restricciones y las estructuras de datos *son* los patrones.

La tabla siguiente recoge la lista de patrones que se incluyen en este trabajo:

Patrón	**Nombre**	**Descripción**
1	Actualización	Se destruye la historia al sobrescribir y eliminar filas pero las fechas de creación y última actualización permiten cierto nivel de información histórica
2	Actualización con borrado lógico	Patrón 1 pero marcando la fila como borrada en lugar de eliminarla
3	Versionado unitemporal	Versiones de objetos indicando el periodo de validez de cada versión
4	Versionado bitemporal	Versiones de objetos en las que las fechas de actividad del sistema (inserción, actualización y borrado) y las fechas de validez para el negocio (inicio, fin) son representadas de manera independiente

Figura 4.2: Lista de Patrones

El patrón 4, de **versionado bitemporal**, es el patrón objetivo a cuya presentación se pretende llegar de forma gradual a lo largo de este trabajo, por considerar que atiende los requisitos definidos en el planteamiento del problema.

Dos son las razones para considerar varios patrones o métodos de gestión del tiempo. La primera es que cada patrón puede ser utilizado para satisfacer un conjunto de requisitos de negocio diferentes, más o menos exigentes en cuanto a gestión del tiempo. Los patrones iniciales son menos complejos que los siguientes, y en algunas ocasiones se puede preferir utilizar un patrón más sencillo cuando los requisitos del sistema lo permitan.

La segunda razón para considerar varios patrones es ir construyendo de forma gradual un patrón de gestión del tiempo lo suficientemente completo como para atender la mayoría de las necesidades de representación temporal en un Data Warehouse.

Por tanto una característica de este trabajo es su aproximación gradual al patrón objetivo. A menudo, los modelos de datos se presentan como objetos completados, sin hacer un esfuerzo para explicar los procesos de pensamiento y las decisiones que llevaron a ellos. De esta forma se pretende no sólo presentar un modelo de gestión del tiempo, sino proporcionar las herramientas básicas necesarias para entender el porqué de dicho modelo y, por supuesto, para construir otros modelos alternativos. De hecho, los patrones que se presentan en este trabajo son sólo algunos ejemplos elegidos para seguir un hilo argumental hasta el modelo objetivo, pero se pueden definir otros modelos diferentes, así como otras implementaciones diferentes para los modelos propuestos.

Antes de comenzar con la presentación de los patrones, se muestra una clasificación de los diferentes tipos de historia que se pueden considerar, cada uno de los cuales es aplicable en uno u otro sistema, en función de los diferentes requisitos especificados sobre sus necesidades de acceso a los datos históricos o temporales.

4.2 TIPOS DE HISTORIA

Cuando los profesionales de las Tecnologías de la Información hablan de historia en bases de datos, pueden tener en mente cosas diferentes. En la Figura 4.3 se muestra una taxonomía de estos diferentes tipos de historia.

Historia: se trata de datos históricos almacenados en tablas de una base de datos relacional. Las dos principales categorías para la historia de base de datos son la historia reconstruible y la historia consultable.

- **Historia reconstruible:** datos sobre el estado pasado de algo, obtenidos restaurando un archivo de backup y aplicando después las transacciones de actualización capturadas en un archivo de log del SGBD, desde ese punto hasta el instante deseado. Requiere la intervención del personal de TI y por tanto no es en tiempo real. La historia reconstruible se utiliza desde que se utilizan los ordenadores. Con copias de respaldo periódicas de archivos o bases de datos y un log de transacciones es posible recuperar el estado de los archivos o bases de datos desde cualquier instante pasado. No existen datos históricos recuperables

por otros métodos que no puedan ser también recuperados mediante una copia de respaldo y ficheros de log.

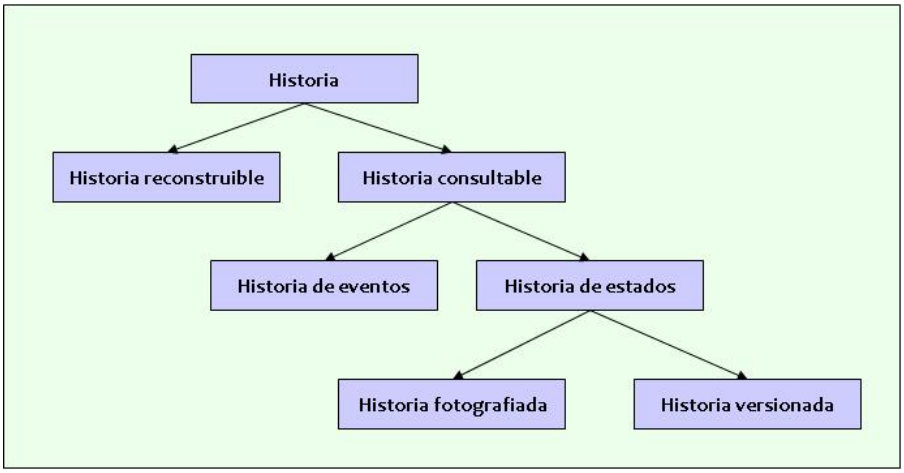


Figura 4.3: Taxonomía de Historia gestionada en SGBDs

Entre los diferentes métodos de gestión de datos temporales, la historia reconstruible es la que más tiempo precisa y tiene el coste más elevado. Los servicios de operaciones tienen que restaurar los archivos de respaldo si no se encuentran online. También se debe acceder a los logs de transacciones, se debe localizar el punto de inicio correcto y aplicar las transacciones hacia delante hasta el instante requerido. Después, se deben ejecutar consultas o informes para recuperar los datos deseados. Si se trata de consultas o informes de producción estarán apuntando a los archivos, tablas o bases de datos actuales, por lo que antes de utilizarlos para recuperar datos históricos se deben redirigir a la copia histórica que se ha creado, proceso que normalmente supone cambio de nombres de bases de datos, tablas o incluso columnas. Una vez recuperados los datos, siempre es necesario un trabajo posterior de “limpieza” para volver a dejar todo en su estado original.

Es habitual que el coste de estas operaciones se mida en hombres*día y que lleve varios días hasta completarse. Si se trata de grandes volúmenes de datos significa que la solicitud no se hace con frecuencia y si los consumidores de esos datos están de acuerdo en esperar varios días para obtener el resultado, entonces este método de gestión de datos temporales es perfectamente aceptable. Una auditoría anual sería un ejemplo en el que aplican estas circunstancias. Pero, por ejemplo, este método no sería válido para responder a un cliente cuando llama por teléfono a un centro de atención al cliente.

- **Historia consultable:** datos sobre el estado pasado de algo, obtenidos mediante una consulta SQL, sin la necesidad de restaurar archivos de backup y reaplicar transacciones capturadas en el archivo de log del SGBD. Dado que la consulta puede ser ejecutada directamente, este tipo de historia puede ser en tiempo real. La historia consultable se puede almacenar bien como un registro de eventos o como un registro de estados.
  - **Historia de eventos:** datos sobre eventos que han alterado el estado de las cosas. Estos datos se capturan como transacciones en tablas. Este método consiste en almacenar el estado inicial de algo y posteriormente almacenar todas las transacciones que lo van actualizando. Es de gran utilidad para gestionar balances, es decir, relaciones en las que se califican las partes relacionadas con una o varias métricas que definen la relación. Por ejemplo un almacén que comienza cada año con las métricas de cantidades en almacén y cantidades bajo pedido para cada uno de sus productos y a partir de ese punto va almacenando los pedidos y entregas de sus proveedores. Se puede resolver una consulta sobre cantidades en almacén y bajo pedido de cualquier producto partiendo de los balances iniciales y aplicando las diferentes transacciones de ese producto.
  - **Historia de estados:** datos históricos capturados como copias del estado actual de un objeto, bien periódicamente o en respuesta a un evento de actualización específico. Excepto para obtener balances, el mejor método consultable para gestionar datos temporales es mantener una historia de estados. Se trata de un método de gestión de datos temporales que mantiene todas las inserciones y borrados con sus dos imágenes, anterior y posterior, para cada modificación.

    La historia de estados se aplica para todas aquellas cosas que tienen estados, es decir, que pueden cambiar a lo largo del tiempo. En contraposición, un evento, como una llamada telefónica, no puede cambiar, una vez que ocurre permanece para siempre en el pasado. Pero un cliente, como propietario de una línea telefónica, sí puede hacerlo.

    La historia de estados se puede almacenar bien como fotografías o como versiones.

    - **Historia fotografiada:** conjunto coordinado de copias en una base de datos relacional. Las fotografías (o *snapshots*) se toman habitualmente considerando la base de datos completa, o también a veces incluyendo únicamente un subconjunto de tablas relacionadas semánticamente. Los resultados de las fotografías se pueden almacenar en la misma base de datos en la que se encuentran los datos originales o en una base de datos diferente. Normalmente, la

primera opción se denomina mantenimiento de un conjunto de "tablas fotografiadas" y la segunda opción se denomina actualización de un Data Warehouse. Estas fotografías se toman a intervalos regulares de tiempo programados. En consecuencia, las fotografías pierden cualquier actualización que haya sido sobrescrita por actualizaciones posteriores del mismo dato realizadas antes de la siguiente fotografía. Otro inconveniente de las fotografías es que se trata de una forma poco eficiente de registrar la historia, ya que se crean copias de todas las filas, hayan cambiado o no.

Existe una única cosa que las fotografías pueden hacer mejor que cualquier otro método: mantener una copia exacta de cómo estaban los datos en el momento de tomar la fotografía, lo que hace que las fotografías sean la mejor opción para las dos situaciones siguientes:

a) Si las fotografías se toman en instantes representativos para el negocio, como el fin de mes, entonces esas fotografías estarán disponibles para volver a ejecutar informes y obtener exactamente los mismos resultados de aquel instante.

b) Se puede utilizar una serie de fotografías de fin de periodo para obtener datos de tendencias en un proceso de data mining. Por ejemplo, a partir de fotografías mensuales de una tabla de clientes se pueden obtener perfiles mensuales del "cliente típico" y a partir de esos perfiles, la compañía podría identificar cambios en la edad, ingresos u otros parámetros de su "cliente base".

- **Historia versionada:** actualizaciones lógicas de filas individuales, implementadas sin sobrescribir datos, "retirando" la versión actual del objeto y reemplazándola con una nueva versión que contiene los datos actualizados del objeto.

  Las versiones se crean en función de las necesidades, es decir, cada vez que tiene lugar una actualización lo suficientemente importante como para ser versionada. En consecuencia, las versiones no pierden ninguna actualización. También se trata de una forma eficiente de registrar la historia puesto que sólo se crean nuevas filas cuando ocurre un cambio versionable.

  A pesar de que los métodos de versionado conservan las imágenes anterior y posterior de las filas que cambian, esto no significa que estos métodos den como resultado un crecimiento explosivo de los requisitos de almacenamiento (como sí puede suceder con la historia fotografiada).

> Al contrario que las fotografías, las versiones no son grandes consumidoras de disco. Ambas hacen copia de datos, pero el versionado copia sólo los datos cuando éstos han cambiado. También, al contrario que las fotografías, las versiones sí son confiables, ya que registran todos los cambios de los datos versionados y no solamente los cambios que han permanecido el tiempo suficiente para ser registrados en la siguiente fotografía.

A continuación se presenta el primer patrón para gestión del tiempo, el cual es tomado como punto de partida para continuar con la introducción gradual del resto de patrones, cada uno más completo que el anterior hasta llegar al patrón objetivo de este trabajo.

4.2.1 PATRÓN 1: Actualización

En este patrón las actualizaciones destruyen la historia al sobrescribir el valor previo de las filas que se actualizan pero, en la mayoría de los casos, esta es la forma en que se manipulan los datos. En estas situaciones, el negocio no requiere que los estados anteriores a la actualización se encuentren disponibles para ser consultados. Para este tipo de datos y requisitos de negocio, la necesidad de los datos históricos es tan poco frecuente, que es suficiente la solución de mantener historia reconstruible.

4.2.1.1 Estructura de datos

La estructura de datos definida para el patrón 1 es la siguiente:

nro_linea (CP)	
nro_cliente (CA)	
tipo_contrato	
…	
fx_alta	*(Fecha de la inserción de la fila)*
fx_ult_act	*(Fecha de la última actualización de la fila)*

La columna `fx_alta` es la fecha en que se insertó la fila. Y la columna `fx_ult_act`, cuyo valor se debe actualizar cada vez que se actualiza una fila, es la fecha de última actualización de esa fila en particular.

Cada fila representa el estado actual una línea telefónica. Cuando ese estado cambia, una transacción de actualización modifica la fila para reflejar el cambio, de manera que se pierden los valores anteriores al ser sobrescritos por los nuevos.

Incluso en sistemas en los que no existen requisitos especiales sobre mantenimiento de historia consultable, es una práctica bastante común incluir las fechas de creación y última actualización de la fila. Estas dos fechas proporcionan un nivel mínimo de tratamiento de datos históricos.

4.2.1.2 Escenarios

Los siguientes escenarios ilustran el comportamiento del patrón 1. Tal como se ha indicado anteriormente, en estos escenarios se utiliza como ejemplo una tabla de líneas telefónicas de una empresa de telecomunicaciones.

Escenario 1.1: Inserción de una nueva línea telefónica

El 01/01/04 se da de alta una línea telefónica para el cliente Juan Dato en la tabla de líneas telefónicas de Tele Acme. La línea empieza a estar operativa en la misma fecha en que se inserta esta fila en la tabla de líneas telefónicas y en cualquier consulta o informe que se genere partir del día 01/01/04, aparecerá la línea de Juan Dato.

El escenario 1.1 muestra el resultado de introducir la nueva línea telefónica en la tabla de líneas telefónicas:

Escenario 1.1				La tabla después de insertar la línea telefónica L617
LINEA				
Cada fila de esta tabla representa una línea telefónica de Tele Acme				
nro_linea (CP)	nro_cliente (CA)	tipo_contrato	fx_alta	fx_ult_act
L617	**C466**	**PP**	**01/01/04**	**{null}**

La clave principal `nro_linea` es el identificador único de la línea.

La clave ajena `nro_cliente` contiene el identificador del cliente al que pertenece la línea telefónica L617.

El tipo de contrato para esta línea telefónica es PP (Pre Pago). La primera columna de metadatos (`fx_alta`) indica que la fila fue insertada el 01/01/04. Y el valor `{null}` en la segunda columna de metadatos (`fx_ult_act`) indica que esta fila no ha sido actualizada.

Escenario 1.2: Actualización de una línea telefónica

El 01/03/05, Juan Dato cambia su tipo de contrato de PP a CF (Contrato Familia). Tras registrar este cambio en la tabla de líneas telefónicas, la fila correspondiente a la línea telefónica de Juan Dato presenta el aspecto siguiente:

Escenario 1.2				La tabla después de actualizar la línea telefónica L617
LINEA				
Cada fila de esta tabla representa una línea telefónica de Tele Acme				
nro_linea (CP)	nro_cliente (CA)	tipo_contrato	fx_alta	fx_ult_act
L617	C466	**CF**	01/01/04	**01/03/05**

La `fx_alta` indica que la fila se insertó el día 01/01/04 y la `fx_ult_act` indica que esa fila cambió el día 01/03/05.

El uso de las fechas de creación y última actualización está bastante generalizado y con estas dos fechas ya no se carece por completo de metadatos temporales. No obstante, hay mucha información que no es posible saber. No se sabe cuántas veces ha cambiado la fila antes del 01/03/05 (si es que ha cambiado alguna vez), ni qué columna o columnas cambiaron en actualizaciones anteriores. Tampoco es posible saber qué columna o columnas cambiaron el 01/03/05. Incluso si fuera posible saber que la columna que cambió fue el tipo de contrato, no se puede saber qué valor tenía antes.

Por tanto, se ha perdido información. Más exactamente, utilizando la taxonomía presentada en la Figura 5.3, se ha perdido información de la historia consultable, ya que la historia reconstruible siempre estará disponible. En este caso, para recuperar la historia reconstruible de la línea telefónica L617, será necesario revisar el fichero de log de la base de datos para recuperar la inserción inicial y las actualizaciones posteriores en la tabla de líneas telefónicas, comenzando el 01/01/04 y siguiendo hacia delante en orden cronológico.

Escenario 1.3: Borrado de una línea telefónica

El 01/06/06 Juan Dato da de baja su línea telefónica con Tele Acme y este hecho se registra, utilizando el patrón 1, borrando físicamente la fila correspondiente en la tabla de líneas telefónicas. Tras registrar este cambio la tabla de líneas telefónicas queda de la siguiente manera:

Escenario 1.3				La tabla después de borrar la línea telefónica L617
LINEA				
Cada fila de esta tabla representa una línea telefónica de Tele Acme				
nro_linea (CP)	nro_cliente (CA)	tipo_contrato	fx_alta	fx_ult_act

Como se puede observar, tras borrar la línea telefónica, se pierde toda la información que hasta este momento se tenía sobre ella.

4.2.1.3 Restricciones semánticas

Para que el patrón 1 esté correctamente implementado se deben forzar las siguientes restricciones a la hora de modificar la tabla:

- **{RS 1-1}** Siempre que se inserta una fila se debe asignar a `fx_alta` la fecha actual.
- **{RS 1-2}** Siempre que se inserta una fila se debe asignar {`null`} a la fecha de última actualización, `fx_ult_act`.
- **{RS 1-3}** Siempre que se actualiza una fila se debe sobrescribir el valor de `fx_ult_act` con la fecha actual.

Con la aplicación de estas restricciones semánticas se puede conocer la fecha en que se insertó la fila y la fecha en que fue actualizada por última vez.

4.2.1.4 Transición al siguiente patrón

Se puede observar que al trabajar con ese patrón, tras la actualización mostrada en el escenario 1.2 se podía conocer el tipo de contrato de la línea de Juan Dato y ciertos metadatos pero después de realizar el borrado del escenario 1.3, se ha perdido todo y no se puede decir siquiera que Juan Dato haya tenido una línea telefónica contratada con Tele Acme.

En ocasiones puede que no importe perder esta información. A veces lo único que se necesita conocer en tiempo real es cómo son las cosas en el momento actual. Pero con frecuencia, los negocios requieren una cantidad mínima de historia disponible en tiempo real como, en este caso, información de clientes que ya no tienen contratado un servicio.

Puesto que el borrado físico destruye incluso la mínima información histórica que proporciona el patrón 1, se suele solicitar a los departamentos de TI alguna forma de preservar ese mínimo nivel de información histórica. La forma más frecuente de hacerlo es implementar un borrado lógico en lugar de un borrado físico.

4.2.2 PATRÓN 2: Actualización con borrado lógico

El patrón 2 supone una mejora sobre el patrón 1 consistente en sustituir el borrado físico por un borrado lógico.

En ocasiones, las filas eliminadas con borrado lógico se archivan fuera de la tabla actual, normalmente en función de su antigüedad. Por ejemplo, puede existir una regla de negocio que diga que “al final de cada mes, se deben archivar todas las filas eliminadas con borrado lógico hace más de 25 meses”. Volviendo a la taxonomía

presentada en la Figura 5.3, esta es una regla que establece bajo qué condiciones una fila pasa de ser consultable a ser reconstruible.

4.2.2.1 Estructura de datos

La estructura de datos definida para el patrón 2 es la siguiente:

nro_linea (CP)	
nro_cliente (CA)	
tipo_contrato	
…	
fx_alta	*(Fecha de la inserción de la fila)*
fx_ult_act	*(Fecha de la última actualización de la fila)*
fx_baja	*(Fecha del borrado lógico de la fila)*

Además de las columnas `fx_alta` y `fx_ult_act` utilizadas en el patrón 1, para poder llevar a cabo borrados lógicos es necesaria una columna adicional, un indicador de borrado. A menudo esto se implementa mediante el uso de un indicador booleano que tome, por ejemplo, valor “S” si la fila se ha eliminado mediante un borrado lógico y valor “N” en caso contrario. Y la fecha de borrado se introduce en `fx_ult_act`.

Otra opción es introducir una nueva columna para la fecha de borrado, `fx_baja`. Con esta opción no es necesario el indicador de borrado ya que las filas eliminadas con borrado lógico son aquellas cuya fecha de borrado no es nula.

En este caso se ha optado por utilizar la segunda opción porque preserva y no sobrescribe la información de metadato de fecha de última actualización.

4.2.2.2 Escenarios

Se muestra a continuación el escenario de borrado lógico, que es el único que cambia respecto al patrón anterior. Si en el patrón 1 se hacía un borrado físico, en este caso se realiza un borrado lógico, marcando la fila como borrada pero manteniéndola en la tabla.

Escenario 2.1: Borrado de una línea telefónica

El 01/06/06 Juan Dato da de baja su línea. A partir de esa fecha ya no tendrá contratada esa línea telefónica con Tele Acme. En esta ocasión, se registra este evento haciendo un

borrado lógico de la fila correspondiente a la línea de Juan Dato en la tabla de líneas telefónicas. Tras registrar este cambio, la tabla queda como sigue:

Escenario 2.1 LINEA Cada fila de esta tabla representa una línea telefónica de Tele Acme			La tabla después de eliminar con borrado lógico la línea telefónica L617.		
nro_linea (CP)	nro_cliente (CA)	tipo_contrato	fx_alta	fx_ult_act	fx_baja
L617	C466	CF	01/01/04	01/03/05	**01/06/06**

Es decir, el borrado lógico se traduce a una actualización física de la fila, para actualizar únicamente el valor de su fecha de baja `fx_baja`.

4.2.2.3 Restricciones semánticas

Para que el patrón 2 esté correctamente implementado se deben forzar las siguientes restricciones a la hora de modificar la tabla:

- **{RS 2-1}** Siempre que se inserta una fila se debe asignar a `fx_alta` la fecha actual.
- **{RS 2-2}** Siempre que se inserta una fila se debe asignar {`null`} a la fecha de última actualización, `fx_ult_act`.
- **{RS 2-3}** Siempre que se inserta una fila se debe asignar {`null`} a la fecha de borrado lógico, `fx_baja`.
- **{RS 2-4}** Siempre que se actualiza una fila, excepto en el caso de tratarse de una actualización que corresponda al borrado lógico de la fila, se debe sobrescribir el valor de `fx_ult_act` con la fecha actual.
- **{RS 2-5}** Siempre que se actualiza una fila con motivo de un borrado lógico se debe asignar a `fx_baja` la fecha actual.

Como se puede observar se ha partido de las restricciones del patrón 1 y se han realizado las modificaciones necesarias para adaptarlas a este nuevo patrón. En concreto, las adaptaciones necesarias son relativas a la nueva columna `fx_baja` y su nueva funcionalidad asociada de borrado lógico. La restricción {RS 2-3} especifica el valor que debe tomar `fx_baja` al insertar una fila, al igual que las dos primeras, comunes a ambos patrones, lo hacen para las fechas de inserción y última actualización. Por último la restricción {RS 1-3} se desdobla en este patrón en dos restricciones, la {RS 2-4} y la {RS 2-5}, ya que ahora existe una excepción, no se debe modificar el valor de la fecha de última actualización cuando se trata de una actualización que corresponda al borrado lógico de la fila. En este caso, tal como especifica la restricción

{RS 2-5} será la columna correspondiente a la fecha de borrado lógico la única que actualizará su valor.

De esta forma, cada fila mostrará siempre información sobre la fecha en que fue insertada, la fecha en que se modificó por última vez y la fecha en que fue "eliminada" de la tabla.

4.2.2.4 Transición al siguiente patrón

Como se ha visto, incluso si se utiliza el patrón 2 de actualización con borrado lógico, todavía hay dos formas importantes en las que se puede perder historia.

En primer lugar, en el momento en que se realiza la actualización, la fila tal como se insertó originalmente se pierde, ya que la actualización sobrescribe la fila. Se pierde información que era cierta en el momento en que la línea fue dada de alta. Y en segundo lugar, se pierden todas las actualizaciones excepto la última, incluso es imposible saber si hubo o no actualizaciones previas. A veces esto puede ser aceptable, pero otras veces no lo es.

4.3 HISTORIA VERSIONADA

El modelo de gestión del tiempo objetivo de este trabajo considera unos requisitos de historia más exigentes que los que pueden soportar los patrones 1 y 2. Tal como se ha indicado en la definición del problema, el modelo de gestión del tiempo definido debe poder "*mostrar el estado de cualquier objeto tal como era en cualquier momento de su vida*". Y los patrones 1 y 2 no pueden hacer esto ya que tal como se ha visto, al actualizar están sobrescribiendo la información anterior, manteniendo únicamente el último estado o estado actual de los objetos.

Para poder "*mostrar el estado de cualquier objeto tal como era en cualquier momento de su vida*", es preciso retener el estado de los objetos antes de su actualización además del estado posterior a la actualización, es decir, es preciso mantener la historia de los objetos. Y entre los distintos tipos de historia vistos en el apartado "4.2 TIPOS DE HISTORIA", es necesario determinar en primer lugar cuál de ellos utilizar para el modelo de gestión de tiempo que se está construyendo.

Tal como indica la taxonomía de historia de la Figura 4.4, la primera elección consiste en decidir si se va a mantener la historia reconstruible o consultable. Puesto que un requisito del modelo de gestión del tiempo dice que "*debe permitir dar respuesta a las preguntas en tiempo real*", la historia reconstruible queda descartada.

Continuando por la rama seleccionada, correspondiente a la historia consultable, la siguiente elección consiste en decidir si se va a mantener la historia como un registro de eventos o de estados. Se podría mantener la historia de las líneas basada en eventos añadiendo una fila a la tabla de líneas telefónicas cada vez que se da de alta una nueva

línea telefónica y manteniendo después una tabla de histórico de transacciones en la que cada fila correspondería a una actualización o un borrado de la línea.

Sin embargo, la historia basada en eventos resulta más apropiada para gestionar cambios en valores numéricos o en relaciones entre objetos, valores como contadores y cantidades. La historia basada en eventos no resulta apropiada para gestionar cambios en valores no numéricos de objetos.

Puesto que la historia fotografiada puede perder actualizaciones, y uno de los requisitos del modelo de gestión del tiempo definido en este trabajo es que debe ser capaz de "*mostrar el estado de cualquier objeto tal como era en cualquier momento de su vida*", la historia fotografiada no se considera apropiada.

Por tanto, para cumplir satisfactoriamente los requisitos que se han establecido, se debe mantener *historia versionada* de los cambios de estado del objeto.

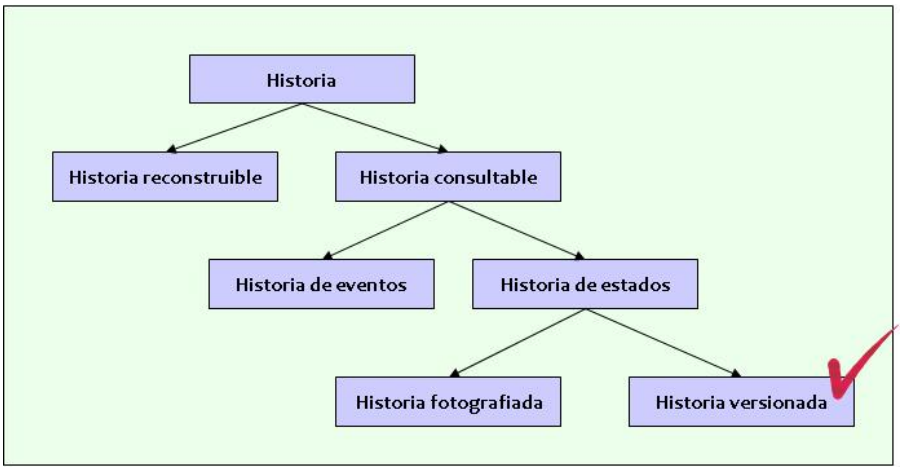


Figura 4.4: Historia versionada

Llegados a este punto, es necesario destacar que la utilización de la historia versionada supone un importante cambio semántico en cuanto a lo que representa el contenido de las tablas. Si hasta ahora en las tablas normales las filas representan *objetos*, en las tablas versionadas las filas representan *versiones de objetos*. A partir de este momento, semánticamente se va a seguir gestionando objetos, pero la tabla que representa objetos desaparece. El objeto existe implícitamente a través de sus versiones.

Antes de empezar con el primer patrón de historia versionada, debe estar claro que, si bien lo que se inserta, actualiza y borra físicamente son versiones de objetos, lo que se está insertando, actualizando y borrando de forma lógica en las tablas versionadas son

los objetos que se versionan. La inserción física de una versión inserta, actualiza o borra lógicamente un objeto de la siguiente forma:

- La inserción física de la primera versión de un objeto es la inserción lógica de dicho objeto.
- La inserción física de las siguientes versiones de ese objeto, que no sean etiquetadas como un borrado lógico de la línea, son actualizaciones lógicas del objeto.
- La inserción física de una versión etiquetada como borrado lógico, supone el borrado lógico del objeto.

En una tabla convencional, no versionada, una fila representa un objeto (en una tabla de líneas telefónicas cada fila representa una línea telefónica). Pero en la tabla de versiones de líneas telefónicas, ninguna fila representa una línea telefónica y cada fila representa lo que se conoce sobre como era la línea telefónica durante un periodo de tiempo determinado. En consecuencia, las acciones sobre las líneas telefónicas no se corresponden una a una con acciones sobre filas de la tabla de versiones de líneas telefónicas.

Esto significa que en lugar de la tabla convencional de líneas telefónicas usada hasta ahora, se va a pasar a utilizar una *tabla versionada*, la tabla de versiones de líneas telefónicas. No se mantiene una tabla de líneas telefónicas y se añade una tabla de versiones de líneas telefónicas, sino que se sustituye la tabla de líneas telefónicas por la *tabla versionada* de versiones de líneas telefónicas.

A continuación se pasa a describir los restantes patrones que se exponen en este trabajo, siendo todos ellos ya patrones de versionado.

4.3.1 PATRÓN 3: Versionado unitemporal

El versionado unitemporal es el que considera una única dimensión de tiempo con la que calificar cada una de las filas de una tabla.

Las dos diferentes dimensiones que se consideran habitualmente para construir modelos de gestión del tiempo son:

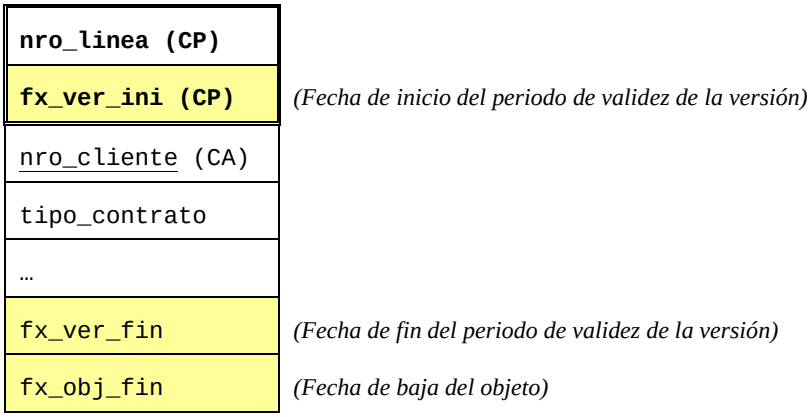
- **Tiempo de validez:** Periodo de tiempo durante el cual un hecho de la base de datos es válido en la realidad modelada.
- **Tiempo de transacción:** Periodo de tiempo durante el cual una fila es considerada válida en la base de datos.

Generalmente, cuando sólo se usa una dimensión, suele corresponder con el tiempo de validez. Y en el caso de incorporar ambas dimensiones nos encontramos con un modelo bitemporal, que se verá más adelante al tratar el patrón 4.

En el patrón 3 se considera una tabla unitemporal con tiempo de validez. Para ello se añaden dos columnas, la primera para especificar cuando una fila comienza a ser válida para el negocio y la segunda para especificar cuándo deja de serlo. El periodo comprendido entre ambas fechas se denomina *periodo de validez* de la fila.

4.3.1.1 Estructura de datos

La estructura de datos definida para el patrón 3 es la siguiente:



Como se puede observar, al introducir el concepto de versionado podrán existir en la tabla varias filas con el mismo identificador de objeto (correspondientes a las diferentes versiones del objeto) por lo que es necesario ampliar la clave principal añadiendo la fecha de inicio de versión, de forma que todas las versiones de un mismo objeto tengan el mismo identificador de objeto y se diferencien entre ellas mediante la fecha de inicio de versión.

Siguiendo con el ejemplo, cada fila en esta tabla representa, para una línea telefónica concreta, el estado de dicha línea telefónica que comenzó a ser efectivo en la fecha `fx_ver_ini`. Cuando ese estado deja de ser válido, porque ha habido un cambio de estado, se actualiza la fila colocando la fecha en que deja de ser válido en `fx_ver_fin` y se inserta, salvo que se trate de la baja del objeto, la nueva versión con su `fx_ver_ini`, reflejando el nuevo estado.

Otra alternativa consiste en introducir únicamente la fecha de inicio del periodo de validez de la versión (`fx_ver_ini`) y no introducir la correspondiente fecha de fin (`fx_ver_fin`). De esa forma la fecha de fin de cada versión está implícita, tratándose siempre del instante inmediatamente anterior a la fecha de inicio de la versión siguiente del mismo objeto. Sin embargo, un patrón con la fecha de fin de versión inferida supone posteriormente una complicación adicional a la hora de escribir el SQL para las consultas, por lo que en general se prefiere utilizar los dos campos e indicar explícitamente una fecha de inicio de versión y una fecha de fin de versión.

Adicionalmente, para diferenciar las versiones que suponen un borrado lógico del objeto se podría utilizar un indicador de borrado, pero una fecha de borrado proporciona más información que un indicador de borrado, por lo que en este caso se prefiere utilizar esta segunda opción y se añade para ello la fecha de fin de objeto (`fx_obj_fin`).

4.3.1.2 Escenarios

Se recrea a continuación la serie de eventos descrita en el patrón 1, comenzando con el evento de creación de la línea telefónica de Juan Dato, que tiene lugar el día 01/01/04.

Escenario 3.1: Inserción de una nueva línea telefónica

La inserción física de la versión inicial de un objeto es también la inserción lógica de un nuevo objeto. Siguiendo el ejemplo, la inserción física de la versión inicial de una línea telefónica es también la inserción lógica de una nueva línea telefónica. Este sería el resultado de insertar la línea telefónica de Juan Dato el 01/01/04:

Escenario 3.1 VERSION_LINEA Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme			La tabla después de insertar la línea telefónica L617		
nro_linea (CP)	**fx_ver_ini (CP)**	nro_cliente (CA)	tipo_contrato	fx_ver_fin	fx_obj_fin
L617	01/01/04	C466	PP	{9999}	{null}

La clave principal de la versión de la línea está formada por el identificador único de la línea (`nro_linea`) más la fecha de inicio de la versión (`fx_ver_ini`). Todas las versiones de la misma línea tienen el mismo número de línea y se diferencian entre ellas por su fecha de inicio.

Al insertar la versión inicial, se ha creado la línea telefónica L617, efectiva desde el 01/01/04 y hasta una "fecha no determinada". La fecha `{9999}` se utiliza para representar una fecha en el futuro y que será concretada en algún momento pero que en este instante no se puede conocer. (ver apartado "4.4.2.3 Estrategia de representación de la "fecha no determinada"").

A continuación se ve cómo se representa en la tabla de versiones de líneas telefónicas la primera actualización.

Escenario 3.2: Actualización de una línea telefónica

El 01/03/05, Juan Dato cambia su línea telefónica de PP a CF (Contrato Familia). Tras registrar el cambio en la tabla de versiones de líneas telefónicas, ésta queda como se muestra a continuación:

Escenario 3.2

VERSION LINEA

Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme

La tabla después de actualizar la línea telefónica L617

nro_linea (CP)	**fx_ver_ini (CP)**	nro_cliente (CA)	tipo_contrato	fx_ver_fin	fx_obj_fin
L617	01/01/04	C466	PP	**01/03/05**	{null}
L617	**01/03/05**	**C466**	**CF**	**{9999}**	**{null}**

La norma de representación de periodos utilizada a lo largo de este trabajo es de intervalo “cerrado-abierto”, es decir, la fecha de inicio está incluida en el intervalo pero la fecha de fin está excluida. (ver apartado “4.4.2.2 Estrategia de representación de periodos”)

Como se puede observar a la vista de los datos afectados por esta operación, una transacción lógica de actualización de una línea telefónica se traduce a dos operaciones físicas, una actualización y una inserción, en la tabla de versiones de líneas telefónicas:

[1] En primer lugar se cierra la versión actual, mediante la actualización física de la fecha de fin de versión (`fx_ver_fin`).

[2] A continuación se inserta la que a partir de ahora será la versión actual, es decir, con una “fecha no determinada” como fecha de fin de versión.

Con este patrón, tras la actualización se sabe que la línea tuvo como tipo de contrato PP desde el 01/01/04 hasta el 28/02/05 exactamente puesto que, después de este cambio, ya se dispone de una fecha de fin para la versión inicial. Se sabe también que el día 01/03/05 se modificó su tipo de contrato pasando de PP a CF, y que éste es el vigente actualmente.

Se observa que se ha actualizado físicamente una fila en la tabla de versiones de líneas telefónicas. Hasta el 01/03/05, la fila con `fx_ver_ini` 01/01/04 tenía valor `{9999}` en la columna `fx_ver_fin`. Desde el 01/03/05 en adelante ya no es así. Es decir, se ha sobrescrito el estado de una fila.

Puesto que anteriormente se ha visto que es necesario evitar las actualizaciones físicas debido a que se pierde información, surge aquí la pregunta sobre si en este caso se ha perdido información. La respuesta es “no”: a partir de los datos se infiere que `fx_ver_fin` tenía valor `{9999}` hasta el día 01/03/05. Por tanto, aunque se ha sobrescrito físicamente esa fila, no se ha perdido información de la línea, o sobre lo que se sabe de la línea en cualquier momento de su vida.

Escenario 3.3: Borrado de una línea telefónica

El 01/06/06 Juan Dato da de baja su línea telefónica. Tras dar de baja la línea de Juan Dato, la tabla de versiones de líneas telefónicas queda de la siguiente forma:

Escenario 3.3	La tabla después de eliminar la línea telefónica L617
VERSION_LINEA	
Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme	

nro_linea (CP)	**fx_ver_ini (CP)**	nro_cliente (CA)	tipo_contrato	fx_ver_fin	fx_obj_fin
L617	01/01/04	C466	PP	01/03/05	{null}
L617	01/03/05	C466	CF	**01/06/06**	**01/06/06**

Una transacción de borrado de un objeto se ha traducido en este caso en una transacción de actualización en la que se sobrescriben las fechas de fin de versión y fin de objeto de la versión actual del objeto.

Se puede identificar la primera fila como la versión inicial de la línea telefónica L617 porque no existe otra fila para L617 con una fecha anterior de `fx_ver_ini`. Se puede identificar la segunda fila como la versión terminal, es decir, el borrado lógico de la línea telefónica L617 porque es la única versión cuya `fx_obj_fin` no es nula.

El tiempo de vida de la línea telefónica L617 va desde el 01/01/04 hasta el 31/05/06, ambos inclusive.

Se observa que estas 2 filas son una forma condensada de expresar la misma semántica que se habría expresando teniendo casi 900 filas en la tabla, una para cada día de vida de esta línea. Esto ilustra como desde el punto de vista de almacenamiento de datos, las versiones son mucho más eficientes que las fotografías.

Se observa también que con esta operación de borrado de nuevo se ha actualizado físicamente una fila en la tabla de versiones de líneas telefónicas. Hasta el 01/06/06, la fila con `fx_ver_ini` 01/03/05 tenía valor {`9999`} en la columna `fx_ver_fin` y valor {`null`} en la columna `fx_obj_fin`. Desde el 01/06/06 en adelante ya no es así.

Tampoco en esta ocasión se ha perdido información de la línea o sobre lo que se sabe de la línea ya que a partir de los datos se puede inferir que hasta el día 01/06/06 la columna `fx_ver_fin` tenía el valor {`9999`} y la columna `fx_obj_fin` tenía valor {`null`}.

4.3.1.3 Restricciones semánticas

Se describen a continuación las restricciones semánticas impuestas en este patrón, presentando en primer lugar las restricciones comunes a cualquier patrón de versionado

y a continuación las restricciones relativas a la fecha de fin de objeto, específica de este patrón:

- **{RS 3-1}** Si una versión no terminal de un objeto precede a otra versión del mismo objeto, la fecha de fin de la primera debe ser el instante anterior al inicio de la segunda.

Esta es la restricción para asegurar que las sucesivas versiones de un mismo objeto no presentan huecos entre ellas. La idea es que las diferentes versiones de un objeto deben ser contiguas en el tiempo, sin huecos entre ellas, de forma que la sucesión de todas ellas representa la vida completa del objeto, igual que lo haría una única fila en una tabla no temporal presente en la tabla durante toda la vida del objeto.

Cuando la fecha de fin de versión está implícita el cumplimiento de esta restricción no requiere trabajo adicional, pero en este caso se ha optado por incluir explícitamente la fecha de fin de versión como parte del modelo por lo que se debe prestar atención especial a esta restricción para controlar que siempre la fecha de fin de versión de una versión es el instante anterior a la fecha de inicio de la versión siguiente (si se ha optado por la estrategia de representación de periodos “cerrado-abierto” esto resulta tan sencillo como asignar el mismo valor a ambas columnas, `fx_ver_fin` de la versión que se está cerrando y `fx_ver_ini` de la versión que comienza).

- **{RS 3-2}** Si una versión de un objeto no está seguida por otra versión del mismo objeto, y no se trata ella misma de una versión de borrado lógico, su fecha de fin de versión, `fx_ver_fin`, debe ser una fecha no determinada.
- **{RS 3-3}** Si una versión de un objeto corresponde a una versión de borrado, el objeto se considera borrado en la fecha de fin de objeto.

En cuanto a la fecha de fin de objeto, se especifica para ella la siguiente restricción semántica:

- **{RS 3-4}** Se debe asignar valor a `fx_obj_fin` únicamente cuando se trate del borrado del objeto. Y en este caso, se asignará el mismo valor a `fx_ver_fin`.

4.3.1.4 Transición al siguiente patrón

Por lo visto hasta ahora el patrón 3 podría ser el patrón que cumple con los objetivos planteados. Con la inclusión del tiempo de validez y el concepto de versionado es posible registrar de manera consistente en la base de datos la situación de un objeto en cualquier momento de su vida y también dar respuestas en tiempo real.

Y así sería, si no fuese porque el patrón 3 responde a un mundo ideal en el que no existen errores ni diferencias entre el instante en que se produce un hecho y éste es registrado y por tanto, no es necesario realizar correcciones ni registrar actividad retroactiva con trazabilidad, es decir, dejando constancia de las acciones que se han realizado.

El problema de intentar utilizar el patrón 3 para realizar correcciones, introduciendo una fecha pasada como fecha de inicio o fin de versión, es que se “cambia el pasado” sin dejar rastro y sin tener la posibilidad de volver a reproducir el pasado tal como estaba en aquel momento.

Por ejemplo, si se debía insertar la línea hace siete días pero se inserta hoy introduciendo su `fx_ver_ini` con la fecha de hace siete días, durante los siete días previos a esta acción la fila no estaba insertada y por tanto, durante ese periodo de tiempo, la base de datos ha estado diciendo que durante esos siete días la línea telefónica no existía. Pero en el momento en que se introduce una fecha pasada en `fx_ver_ini` la base de datos dice que la fila estaba insertada durante esos siete días, y por tanto, que esa línea telefónica estaba activa. Durante los últimos siete días la línea podía estar activa o inactiva, pero no ambas cosas. Y si lo que se necesita es volver a generar el informe tal como fue generado hace siete días, esto ya nunca será posible.

Con esta inserción retroactiva, se ha modificado el pasado y no se ha dejado ninguna indicación de lo que se ha hecho, con lo cual se pierde la trazabilidad. La consulta que cuenta líneas activas ejecutada durante la semana pasada, no incluiría la línea de Juan Dato en la cuenta. La misma consulta, ejecutada en cualquier momento después de este evento, sí incluiría la línea de Juan Dato en la cuenta para ese mismo periodo.

En sistemas Data Warehouse es donde se producen situaciones como la del analista que no puede justificar las decisiones que tomó porque al ir a hacerlo, y después de muchos quebraderos de cabeza, finalmente descubre que los datos que recibió hoy sobre el periodo de tiempo analizado son diferentes a los que le fueron entregados en la ocasión anterior y ya no es posible llegar a los mismos resultados.

El origen del problema es que la fecha en que se inserta en la tabla es también la fecha de alta efectiva para el negocio, y la fecha en que se realiza el borrado físico es también la fecha de baja efectiva para el negocio. Y se debe poder insertar y borrar en el instante adecuado ya que si se deja de hacer una inserción o un borrado, la única forma de corregir el error será mentir sobre el pasado.

Pero este problema se resuelve sin necesidad de mentir sobre el pasado, simplemente eliminando los homónimos que son el origen del problema. `fx_ver_ini` es homónimo porque significa dos cosas “la fecha en que la fila se inserta” y también “la fecha en que el objeto representado por la fila es efectivo”. Y `fx_ver_fin` es homónimo por razones análogas.

Distinguir las fechas de actividad de la base de datos y las fechas de validez de la información para el negocio crea lo que se denomina un patrón bitemporal, que se trata en detalle a continuación, con el patrón 4.

4.3.2 PATRÓN 4: Versionado bitemporal

El patrón 3 es un gran avance ya que es el primero que preserva el conocimiento del estado de un objeto en cualquier momento de su vida. Sin embargo, este patrón no diferencia entre fechas del negocio y fechas del sistema, por lo que no permite registrar eventos con retraso ni corregir datos erróneos.

Con el patrón 4 se implementa un modelo bitemporal, es decir, se introduce un segundo conjunto de fechas. El primer conjunto de fechas son las fechas de validez para el negocio, que para los patrones 3 y 4 son:

- Fecha de inicio de versión (`fx_ver_ini`): el instante en el que esa versión del objeto comienza a ser la versión actualmente en vigor.
- Fecha de fin de versión (`fx_ver_fin`): el instante en el que esa versión del objeto deja de estar en vigor.

Juntas, la fecha de inicio de versión y la fecha de fin de versión, determinan el *tiempo de validez* de los datos.

El segundo conjunto, incluido en el patrón 4 pero no en el patrón 3, está formado también por una pareja de fechas:

- Fecha de creación (`fx_alta`): el instante en el que esa fila fue físicamente insertada en la tabla.
- Fecha de borrado lógico (`fx_baja`): el instante en el que esa fila deja de considerarse válida.

Juntas, la fecha de alta y la fecha de baja, determinan el *tiempo de transacción* de los datos.

Estos dos conjuntos de fechas son ortogonales entre sí, es decir, se puede utilizar uno o ambos (o ninguno de los dos) y los cambios en uno de los conjuntos no implican necesariamente cambios en el otro conjunto.

Con un modelo bitemporal es posible corregir errores dejando constancia de que así ha sido y permitiendo por tanto reconstruir la situación de la tabla en cualquier instante del tiempo. Es innegable que los errores existen y en un Data Warehouse hay que sumar a los posibles errores cometidos en el propio sistema, los errores heredados en los datos procedentes de otros N sistemas que le sirven como fuente de datos.

Por este motivo es necesario establecer mecanismos de corrección que permitan conservar el aspecto de los datos antes y después de realizar la corrección de los mismos.

Cuando se trata de bases de datos transaccionales, la forma de corregir un error es:

a) crear una transacción que anule la transacción errónea y

b) añadir la transacción correcta.

A pesar de que la tabla de líneas telefónicas no es transaccional, es versionada, se debería realizar algo similar a esto. No se debería eliminar o sobrescribir el error. Se debería poder identificar de alguna manera, de forma que sea invisible para las consultas normales (pero recuperable con consultas específicas) y que la versión correcta aparezca en su lugar.

Este es el enfoque de "preservar la evidencia" en la corrección de errores, que no puede ser abordado con un modelo unitemporal.

4.3.2.1 Estructura de datos

La estructura de datos definida para el patrón 4 es la siguiente:

nro_linea (CP)	
fx_ver_ini (CP)	*(Fecha de inicio del periodo de validez de la versión)*
fx_alta (CP)	*(Fecha de la inserción de la fila)*
nro_cliente (CA)	
tipo_contrato	
…	
fx_ver_fin	*(Fecha de fin del periodo de validez de la versión)*
fx_baja	*(Fecha de fin de validez de la fila)*
fx_obj_fin	*(Fecha de baja del objeto)*

Cada fila en esta tabla representa, para una línea telefónica concreta, el estado de esa línea telefónica que comenzó a ser efectivo en la fecha `fx_ver_ini` y que fue añadido a la tabla en la fecha `fx_alta`. Cuando ese estado deja de ser válido, porque ha habido un cambio de estado, se actualiza la fila colocando la fecha en que deja de ser válido en `fx_ver_fin` y se inserta, salvo que se trate de la baja del objeto, la nueva versión con su `fx_ver_ini`, reflejando el nuevo estado.

En el caso de correcciones, si se borrase la fila físicamente se perdería la información, pero en el modelo bitemporal es posible añadir la fila correcta con el mismo `nro_linea` y `fx_ver_ini`, pero con una nueva `fx_alta`, la fecha en que la fila se inserta físicamente. Y la fila errónea se actualiza colocando esta misma fecha en `fx_baja`.

Adicionalmente, para diferenciar las versiones que suponen un borrado lógico del objeto, se incluye la fecha de fin de objeto (`fx_obj_fin`).

4.3.2.2 Escenarios

Para este patrón no se muestran los escenarios de alta, actualización y borrado de la línea telefónica vistos hasta ahora, en los cuales la fecha de entrada en vigor de los datos para el negocio coincide con la fecha en la que los datos son registrados en la base de datos, ya que tienen un tratamiento equivalente al descrito en el patrón 3. La diferencia en el patrón 4 es la aparición de las fechas `fx_alta` y `fx_baja` que para dichos escenarios siempre van a tener los valores de la fecha actual y la fecha no determinada, `{9999}`, respectivamente.

Se pasa por tanto a mostrar escenarios que no eran posibles con los patrones vistos hasta ahora, y que ya sí lo son con el patrón 4 por tratarse de un patrón bitemporal. De hecho, son estos escenarios los que tienen especial relevancia en sistemas Data Warehouse, por tratarse del tipo de operaciones más habituales en sistemas de este tipo. Se trata de la manipulación retroactiva de los datos, tanto para el caso de introducción retroactiva de datos como para la corrección de datos con retención de errores

En esta ocasión, para facilitar el seguimiento de los ejemplos en los escenarios, se van a mostrar juntas en los escenarios las fechas de inicio y fin de versión, así como las fechas de alta y baja.

Escenario 4.1: Inserción retroactiva de datos

Se supone que en lugar de insertar la línea de Juan Dato el mismo día de su entrada en vigor, el 01/01/04, la información del alta de la línea se recibe el día 01/10/07. El aspecto de la tabla de versiones de líneas telefónicas sería el siguiente:

Escenario 4.1							
VERSION_LINEA							
Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme							
nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	**01/01/04**	**{9999}**	**01/10/07**	**{9999}**	**C466**	**PP**	**{null}**

La tabla después de insertar la línea telefónica L617 de forma retroactiva

Durante el periodo del 01/01/04 al 01/10/07 la base de datos decía que la línea de Juan Dato no existía. A partir de esa fecha, dice lo contrario. Sin embargo, en este caso ya sí es posible reproducir en cualquier momento lo que la base de datos decía durante ese periodo (utilizando la información de `fx_alta`).

De esta forma el modelo bitemporal permite cambiar el pasado dejando una indicación de ello, que es la fecha en que dicho pasado fue modificado (`fx_alta`).

Supóngase que, dos meses después, el 01/12/2007 llegó la información de actualización y otros dos meses después, la de la baja de la línea, ambas se podrían reflejar tal como muestran los dos escenarios siguientes.

Escenario 4.2: Actualización retroactiva de datos

Se supone que en lugar de actualizar la línea de Juan Dato el 01/03/05, la información de la modificación se recibe el día 01/12/07. Tras registrar esta actualización el aspecto de la tabla de versiones de líneas telefónicas es el siguiente:

Escenario 4.2

VERSION LINEA

Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme

La tabla después de actualizar la línea telefónica L617 de forma retroactiva

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/01/04	{9999}	01/10/07	**01/12/07**	C466	PP	{null}
L617	**01/01/04**	**01/03/05**	**01/12/07**	**{9999}**	**C466**	**PP**	**{null}**
L617	**01/03/05**	**9999**	**01/12/07**	**{9999}**	**C466**	**CF**	**{null}**

Como se puede observar, la actualización de la línea telefónica se ha traducido a tres transacciones físicas:

[1] Se finaliza el periodo de validez de la fila que representa la versión actual, introduciendo la fecha de baja del mismo (`fx_baja`).

[2] Se inserta la versión que sustituye a la anterior, ahora ya con la fecha de fin de versión conocida.

[3] Se inserta la versión actual de la línea telefónica con su fecha de inicio de versión y su nuevo tipo de contrato CF, válida hasta una “fecha no determinada”.

Como resultado de esta operación la historia de la línea telefónica ha quedado reflejada mediante 3 filas. Además ha quedado registrado que se produjo una corrección el día 01/12/07. A fecha actual, la historia de la línea está reflejada por las 2 filas con `fx_baja` no determinada ({9999}). Del 01/01/04 al 01/03/05 su tipo de contrato fue PP y desde entonces hasta la actualidad es CF. Pero además es posible saber que desde el 01/10/07 hasta el 01/12/07 se creía, erróneamente, que el tipo de contrato era PP (cuando sólo lo fue hasta el 01/03/05).

Escenario 4.3: Baja retroactiva de datos

Y por último se supone que en lugar de dar de baja la línea de Juan Dato el 01/06/06, la información de la baja se recibe el día 01/02/08. El aspecto de la tabla de versiones de líneas telefónicas sería el siguiente:

Escenario 4.3

VERSION LINEA

Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme

La tabla después de eliminar la línea telefónica L617 de forma retroactiva

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/01/04	{9999}	01/10/07	01/11/07	C466	PP	{null}
L617	01/01/04	01/03/05	01/11/07	{9999}	C466	PP	{null}
L617	01/03/05	9999	01/11/07	**01/02/08**	C466	CF	{null}
L617	**01/03/05**	**01/06/06**	**01/02/08**	**{9999}**	**C466**	**CF**	**01/06/06**

También en este caso, una operación lógica (dar de baja una línea telefónica) se ha traducido en varias operaciones físicas:

[1] Se "cierra" el periodo de validez de la fila que representa la versión actual, introduciendo la fecha de baja del mismo (`fx_baja`).

[2] Se inserta una copia de la versión que hasta ahora era la actual, pero ahora ya con su fecha de fin de versión y fecha de fin de objeto actualizadas.

A pesar de haber utilizado para el ejemplo unas fechas algo exageradas (resulta bastante poco creíble que alguien tarde 3 años en darse cuenta de que faltó registrar la historia de una línea telefónica), nada impide que esto ocurra realmente. Y de hecho, ocurre, quizá con intervalos de tiempo menores, pero esa sería la única diferencia. Como se puede ver, el patrón 4 permite este tipo de modificaciones retroactivas porque siempre se tiene constancia del tiempo de transacción, es decir, del instante en que se registró esa información, permitiendo así volver a reconstruir el estado de la tabla en cualquier instante del tiempo y asegurando, por tanto, la trazabilidad.

Por último, se muestran a continuación dos escenarios de corrección con retención de errores.

Escenario 4.4a: Corrección con retención de errores (tipo 1)

Se supone que un mes después de dar de alta una línea telefónica, alguien se da cuenta de que los datos son incorrectos porque se registró en la base de datos con un tipo de contrato PP cuando realmente su tipo es CF. No basta con actualizar la línea para que a partir de este momento aparezca como CF, sino que se trata de corregir el error en el sistema y registrar que la línea desde el instante inicial de su entrada en vigor, el

01/01/04, tenía el tipo de contrato correcto, CF. Tras registrar esta información, la tabla de versiones de líneas telefónicas tiene el siguiente aspecto:

Escenario 4.4a

VERSION LINEA

Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/01/04	{9999}	01/02/04	**01/03/04**	C466	PP	{null}
L617	**01/01/04**	**{9999}**	**01/03/04**	**{9999}**	**C466**	**CF**	**{null}**

La tabla después de corregir la línea telefónica L617

Como se observa, la operación de corregir los datos de una línea telefónica se realiza en dos pasos:

[1] Se "cierra" el periodo de validez de la fila que representa la versión actual, introduciendo la fecha de baja del mismo (`fx_baja`).

[2] Se inserta una copia de la versión que hasta ahora era la actual, pero ahora ya con su tipo de contrato correcto CF.

La gran diferencia con el modelo unitemporal es que para un mismo objeto pueden existir varias filas con la misma fecha de inicio de versión. Y con las fechas `fx_alta` y `fx_baja` se puede saber cuál de ellas era la que existía en la tabla de versiones de líneas telefónicas en cada instante. De esta forma ya sí es posible reproducir cuál era la situación de los objetos en cualquier momento.

Tal como se ha visto en este escenario, una corrección que no altera el periodo de validez de una versión tiene exactamente el mismo periodo de validez que la versión que dicha actualización corrige.

Se presenta a continuación un escenario cuyo objetivo es corregir precisamente un periodo de validez.

Escenario 4.4b: Corrección con retención de errores (tipo 2)

Considérese una versión no inicial introducida con una fecha de inicio anterior a la que debería ser. En ese caso, es necesario mover su fecha de inicio hacia delante. Pero al hacerlo se introduciría un hueco entre ella y su versión predecesora. Por el mismo razonamiento, si se mueve una fecha de fin hacia atrás se estaría introduciendo un hueco entre ella y su sucesora. Para evitar que dichas correcciones violen la restricción semántica {RS 3-1}, que dice que "si una versión no terminal de un objeto precede a otra versión del mismo objeto, la fecha de fin de la primera debe ser el instante anterior al inicio de la segunda", se deben corregir siempre dos versiones en ambos casos,

cuando se mueve hacia delante una fecha de inicio en una versión no inicial o cuando se mueve hacia atrás la fecha de fin de una versión no terminal. Una es la versión que se está corrigiendo y otra es la versión adyacente, que debe ser corregida para “rellenar el hueco” creado por la primera corrección.

No se debería pensar en estas dos actualizaciones como dos transacciones. Existe una transacción única, en la que se mueve el punto de transición entre dos versiones adyacentes. La actualización semántica requiere que sean actualizadas físicamente dos filas.

Se toma como situación de partida la obtenida tras realizar los cambios correspondientes al escenario 3.2, en el que se actualizaba la línea telefónica para cambiar su tipo de contrato de PP a CF, representada a continuación tal como sería si se hubiese utilizado el patrón 4:

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/01/04	**01/03/05**	01/01/04	{9999}	C466	PP	{null}
L617	**01/03/05**	**{9999}**	**01/03/05**	**{9999}**	**C466**	**CF**	**{null}**

En el mes de julio se detecta que el tipo de contrato CF para esta línea no se debía haber aplicado a partir del mes de marzo, sino dos meses más tarde, es decir, a partir del mes de mayo.

Tal como se ha visto, para corregir este error es necesario corregir también el periodo de validez de la versión inmediatamente anterior para extender su periodo de validez hasta la nueva fecha. Tras realizar las modificaciones, el aspecto de la tabla de versiones de líneas telefónicas es el siguiente:

Escenario 4.4b

VERSION LINEA

Cada fila de esta tabla representa una versión de una línea telefónica de Tele Acme

La tabla después de corregir la línea telefónica L617

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/01/04	01/03/05	01/01/04	**01/07/05**	C466	PP	{null}
L617	01/03/05	{9999}	01/03/05	**01/07/05**	C466	CF	{null}
L617	**01/01/04**	**01/05/05**	**01/07/05**	**{9999}**	**C466**	**PP**	**{null}**
L617	**01/05/05**	**{9999}**	**01/07/05**	**{9999}**	**C466**	**CF**	**{null}**

En primer lugar, se asigna valor a la `fx_baja` de las filas que representan las dos versiones cuyos periodos de validez se ven afectados por esta operación. Y a continuación se insertan las filas equivalentes a las anteriores, pero ya con sus periodos de validez correctos.

Se destaca en este momento la importancia de operar correctamente con los periodos. Dado que no se trata de una funcionalidad que proporcione el SGBD se debe controlar mediante código las diferentes situaciones para no generar huecos ni solapamientos entre versiones, ambas situaciones serían un error y supondrían un desastre en los datos.

Mientras los SGBDs comerciales sigan sin proporcionar un tipo de dato `PERIODO` con su correspondiente constructor y operaciones, depende de la habilidad de los profesionales de las TI la correcta gestión de la relaciones entre periodos en los sistemas que incorporan gestión del tiempo. Ver más detalles en el apartado “4.4.2.2 Estrategia de representación de periodos”.

4.3.2.3 Restricciones semánticas

El patrón 4 opera bajo las mismas restricciones semánticas que el patrón 3 para la pareja de fechas que indican el periodo de validez. En cuanto a la pareja de fechas que indican el periodo de transacción se definen las siguientes restricciones semánticas:

- **{RS 4-5}** Siempre que se inserta una fila se asigna la fecha actual a la columna `fx_alta`.
- **{RS 4-6}** Cuando una fila deja de ser válida se asigna la fecha actual a la columna `fx_baja`.

Mediante la incorporación de esta segunda pareja de fechas, el periodo de transacción, este patrón mantiene el nivel de trazabilidad necesario para poder seguir la trayectoria de los datos, al quedar registrado cada uno de los instantes en que éstos son manipulados desde su aparición en el sistema. Así, en el caso de realizarse una corrección de datos, este hecho queda registrado y son mantenidas ambas imágenes de los datos, anterior y posterior a la corrección.

Como consecuencia, con este patrón es posible reconstruir el aspecto que tenían los datos en cualquier momento del pasado, cosa que con el patrón 3, basado en un modelo unitemporal no era posible.

Una vez presentado el patrón que atiende todos los requisitos definidos, se pasa a ver el modelo de gestión del tiempo en su conjunto, ya que la estructura de datos y las restricciones semánticas constituyen tan sólo una parte del modelo final.

4.4 MODELO DE GESTIÓN DEL TIEMPO

Un modelo de datos debe incluir la definición de los constructores y formalismos que proporciona para definir, modificar y acceder a los datos. El modelo de datos provee a

sus usuarios de **estructuras de datos** que ocultan muchos de los detalles sobre cómo están almacenados físicamente los datos. Utiliza conceptos lógicos como objetos, sus propiedades y relaciones, que para la mayoría de los usuarios son más fáciles de entender que los conceptos de almacenamiento físico directamente. También proporciona **operaciones** para acceder y modificar los datos, así como **restricciones de integridad** para expresar lo que se considera un estado consistente de la base de datos.

Desgraciadamente, la madurez de los SGBDs bitemporales no ha alcanzado todavía el nivel suficiente como para postularse como alternativa a los SBGDs relacionales tradicionales a la hora de abordar un proyecto Data Warehouse en entornos empresariales. Por ahora, y previsiblemente también en el corto plazo, los SGBDs relacionales seguirán constituyendo la pieza fundamental de los sistemas analíticos. Por tanto, la capacidad de éstos de incluir una gestión del tiempo adecuada y eficaz seguirá estando en las manos de los responsables de TI.

En este sentido, el objetivo de este apartado es proporcionar una guía con los elementos fundamentales a considerar en la construcción de un sistema Data Warehouse utilizando una gestión del tiempo basada en un modelo bitemporal.

4.4.1 Consideraciones previas

A la hora de diseñar un modelo de gestión del tiempo se deben tener en cuenta una serie de cuestiones que son de vital importancia ya que de las decisiones adoptadas dependerá en gran medida la complejidad del modelo que se vaya a implementar.

Los aspectos que se han identificado como los de mayor impacto en la complejidad futura del modelo son:

a) Actividad proactiva.

b) Recurrencia de objetos.

c) Versionado integrado o separado.

Se ve a continuación cada una de ellas con más detalle.

4.4.1.1 Actividad proactiva

Considerando el momento en que se genera respecto al momento actual, se pueden diferenciar tres tipos de actividad: actividad actual (cuando la fecha de validez para el negocio coincide con la fecha del sistema), actividad retroactiva (cuando la fecha de validez es anterior a la fecha del sistema) y actividad proactiva (cuando la fecha de validez es posterior a la fecha del sistema).

En los sistemas Data Warehouse lo habitual es trabajar con actividad actual y retroactiva. Y como se ha visto, para dar soporte a la actividad retroactiva con trazabilidad es necesario trabajar con un modelo bitemporal.

No obstante, es posible también considerar actividad proactiva, para lo cual es necesario realizar las adaptaciones necesarias al modelo para tener en cuenta las situaciones derivadas de la incorporación de este tipo de actividad.

En el caso de considerar actividad actual y retroactiva existen únicamente dos tipos de filas en la tabla:

a) filas que comienzan en el pasado y finalizan en el pasado.

b) filas que comienzan en el pasado y finalizan en una fecha no determinada.

Pero en el caso de la actividad proactiva se permiten dos tipos de filas adicionales:

c) filas que comienzan en el pasado y finalizan en el futuro, pero antes de una "fecha no determinada".

d) filas que comienzan en el futuro y finalizan en el futuro, en una fecha determinada o en una "fecha no determinada".

Por ejemplo**,** al hacer una actualización de un objeto se asume que el significado de la misma es "desde este momento hasta una fecha no determinada". Sin embargo, si se permite actividad proactiva, puede existir en la base de datos una versión futura del objeto y será necesario decidir si se mantiene esta definición de actualización, en cuyo caso habría que "anular" esa versión futura que fue insertada con anterioridad, o si se modifica esta definición para pasar a ser "desde este momento y hasta que se encuentre otra versión". Como se ve, el asunto puede complicarse bastante.

Se muestra a continuación un escenario que representa un ejemplo de introducción de datos de manera proactiva en la base de datos.

Escenario 4.5: Inserción proactiva de una línea telefónica

En este escenario se asume que la línea de Juan Dato, introducida en la base de datos el 01/01/04 no entra en vigor hasta el 01/06/04:

Escenario 4.5

VERSION LINEA

Cada fila de esta tabla representa una versión de cada una de las líneas telefónicas de Tele Acme

La tabla después de insertar la línea telefónica L617 mediante inserción proactiva

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	**01/06/04**	**{9999}**	**01/01/04**	**{9999}**	**C466**	**PP**	**{null}**

Con esta capacidad es posible "insertar y olvidar" en el sentido de que se puede insertar un objeto nuevo tan pronto como se sabe de su existencia sin necesidad de hacer nada

posteriormente para activarlo. El simple paso del tiempo asegurará que el objeto pase a estar activo en la fecha de entrada en vigor indicada.

Mientras que sólo existe un tipo de inserción proactiva de una línea telefónica, existen dos tipos de actualizaciones proactivas. Con el primer tipo, Juan Dato solicita un cambio en su línea para que sea efectivo después de que la línea entre en vigor. Con el segundo tipo, Juan Dato solicita un cambio en su línea para que sea efectivo previamente a la fecha de entrada en vigor de la línea.

Escenario 4.6a: Actualización proactiva (tipo 1) de una línea telefónica

Para ilustrar la actualización proactiva del tipo 1, se asume que a finales de marzo Juan Dato solicita un cambio del tipo de línea de PP a CF, para que sea efectivo el 01/09/04 y este cambio es registrado físicamente en la tabla de versión de línea el día 01/04/04. Se trata de una actualización proactiva ya que la fecha de creación es anterior a la fecha de entrada en vigor.

Tras insertar la nueva versión (una inserción física que semánticamente actualiza la línea) la tabla de versiones de líneas tiene el siguiente aspecto:

Escenario 4.6a

VERSION LINEA

Cada fila de esta tabla representa una versión de cada una de las líneas telefónicas de Tele Acme

La tabla después de actualizar la línea telefónica L617 utilizando el primer tipo de actualización proactiva

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/06/04	{9999}	01/01/04	**01/04/04**	C466	PP	{null}
L617	**01/06/04**	**01/09/04**	**01/04/04**	**{9999}**	**C466**	**PP**	**{null}**
L617	**01/09/04**	**{9999}**	**01/04/04**	**{9999}**	**C466**	**CF**	**{null}**

Semánticamente las dos últimas filas indican que la línea telefónica L617 entrará en vigor el día 01/06/04 como una línea del tipo PP. Tres meses después de esta última fecha, la línea continuará en vigor pasando a ser una línea del tipo CF.

Escenario 4.6b: Actualización proactiva (tipo 2) de una línea telefónica

Pero existe un segundo tipo de actualización proactiva a considerar. Se trata de la situación en la cual se registra una modificación cuya fecha de entrada en vigor es anterior a la fecha de entrada en vigor de la propia línea (que fue registrada mediante un alta proactiva en el último escenario de alta de línea).

Para este escenario se asume que el 01/04/04 se solicita un cambio de tipo de contrato de PP a CF para que sea efectivo el 01/05/04.

Claramente se trata también de una actualización proactiva, puesto que se introduce dos meses antes de su fecha de entrada en vigor. Por tanto, replicando el funcionamiento visto en el escenario anterior, la tabla de versiones de líneas telefónicas tendría el siguiente aspecto:

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/06/04	**01/05/04**	01/01/04	{9999}	C466	PP	{null}
L617	**01/05/04**	**{9999}**	**01/04/04**	**{9999}**	**C466**	**CF**	**{null}**

Lo que indican semánticamente estas dos últimas filas es incorrecto. En otras palabras, el resultado no es el deseado. Estas dos filas indican que la línea telefónica L617 entrará en vigor el día 01/05/04 como una línea de tipo CF y que un mes después seguirá en vigor pero se transformará en una línea de tipo PP.

La diferencia entre estos dos tipos de actualizaciones proactivas radica en que el tipo 1 es una actualización de una línea telefónica que ya se encuentra en vigor, es decir que tiene una versión efectiva en el momento en que es insertada físicamente la nueva versión. Sin embargo, el tipo 2, se trata de una actualización de una línea telefónica que no está en vigor.

Se utiliza exactamente el mismo código para realizar ambos cambios. En el primero, una actualización en la base de datos implementa correctamente su solicitud de cambio. Pero en el segundo caso, el mismo código produce un desastre en su ejecución. Durante dos meses y medio tras la actualización del tipo 2, todo funciona como debería. Después, el 01/06/04, la línea de Juan Dato de repente pasa a ser del tipo CF.

En este caso lo que se debe implementar es una “corrección” y no una actualización de la línea telefónica, de la siguiente forma:

Escenario 4.6b

VERSION_LINEA

Cada fila de esta tabla representa una versión de cada una de las líneas telefónicas de Tele Acme

La tabla después de actualizar la línea telefónica L617 utilizando el segundo tipo de actualización proactiva

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/06/04	{9999}	01/01/04	**01/03/04**	C466	PP	{null}
L617	**01/05/04**	**{9999}**	**01/03/04**	**{9999}**	**C466**	**CF**	**{null}**

La última fila equivale a una inserción de la primera versión de la línea de Juan Dato con las últimas características solicitadas. Previamente se ha registrado la baja lógica, mediante la introducción de la fecha de baja, `fx_baja`, de la transacción de alta proactiva anterior por tener una fecha de inicio de versión posterior a la nueva que la sustituye.

Escenario 4.7: Borrado proactivo de una línea telefónica

Un borrado proactivo de tipo 1 finaliza una línea telefónica existente. Un borrado proactivo de tipo 2 finaliza una línea telefónica que todavía no existe. Se podría decir lo siguiente: Un borrado proactivo de tipo 1 finaliza una línea telefónica existente proporcionando una fecha de finalización de la versión de línea que estaba en vigor en el momento de realizar la acción física de tipo 1. Y un borrado proactivo de tipo 2 finaliza la versión de una línea que tiene la fecha de entrada en vigor más tardía, pero lo hace antes de dicha fecha. Es decir, a menos que otra acción tenga lugar, un borrado proactivo de tipo 2 asegura que la correspondiente línea nunca entrará en vigor.

Se supone para el ejemplo que el día 01/03/04 se solicita la baja de la línea que iba a entrar en vigor el día 01/06/04. Tras realizar los cambios, este sería el aspecto de la tabla de versiones de líneas telefónicas:

Escenario 4.7

`VERSION LINEA`

Cada fila de esta tabla representa una versión de cada una de las líneas telefónicas de Tele Acme

La tabla después de borrar la línea telefónica L617 mediante borrado proactivo

nro_linea (CP)	**fx_ver_ini (CP)**	fx_ver_fin	**fx_alta (CP)**	fx_baja	nro_cliente (CA)	tipo_contrato	fx_obj_fin
L617	01/06/04	{9999}	01/01/04	**01/03/04**	C466	PP	{null}
L617	**01/06/04**	**01/03/04**	**01/03/04**	**{9999}**	**C466**	**PP**	**01/03/04**

La tabla muestra el objeto que nunca llegó a estar en vigor, puesto que fue dado de baja antes de que eso ocurriera. Sin embargo, incluso la historia de un objeto que no llegó a "existir" es almacenada con la historia versionada.

4.4.1.2 Recurrencia de objetos

A lo largo de este trabajo se han utilizado los conceptos de *objeto* y *versión*. Y se ha presentado un objeto como una sucesión continua de versiones.

Sin embargo, la realidad puede ser algo más complicada: Un objeto puede desaparecer y volver a aparecer tiempo después. Un ejemplo sería el de un cliente que se da de baja en Tele Acme y un tiempo después decide volver y darse de nuevo de alta. Una opción para Tele Acme sería dar de alta un nuevo cliente, que podría ser idéntico al antiguo

excepto en su identificador de cliente, por lo que se trataría, de hecho, de un cliente nuevo sin relación con el anterior. Y otra opción es dar de alta la "reaparición" de un antiguo cliente, lo cual es más realista y útil desde el punto de vista del análisis de la información.

En una tabla versionada, una reaparición significa que entre las sucesivas versiones de un mismo objeto pueden aparecer huecos. Sin embargo, a lo largo de este trabajo en todos los modelos se ha incluido una restricción semántica encargada precisamente de garantizar que esto no ocurriese.

Para resolver este dilema, se introduce un nuevo concepto, el *episodio*, que se define como cada uno de los diferentes periodos de tiempo en los que un objeto está presente en la base de datos. Un episodio es una sucesión de versiones contiguas. Y un objeto puede tener uno o más episodios.

Es decir, en el caso de que uno de los requisitos de negocio del sistema sea que se debe permitir la recurrencia de objetos (es decir, que un objeto tenga más de un episodio) entonces se debe aplicar a nivel de *episodio* todo lo descrito en este trabajo para *objetos*.

Si un objeto pudiera reaparecer, ¿qué ocurriría? En el caso de la línea telefónica de Juan Dato aparecería un borrado lógico de la línea telefónica P138 seguido después de un tiempo por otra versión de la línea telefónica P138, que no es un borrado lógico. Normalmente, los objetos persistentes reaparecen pasado un periodo de tiempo. Por tanto, en este sentido, puede haber huecos entre varios episodios del mismo objeto.

Puesto que la siguiente versión es una versión inicial de la línea tras un borrado lógico anterior de la misma línea, se trata de una reaparición de la línea. Se denomina la primera versión de la línea telefónica P138 (la versión del 01/01/05) como la *versión inicial original*, y cada una de las versiones que aparecen inmediatamente después de un borrado lógico de P138 (la versión del 01/01/05) como *versión inicial sucesora*. El número de versiones iniciales de un objeto es el número de episodios de ese objeto. Pueden existir o no huecos entre los sucesivos episodios. En este caso, el hueco se extiende desde el instante posterior a la finalización del primer episodio hasta el instante en el que comienza el segundo episodio.

Si el número de borrados lógicos coincide con el número de episodios, entonces el objeto no existe actualmente en la base de datos. Esto es porque cada episodio, incluido el más reciente, ha finalizado con su correspondiente borrado lógico.

El otro único caso que se puede dar es aquel en que el número de episodios es una unidad mayor que el número de borrados lógicos. Esto es porque en una serie de versiones del mismo objeto, lo que lo divide en episodios son los borrados lógicos. En este caso, el objeto existe actualmente en la base de datos porque no existe el borrado lógico para el episodio más reciente (o único).

4.4.1.3 Estrategia de versionado: integrado o separado

Una forma de implementar el versionado es manteniendo todas las versiones no actuales en una tabla separada. En ese caso se denomina “versionado separado”. La tabla original mantiene su estatus de tabla no-temporal y sus filas siempre reflejan el estado actual de los objetos que representan. Todas las versiones no actuales se mantienen en una tabla separada, normalmente en la misma base de datos. Esta tabla separada se denomina habitualmente “tabla histórica” o “tabla de versión”.

Un motivo para mantener las versiones separadas de los datos actuales es evitar el impacto negativo en el rendimiento de las consultas que necesitan acceder únicamente a los datos actuales. Otro motivo es evitar que estas consultas sean más complejas. Si se almacenan versiones en la misma tabla que los datos actuales, entonces cada consulta que necesite acceder sólo a los datos actuales debería especificar que quiere obtener la versión cuya fecha de versión (en la clave principal) es la última fecha dentro de las versiones cuya clave principal es idéntica salvo por la fecha.

La otra forma de implementar el versionado es manteniendo todas las versiones, incluida la versión actual, en la misma tabla. En este caso se denomina “versionado integrado” y transforma la tabla original no-temporal en una tabla versionada. Dependiendo de si esto se hace añadiendo una o dos fechas a la clave principal el resultado es la transformación de la tabla no-temporal en una tabla unitemporal o bitemporal, es decir, con una o dos dimensiones temporales.

Un motivo para utilizar versionado integrado en lugar de separado es la simplicidad. Con el versionado integrado sólo existe una tabla sobre la que consultar, por tanto, para consultar estados de los objetos en diferentes instantes de tiempo sólo es necesario modificar la fecha utilizada en la cláusula `WHERE` de la consulta. Para recuperar simultáneamente datos de versiones actuales y no actuales de los objetos no es necesario realizar cruces entre dos tablas, cruces que sí son necesarios cuando se utiliza versionado separado. Una única tabla también supone una única unidad de trabajo para los departamentos de operaciones, un único objeto del que hacer copia de respaldo y archivar y un único objeto a restaurar si es necesario.

Las tres características descritas deben ser analizadas por su influencia posterior en el diseño de las operaciones que formarán parte del modelo de gestión del tiempo que se vaya a implementar.

4.4.2 Estructuras de datos

La estructura de datos viene dada por el patrón seleccionado.

Tratándose de sistemas Data Warehouse lo normal será utilizar un patrón basado en un modelo bitemporal ya que, como se ha visto, es el único que soporta los requisitos especificados.

Pero la estructura de datos definida por el patrón a utilizar no es la única estructura de datos que interviene en un modelo de gestión del tiempo. Se debe pensar que con un modelo de estas características se va trabajar con rangos de fechas y que el SGBD no proporciona un tipo de dato que soporte los rangos de fechas o periodos de tiempo. Quizá porque resulta tan obvio simular un periodo de tiempo mediante una pareja de fechas, una que identifique la fecha de inicio del periodo y otra para la fecha de fin de periodo, éste es un aspecto a cuyo análisis no se suele dedicar demasiado tiempo. Sin embargo, es necesario definir no sólo cómo se van a representar los periodos, sino también qué valores especiales es necesario definir y cómo se va a operar con los periodos de tiempo.

4.4.2.1 Un tipo de dato Periodo

Una de las primeras cosas que se echa de menos en los SGBDs actuales al empezar a trabajar con versionado de objetos es el soporte para un tipo de dato `PERIODO`.

La alternativa que queda por tanto, es hacer uso de las facilidades proporcionadas por el SGBD para construirse un tipo de dato de usuario. Para ello en primer lugar debe definirse la estrategia que se va a utilizar para representar periodos en el sistema, debe también definirse qué valores se van a reservar para representar “fechas especiales”, y por último deben definirse las operaciones que va a ser necesario realizar con ellos.

4.4.2.2 Estrategia de representación de periodos

Una forma de representar rangos de fechas es mediante una pareja de fechas. Otra es especificando una fecha de inicio y una duración. Estas dos formas de representación de rangos de fechas son semánticamente equivalentes, es decir, cada una puede manejar los mismos requisitos de negocio que la otra, difiriendo únicamente en detalles de implementación. En este trabajo se ha optado por representar rangos de fechas mediante una pareja de fechas.

Y cuando se utiliza una pareja de fechas, se debe especificar cuáles de ellas, si alguna, están incluidas en el rango que especifican. De nuevo, las opciones son semánticamente equivalentes. En este trabajo la norma utilizada ha sido incluir las fechas de inicio y excluir las fechas de fin, es decir los periodos se representan mediante un intervalo de fechas “cerrado-abierto”.

4.4.2.3 Estrategia de representación de la “fecha no determinada”

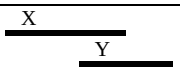
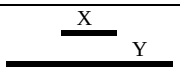
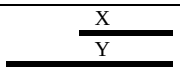
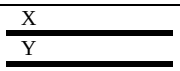
Cuando se introduce una nueva versión de un objeto, lo normal es que en ese momento no se conozca su fecha de fin, ya que lo que se está introduciendo en la base de datos es el nuevo estado del objeto que se mantendrá así “hasta nuevo aviso”. En este caso se debe decidir qué fecha utilizar para representar este valor de “fecha no determinada” que se trata de una fecha en el futuro que será concretada más adelante, cuando se produzca un nuevo cambio de estado de el objeto. Una alternativa es utilizar el valor

`NULL` y otra es utilizar la máxima fecha permitida por el SGBD. Lo que se recomienda en este caso es utilizar la máxima fecha permitida por el SGBD. Las ventajas de utilizar la máxima fecha permitida por el SGBD son las siguientes:

- Las consultas son más sencillas. Se recuerda aquí que cualquier comparación con `NULL` devuelve siempre falso, por lo que si se utiliza `NULL` es necesario no olvidar incorporar el "… `OR fx_fin IS NULL` …" en las consultas.
- Se deja el uso de `NULL` para lo que realmente significa, que es "fecha desconocida".
- Consideraciones internas de rendimiento del SGBD: Cuando se actualiza una fila con un valor que ocupa físicamente más que el existente previamente en la base de datos (como cuando se va a introducir una fecha en el lugar que hasta ahora ocupaba un valor `NULL`), puede ser necesario un trabajo adicional del SGBD para reubicar la fila en otro lugar. Esta operación afecta al tiempo que lleva realizar la actualización y puede derivar en un problema de fragmentación de la tabla, ambos aspectos fundamentales a tener en cuenta en sistemas de este tipo que manejan grandes volúmenes de datos.

4.4.2.4 Operaciones con periodos

Cuando se trabaja con historia versionada es continuamente necesario trabajar con relaciones y operaciones entre periodos.

X antes que Y (1)	
X precede a Y (1)	
X se solapa con Y (1)	
X inicia Y (1)	
X durante Y (1)	
X termina Y (1)	
X igual a Y	

(1) también la relación inversa

Figura 4.5: Relaciones de Allen

Las diferentes relaciones que se pueden dar entre dos periodos han sido estudiadas por James F. Allen [Allen, 83], que define 13 relaciones base entre periodos, que se conocen como *Relaciones de Allen*. En la Figura 4.5 se muestran las diferentes relaciones que, junto con las inversas de las seis primeras, constituyen las 13 relaciones definidas por Allen.

Un tipo de dato, además de una representación, tiene que tener definidas unas operaciones. Y se deberá tener prevista la implementación de este conjunto de operaciones de forma que la misma implementación sea utilizada por todo el equipo de desarrollo, sin dejar a la libre interpretación de cada persona la implementación de cada una de ellas en cada lugar del sistema donde sean necesarias.

En el momento de escribir este trabajo, Oracle en su versión 11G ya incluye un tipo de dato `WM_PERIOD` para trabajar con periodos, así como sus correspondientes operaciones de comprobación y asignación. Al final de este trabajo se incluye el "ANEXO A: Workspace Manager en Oracle 11 G" donde se describen muy brevemente las principales características de este producto, que pueden servir como referencia de qué funcionalidades puede ser necesario tener en cuenta cuando se debe simular uno mismo este tipo de dato.

4.4.3 Operaciones de consulta y modificación de datos

Las operaciones básicas de manipulación de datos soportadas por un SGBD son de dos tipos, en primer lugar, la consulta de datos y en segundo lugar las operaciones de modificación de datos (inserción, actualización y borrado).

Para el modelo de gestión del tiempo que se está construyendo se debe tener en cuenta ambos tipos de operaciones.

En primer lugar, las operaciones de consulta de datos, que deberán considerar las dimensiones temporales definidas por el modelo para devolver siempre los datos correctos. Por ejemplo, cuando se quiere obtener un listado de líneas telefónicas junto con los datos del cliente al que pertenecen, en un sistema convencional o no-temporal la consulta que hay que escribir, está clara. Peor en el caso de mantener historia versionada, surge la pregunta ¿qué datos del cliente? Pueden ser los actuales. O pueden ser los datos que tenía el cliente cuando la línea se dio de alta. O pueden ser los datos que tenía el cliente en el momento en que se emitió la última factura, etc, etc.

Es de gran importancia en este sentido que todos los involucrados en el proyecto, tanto el equipo técnico como los usuarios, conozcan perfectamente la semántica de los datos a los que se ha añadido los atributos de tipo fecha. Todos deben saber qué representa cada uno de esos atributos para poder llegar a utilizarlos con naturalidad y corrección.

En cuanto a las operaciones de modificación de los datos, como se ha visto en un modelo de versionado, lo que se manipula a nivel lógico son los objetos, es decir, se insertan, actualizan y borran objetos. Pero se deben construir estas operaciones sobre las

operaciones proporcionadas por el SGBD, es decir, inserción, actualización y borrado de filas (o versiones).

Para ello, dentro del patrón de versionado construido, se proporcionará la especificación de cada una de estas operaciones. Habrá operaciones que se traduzcan en la inserción física de una única fila (como se ha visto, la inserción de la primera versión de un objeto, que supone la inserción del objeto) y habrá operaciones que afecten a varias filas. El nivel de complejidad que puede alcanzar la implementación de estas operaciones depende también en gran medida de los aspectos presentados en el apartado "4.4.1 Consideraciones previas" en este mismo capítulo.

4.4.4 Restricciones de integridad

Las restricciones de integridad son las que definen qué se considera un estado consistente de la base de datos. En el modelo de gestión del tiempo que se va a construir las primeras restricciones que se identifican son las restricciones semánticas que forman parte del patrón seleccionado.
Pero además, está la integridad del SGBD que, una vez más, poco puede hacer en su situación actual para garantizar la integridad de los datos cuando se trata de datos temporales. Se ve a continuación cada uno de los componentes en más detalle.

4.4.4.1 Claves principales

Al introducir el concepto de versionado se está modificando el significado de la clave principal de la tabla. Si hasta ahora el significado de una clave principal era "se trata de un valor único", ahora se debe ampliar a "se trata de un valor único en un instante del tiempo". Desafortunadamente el constructor de `PRIMARY KEY` no sirve para tablas versionadas.

El valor de la clave principal debe ser único, es decir, cada valor puede existir como máximo en una fila de la tabla. Cuando se añade la historia pueden existir varias filas con el mismo identificador único. Para manejar esta situación, es necesario añadir una o ambas columnas temporales a la clave principal.

En el caso más sencillo, de versionado unitemporal, existen tres alternativas para definir la clave principal: se puede añadir la fecha de inicio de versión (`fx_ver_ini`), la fecha de fin de versión (`fx_ver_fin`) o ambas. El problema es que ninguna de estas tres alternativas es suficientemente restrictiva, pues ninguna impide que se solapen periodos. Considérense los datos de la tabla siguiente, consistentes con las tres alternativas de posible clave principal existentes. Se puede observar que el periodo de validez de la primera fila incluye el mes de mayo, al igual que la segunda fila:

nro_cliente	nombre_cliente	fx_desde	fx_hasta
C466	Juan	01/01/04	01/06/04
C466	Juan Manuel	01/05/04	01/08/04

Lo que se necesita especificar es que sólo puede existir un cliente C466 en un momento dado. Dicho de otra manera, un cliente no puede tener dos nombres a la vez y la tabla anterior viola esta restricción: durante el mes de mayo, el cliente C466 tenía dos nombres diferentes. Una restricción de clave principal no debería permitir que esto ocurriera pero desafortunadamente, ninguna de las alternativas de clave principal posible lo evitará.

Puesto que no es posible utilizar el SGBD para definir este tipo de clave principal, se debe tener en cuenta en el desarrollo del proyecto incluir código que asegure este tipo de control.

4.4.4.2 Claves ajenas

En tablas no versionadas, las claves ajenas relacionan objetos. Cada fila en la tabla de clientes representa un cliente. Cada fila en la tabla de líneas telefónicas representa una línea telefónica. Lo que hace una clave ajena de la tabla de líneas telefónicas a la tabla de clientes se puede expresar de dos formas:

a) se puede decir que relaciona la fila que representa una línea telefónica, que contiene la clave ajena, con la fila que representa al cliente para esa línea telefónica.

b) se puede decir que relaciona la línea telefónica con el cliente.

La primera forma de decir lo que hace la clave ajena se refiere a los datos, la segunda se refiere a los objetos que representan esos datos. La primera forma, desde el punto de vista de la gestión de los datos, es física. La segunda forma es lógica o semántica.

En tablas no versionadas, existe una relación uno a uno entre los objetos (clientes, líneas telefónicas) y los datos que los representan (filas en tablas). Dado que la relación entre lo que es representado y su representación es uno a uno, no se suele prestar mucha atención a su distinción. Se puede hablar de lo que las claves ajenas relacionan indistintamente como objetos o como filas.

Pero cuando se trabaja con tablas versionadas, la relación uno a uno no se aplica. Y lo que era algo a lo que no se suele prestar atención, ahora se convierte en algo completamente nuevo a tener en consideración.

En general, la preocupación es qué hacer con todas las claves ajenas que hacen referencia a un objeto cuando se inserta en una tabla una nueva versión de dicho objeto.

Y un error muy común al trabajar con versionado es creer que cuando se crea una nueva versión de un objeto, todas las claves ajenas que apuntaban a la versión anterior se deben actualizar para apuntar a la nueva versión, pues parece lógico apuntar siempre a la versión actual del objeto, de hecho, con tablas no temporales es así como funciona.

Sin embargo, si esas actualizaciones tienen lugar en tablas a su vez versionadas, entonces esas actualizaciones crean nuevas versiones adicionales, y todas las claves ajenas que apuntan a ellas también se deben actualizar. De esta forma, una única actualización puede desencadenar actualizaciones en cascada sobre otras tablas versionadas, finalizando únicamente cuando la actualización en cascada alcanza o bien una tabla no versionada o una tabla versionada que no contiene claves ajenas. Y con el riesgo de que existan referencias circulares en la base de datos, lo cual provocaría que se entrase en un bucle infinito.

Con tablas normales (no versionadas), las actualizaciones en cascada no son frecuentes porque tampoco lo son los cambios de claves. Si se está tratando con tablas de clientes y de líneas telefónicas, en lugar de tablas de versiones de clientes y de versiones de líneas telefónicas, no es necesario considerar ninguna situación que pueda requerir una actualización en cascada. Pero dado que se trata de tablas de versiones relacionadas mediante claves ajenas, cada nueva versión de la tabla referenciada (versiones de cliente) requiere una actualización en cascada de la tabla dependiente (versiones de líneas telefónicas).

Se necesita por tanto una alternativa que resuelva el problema de la actualización en cascada, como las siguientes:

a) Hacer las actualizaciones de claves ajenas no versionables, es decir, actualizaciones que sobrescriban la clave ajena y no generen nuevas versiones en el proceso.

   Tanto la tabla de clientes como la de líneas telefónicas son tablas versionadas. Asumiendo que se acaba de crear una nueva versión del cliente C466, a continuación se deben actualizar todas las claves ajenas que apuntaban a la versión reemplazada y hacer que apunten a la nueva versión. Y si se hace como un cambio no versionable, entonces no existe el problema de la actualización en cascada.

   No existe problema en cascada porque el cambio en la clave ajena en la tabla de versiones de línea se implementa mediante una actualización física de la versión actual de la línea telefónica del cliente C466. No se crea una nueva versión para esa línea.

   Esta es una forma de mantener las claves ajenas apuntando a las versiones actuales de los objetos que referencian y al mismo tiempo evitar el problema de la actualización en cascada. Pero se observa un inconveniente de esta solución, y es que se pierde información. La información concreta que se pierde es que hasta este momento la versión de la línea telefónica apuntaba a la versión anterior del cliente,

y no a la que apunta ahora. Un cruce de esa versión de la línea con su versión de cliente relacionada habría mostrado unos datos del cliente diferentes a los que muestra ahora. Esta es la información que se ha perdido.

b) No actualizar las claves ajenas entre versiones. Se asume que se acaba de crear una nueva versión de un cliente. Con esta solución de "no actualización" no es necesaria ninguna otra actividad en la base de datos, ya que no se actualizan las claves ajenas que apuntan a la versión del cliente reemplazada.

   En este caso, se puede definir la siguiente regla de asignación de valor a las claves ajenas: Cuando se crea una nueva versión, asignar valor a sus claves ajenas de forma que apunten a la versión del objeto relacionado que es la actual en el momento en que esta versión se crea.

   Con esta regla, cada clave ajena que apunta a una tabla versionada lo hace a la versión del objeto relacionado que era la versión actual cuando se creó la fila que contiene la clave ajena. Los cruces que utilizan tales claves ajenas versionadas muestran cómo eran las cosas en el momento del tiempo en que la fila que se está cruzando fue creada. Por lo tanto, utilizando esta regla, es posible seguir una cadena de claves ajenas a partir de una fila particular, y obtener las versiones de los objetos relacionados tal como eran en el instante en que la fila fue creada. Es posible, en otras palabras, obtener datos equivalentes a una fotografía de esos datos tomada en el momento en que fue creada la fila original.

   Pero en ocasiones no es eso lo que se quiere. A veces lo que se quiere es obtener la versión actual de todos los objetos relacionados. Y la forma en que se puede hacer es escribiendo consultas que especifiquen el identificador del objeto, el número de cliente en este caso, junto con una cláusula que especifique la mayor fecha de inicio de versión menor o igual que la fecha actual y que contenga una fecha de fin de objeto nula. El calificador de "menor o igual que la fecha actual" es necesario para descartar versiones futuras, versiones que todavía no están en efecto en el momento en que se está ejecutando la consulta. El calificador "que contenga una fecha de fin de objeto nula" es necesario para descartar versiones pertenecientes a objetos que han sido eliminados.

   Se debe tener en cuenta que aun cuando las claves ajenas apuntan a las filas que eran actuales en el momento en la fila que contiene la clave ajena fue creada, esto no garantiza que siempre se devolverán las versiones actuales cuando se creó la fila origen. Estas situaciones se pueden dar porque la relación "actual cuando" no es transitiva, lo cual obliga a prestar especial cuidado a la hora de escribir las consultas para incluir los cruces correctos entre las distintas tablas para obtener los resultados realmente deseados.

Por tanto, se ve que cuando no se actualizan las claves ajenas al sustituir las versiones a las que apuntan, entonces las relaciones implementadas mediante esas claves ajenas van “envejeciendo” lentamente.

Puesto que con la definición convencional de clave ajena no se consigue llegar a una solución satisfactoria, se puede considerar la alternativa de no utilizar las claves ajenas convencionales, sino considerar que las claves ajenas temporales no relacionan filas con filas o versiones con versiones, sino que relacionan objetos. Se pasa entonces a la última alternativa considerada:

c) Almacenar únicamente el identificador del objeto relacionado. Continuando con el ejemplo de clientes y líneas telefónicas, se podría almacenar únicamente el identificador del cliente, sin preocuparse de la fecha de inicio de versión. Posteriormente, cada consulta tendría que especificar explícitamente cuál es la fecha por la que se desea filtrar las versiones de clientes: Si se desea cruzar una línea telefónica con la versión actual del cliente cuando ésta fue creada, la cláusula WHERE incluiría el calificador para obtener la versión del cliente cuya fecha de inicio es la mayor fecha menor o igual a la fecha de inicio de la versión de la línea y cuya fecha de fin de objeto es nula.

   Por otra parte, si se desea cruzar una línea telefónica con la versión actual del cliente asociado, la cláusula `WHERE` incluiría el calificador para obtener la versión del cliente cuya fecha de inicio es la mayor fecha menor o igual que [ahora] y cuya fecha de fin de objeto es nula.

4.4.4.3 Integridad referencial

Sólo se tiene correctamente implementado un modelo de versionado si:

a) se incluyen las columnas definidas para el patrón en las tablas que se desea versionar,

b) se fuerza cada una de sus restricciones semánticas y

c) se implementa la integridad temporal necesaria.

La integridad referencial convencional es la que puede forzar el SGBD basado en declaraciones hechas mediante el DML. Se trata de integridad referencial sobre tablas no versionadas, pero ahora se trata de integridad referencial temporal, es decir de la integridad referencial aplicada sobre tablas versionadas.

Las restricciones de integridad referencial son forzadas a través del SGBD. Pero no es posible utilizar el SGBD para forzar la integridad referencial temporal. Por el momento todavía rige la era del “hágalo usted mismo” y se debe implementar la integridad referencial temporal por uno mismo, al igual que las restricciones específicas que definen cada patrón.

Una restricción de integridad referencial establece que el valor de una determinada columna en todas las filas de la tabla hija debe existir en una fila de la columna referenciada o padre.

La integridad referencial temporal realiza las mismas consideraciones que la integridad referencial pero teniendo en cuenta la dimensión tiempo. Y esta es la definición de la integridad referencial temporal, declarada inicialmente como una restricción que involucra a dos tablas versionadas:

- Si existe una dependencia de integridad referencial temporal de una tabla versionada Y hacia una tabla versionada X (no necesariamente distintas), entonces todo objeto de Y debe estar vinculado mediante una clave ajena a un objeto de X cuyo periodo de validez contenga totalmente al periodo de validez del objeto de Y.

También se debe considerar las restricciones de integridad referencial temporal entre dos tablas, siendo versionada sólo una de ellas (la tabla padre):

- Si existe una dependencia de integridad referencial temporal de una tabla convencional Y a una tabla versionada X, entonces toda fila de Y debe estar vinculada mediante una clave ajena a un objeto de X cuya fecha de inicio de validez es anterior o igual a la fecha en que fue insertada la fila en Y, y cuya fecha de fin de validez es posterior o igual a la fecha en que esa fila fue borrada de Y.

No existe ningún SGBD comercial que proporcione este tipo de integridad referencial. En su lugar, se deja a los desarrolladores la gestión de la integridad referencial temporal mediante su propio código, posiblemente utilizando disparadores, para forzar las siguientes restricciones en la actividad contra la base de datos:

1. Cuando se **inserta** una fila en la tabla versionada Y, esta fila debe hacer referencia a un objeto en la tabla versionada X que:

   a) exista en la base de datos o haya sido insertado como parte de la misma transacción, y

   b) cuyo periodo de validez contenga completamente al periodo de validez del objeto que está siendo insertado en la tabla Y.

2. Cuando se **actualiza** la clave ajena de una fila en la tabla versionada Y, la nueva clave ajena debe hacer referencia a un objeto en la tabla versionada X que:

   a) exista en la base de datos o haya sido insertado como parte de la misma transacción, y

   b) cuyo periodo de validez contenga completamente al periodo de validez del objeto que está siendo actualizado en la tabla Y.

3. Cuando se **borra** una fila en la tabla versionada X, no deben quedar filas en la tabla Y que hagan referencia a la fila borrada. (borrado lógico)
4. Cuando la actualización de una fila en la tabla versionada X tiene como efecto el **cambio en su periodo de validez** (mediante la creación de una nueva versión de X que especifica el nuevo periodo de tiempo), este periodo de validez se puede acortar sólo en la medida en que el periodo de validez de las filas relacionadas de la tabla versionada Y siga contenido dentro de dicho nuevo periodo.

Dicho en pocas palabras, la integridad referencial temporal es la integridad referencial más la restricción de que el periodo de validez de un hijo debe estar contenido en el periodo de validez de su padre.

5 RESULTADOS

Si bien el objetivo definido originalmente para este trabajo consiste en la definición de un modelo de datos con unas características determinadas, se presenta también como resultado del mismo una guía de trabajo útil para abordar proyectos en los que sea preciso incorporar gestión del tiempo. Se definen a continuación ambos resultados.

En primer lugar se ha definido un modelo de gestión del tiempo aplicable a sistemas Data Warehouse que puede ser desarrollado utilizando las capacidades de los SGBDs relacionales y el SQL disponibles actualmente. Dicho modelo aborda los aspectos siguientes:

1. estructuras de datos
2. operaciones de consulta y modificación de datos
3. restricciones

En el modelo de gestión del tiempo definido se han adaptado las **estructuras de datos** para poder almacenar datos que varían con el tiempo y se ha contemplado la ampliación de las **operaciones** de modificación y consulta de datos incorporando semántica temporal. Adicionalmente, para cada **restricción** expresable en un modelo no temporal, se ha considerado la implementación de la versión temporal equivalente.

Como resultado del proceso de definición de dicho modelo, se ha conseguido dar una visión de conjunto de todos los aspectos en los que es necesario pensar a la hora de incorporar gestión del tiempo en un sistema Data Warehouse. Si bien la idea de añadir un par de fechas para registrar el tiempo de validez puede parecer algo bastante evidente, desde luego no resultan tan evidentes aspectos como la necesidad de plantearse desde el inicio del proyecto si el sistema requerirá registrar actividad proactiva o considerar la recurrencia de objetos. Sin embargo, como se ha visto, ambos aspectos se deben tener en cuenta antes de comenzar con cualquier desarrollo para evitar sorpresas desagradables cuando ya es demasiado tarde. Tampoco es evidente la necesidad de considerar un segundo par de fechas para registrar el tiempo de transacción. Ni mucho menos es evidente que va a ser necesario realizar cierto tipo de operaciones con periodos de fechas, así como implementar uno mismo la integridad temporal de los datos.

Y en segundo lugar, se presenta como resultado adicional una guía de trabajo que recopila todo lo tratado en formato de *checklist*, mostrando todos esos aspectos que se deben considerar a la hora de abordar un proyecto en el que sea necesario incorporar gestión del tiempo.

Tal como se sigue a partir del *checklist*, en primer lugar se debe decidir si el sistema va a contemplar o no la actividad proactiva (1.1) y la recurrencia de objetos (1.2). También hay que decidir si se diseñará una estrategia de versionado integrada o separada (1.3).

Estos tres aspectos se deben definir al inicio del proyecto debido al impacto que las diferentes alternativas tienen sobre el diseño posterior del modelo de gestión del tiempo y la complejidad del desarrollo. Descubrir, por ejemplo, que es necesario considerar la recurrencia de objetos cuando ya se encuentra avanzado el desarrollo sin contemplarla puede suponer un auténtico drama.

☐	**1. CONSIDERACIONES PREVIAS**
	☐ 1.1. Actividad proactiva
	☐ 1.2. Recurrencia de objetos
	☐ 1.3. Versionado integrado o separado
☐	**2. ESTRUCTURA DE DATOS**
	☐ 2.1. Estructura de datos (patrón)
	☐ 2.2. Tipo de dato PERIODO
	☐ 2.2.1. Representación
	☐ 2.2.2. Valores especiales
	☐ 2.2.3. Operaciones
☐	**3. OPERACIONES**
	☐ 3.1. Consulta de datos
	☐ 3.2. Modificación de datos
☐	**4. RESTRICCIONES DE INTEGRIDAD**
	☐ 4.1. Restricciones semánticas (patrón)
	☐ 4.2. Gestión de claves principales “temporales”
	☐ 4.3. Gestión de claves ajenas “temporales”
	☐ 4.4. Gestión de la integridad referencial “temporal”

Figura 5.1: Checklist para la gestión temporal

Una vez decididos los puntos anteriores, ya se puede pasar a diseñar el patrón, con su estructura de datos (2.1) y sus restricciones semánticas (4.1). Como parte de la actividad de definición del patrón se diseñarán también las operaciones para la consulta (3.1) y modificación de datos (3.2).

Por otra parte, se definirá la forma de trabajar con los rangos de fechas o periodos (2.2), lo cual incluye el diseño del método de representación que se va a utilizar (2.2.1), de los valores utilizados para fechas especiales (2.2.2) y de las operaciones necesarias para

operar con periodos (2.2.3). Este punto suele ser infravalorado o totalmente obviado por su aparente sencillez. Sin embargo no es comparable la sencillez de establecer qué método de representación se va a utilizar y qué valores especiales se van a utilizar con la complejidad derivada de no hacerlo. En este último caso, lo que ocurrirá será que cada desarrollador adoptará el método que él considere como el mejor, algunos considerarán incluidas en el intervalo las dos fechas, otros no considerarán ninguna de las dos, etc. son cuatro posibilidades diferentes, incompatibles entre ellas. En cuanto a los métodos para recuperar los datos, algunos utilizarán un "`< fecha_fin`" otros un "`<= fecha_fin`", los creativos "`<= fecha_fin + 1 DAY`", alguno añadirá la comprobación de " `... OR fecha_fin IS NULL`", porque para eso él está utilizando `NULL` como fecha especial para indicar "hasta nuevo aviso", etc. Teniendo en cuenta la importancia de la gestión temporal en este tipo de sistemas, sería imperdonable que cualquiera de estas cosas sucediera. Y para ello se debe establecer al inicio del proyecto cuál va a ser el tratamiento único que se va a dar.

Finalmente, se debe diseñar la estrategia a aplicar para mantener la consistencia de la base de datos, para lo cual, además de las restricciones semánticas (4.1) del patrón seleccionado, se deberá implementar la gestión de claves principales (4.2), claves ajenas (4.3) y la integridad referencial (4.4). De nuevo se trata de aspectos que suelen pasar desapercibidos hasta que alguien, ya en fase de desarrollo, se ve obligado a detener un proceso que entró en un bucle infinito por culpa de una actualización en cascada que nadie esperaba. Al principio intentará averiguar qué es lo que hizo mal en su proceso hasta llegar a la conclusión de que quizá no sea tan buena idea utilizar las claves ajenas del SGBD en el caso de la historia versionada.

Resumiendo, a lo largo de este trabajo se ha mostrado la complejidad de la gestión temporal en los sistemas Data Warehouse y la necesidad de abordar su diseño de una forma sistemática, ya que el éxito de un proyecto Data Warehouse depende en gran medida de lo correcta que sea la gestión temporal de la información almacenada en el mismo.

Y disponer de una guía de trabajo y de un modelo de gestión temporal adecuados facilita el camino hacia la construcción del Data Warehouse. Una guía de trabajo y un modelo de gestión temporal, por sí solos, no garantizan el éxito del proyecto pero no disponer de ellos sí contribuirá a su fracaso.

6 CONCLUSIONES

En base a la teoría y los conceptos de las bases de datos temporales y desde el conocimiento de la problemática asociada a la construcción de los sistemas Data Warehouse se ha definido en este trabajo un modelo de gestión temporal para el que se marcaron cuatro objetivos o requisitos considerados necesarios para cubrir la mayor parte de las necesidades de los sistemas de este tipo y que se muestran en la siguiente figura.

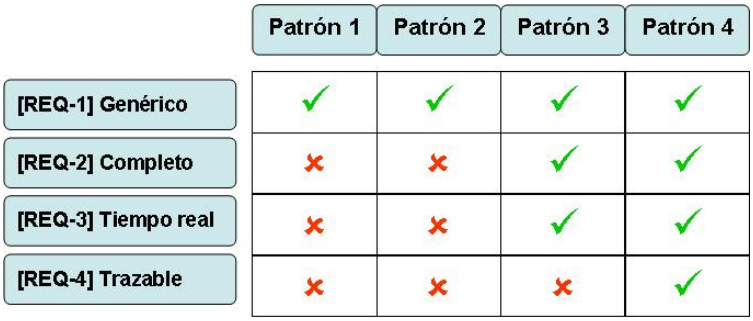


Figura 6.1: Requisitos cubiertos por cada Patrón

En la figura se muestran también los cuatro modelos distintos de gestión del tiempo (los patrones “1”, “2”, “3” y “4”), que se han presentado a lo largo de este trabajo, comenzando por el más simple y construyendo cada uno de los siguientes a partir del modelo anterior, completando su diseño con nuevas funcionalidades temporales.

Tal como se ha visto a lo largo de este trabajo y como se muestra en la Figura 6.1, el patrón “4”, basado en un modelo bitemporal, cumple con todos los requisitos definidos.

El modelo de gestión del tiempo construido en base a este patrón “4” es **genérico**, ya que no depende del sistema ni del tipo de negocio que se está modelando. Es más, debería considerarse su implementación como una capa de gestión temporal independiente que posteriormente pueda ser utilizada por otros sistemas de la empresa, no únicamente por el Data Warehouse. Por otro lado, su implementación puede aplicarse únicamente sobre aquellas áreas del Data Warehouse que requieran una gestión temporal, sin afectar al resto de áreas. También se trata de un modelo **completo** en el sentido de que permite registrar el estado de cualquier objeto en cualquier instante del tiempo. La utilización del concepto de versión del objeto permite mantener una representación de la vida del objeto consistente en el tiempo, capaz de capturar fielmente sus cambios de estado sin perder la identidad del objeto. El enfoque propuesto se trata de una historia consultable en contraposición con una historia reconstruíble, que permite responder las consultas en **tiempo real**, es decir, sin necesidad de acceder a

dispositivos de almacenamiento secundarios o que exijan un procedimiento de actuación que impida recuperar los datos de forma inmediata por los usuarios. Por último, la inclusión de características típicas de los modelos bitemporales, como son el tiempo de validez y el tiempo de transacción, confiere al modelo la funcionalidad de **trazabilidad** ya que permite registrar y por tanto, posteriormente consultar, cuál ha sido la trayectoria de los datos, pudiendo conocer en qué diferentes momentos éstos han sido manipulados, desde su inserción en el sistema, hasta su eliminación, pasando por sus posibles modificaciones y correcciones.

Las ventajas proporcionadas por la integración de un modelo que dé soporte de manera consistente a las necesidades de gestión temporal en un Data Warehouse incluyen un modelado de mayor fidelidad, mejor eficiencia a la hora de reflejar la historia de la organización, así como en el análisis de la secuencia de cambios de dicha historia.

Las exigencias de una gestión temporal adecuada en los sistemas Data Warehouse aumentan día a día debido a que se trata de los sistemas críticos que dan soporte a la toma de las decisiones estratégicas de las compañías y éstas deben estar basadas en información lo más exacta y fiable posible.

Es por tanto fundamental proporcionar a los “trabajadores del conocimiento” un marco de trabajo simple, consistente y eficiente para el Data Warehouse. Ignorar los aspectos temporales da lugar a la disminución en la flexibilidad de la representación de la información y de la capacidad semántica de las consultas en muchos de los escenarios de la vida real.

Casi dos décadas después de que empezaran a surgir los primeros sistemas Data Warehouse, se sigue investigando y aprendiendo sobre la gestión temporal. Y a pesar de que las denominadas bases de datos temporales, área de investigación desde los años 70, no han llegado a consolidarse en productos comerciales, toda su teoría y sus conceptos están sirviendo actualmente para que los SGBDs relacionales comerciales empiecen a incorporar funcionalidad temporal en sus productos.

Pero mientras los SGBDs temporales sigan siendo inexistentes en el ámbito empresarial y la funcionalidad temporal siga sin estar totalmente consolidada en los SGBDs relacionales, la única alternativa existente para abordar la gestión temporal seguirá siendo la del “hágalo usted mismo”. En este sentido este trabajo aporta un enfoque práctico y sistemático sobre cómo abordar un proyecto de este tipo.

7 FUTURAS LÍNEAS DE TRABAJO

A lo largo de este trabajo se ha dado una visión de conjunto de todos los aspectos a tener en cuenta a la hora de considerar la gestión del tiempo en un sistema Data Warehouse. En uno de dichos aspectos, la estructura de datos, se ha entrado en suficiente detalle, llegando a definir una estructura de datos bitemporal partiendo de un modelo prácticamente sin gestión del tiempo y llegando al modelo final paso a paso, explicando el porqué de cada incremento de funcionalidad.

Para el resto de aspectos que constituyen el modelo completo de gestión del tiempo se ha dado una visión general dentro del proceso global de construcción del modelo, pero cada uno de ellos por separado podría ser objeto de un trabajo más detallado, en el que se muestre como implementar cada una de estas funcionalidades sobre los SGBDs relacionales y el SQL "no temporal" disponibles actualmente. Se listan a continuación cuáles son:

- El tipo de dato PERIODO. Constructores y operadores. Como se ha visto, los SGBDs disponibles actualmente no incorporan este tipo de dato, imprescindible cuando el área de interés principal de los sistemas, tal como ocurre en los sistemas Data Warehouse, tiene que ver con la gestión del tiempo.
- Operaciones de consulta y modificación de datos temporales. Mientras los SGBDs no lo incorporen de manera nativa, será necesario construir estas operaciones temporales a partir del SQL no temporal disponible actualmente. Antes de abordar su diseño, se debe pensar en aspectos que van a tener un impacto definitivo en la complejidad de las operaciones que se vayan a implementar, como son:
  - Consideración de actividad proactiva.
  - Consideración de recurrencia de objetos.
  - Consideración de versionado integrado o separado.
- Integridad temporal. Al igual que los dos puntos anteriores, mientras no sean soportados por los SGBDs comerciales, será necesario implementar los equivalentes temporales a las siguientes funcionalidades:
  - Gestión de claves principales temporales.
  - Gestión de claves ajenas temporales.
  - Gestión de integridad referencial temporal.

En otro orden de cosas, en cuanto a otras áreas de conocimiento relacionadas con este trabajo, cabe destacar temas como las bases de datos temporales, la especificación definitiva de un estándar para SQL temporal y finalmente su incorporación a los SGBDs actuales. Todos ellos suponen áreas de investigación activas en la actualidad.

7.1 Modelos de datos temporales

La investigación en el área de las bases de datos temporales ha dado como resultado una diversidad de modelos centrados en aspectos concretos, pero todavía no ha hecho posible la implementación de una base de datos temporal comercial. No obstante, la investigación en esta área sigue avanzando dando lugar a nuevas teorías, conceptos y modelos. Como ejemplo, el modelo presentado por Johnston y Weis [Johnston, 07], que a pesar de denominarse tri-temporal, se trata en realidad de un modelo bitemporal en el que sus autores hacen una nueva lectura semántica del modelo bitemporal y que sirve como muestra de que en gestión del tiempo "no está todo dicho" y que es posible definir nuevos modelos en función de necesidades específicas de cada caso particular. En este caso el primer par de fechas define lo que el modelo bitemporal estándar denomina "tiempo de validez", y el segundo par de fechas define lo que en el modelo tri-temporal se denomina como "tiempo de confirmación de la validez". La novedad en este modelo es el concepto de versiones que se pueden añadir a las tablas antes de "activarlas" o "confirmarlas". Esto es nuevo en la semántica temporal, no soportado por el modelo bitemporal estándar.

7.2 Lenguaje SQL temporal

Sigue sin existir acuerdo sobre cuál será el estándar definido. Por ejemplo, si bien las operaciones de cruce (join), proyección y diferencia temporales fueron presentadas en el trabajo original de Snodgrass [Snodgrass, 00], no ocurre así con otras operaciones, como las funciones de agregación `COUNT`, `MIN`, `MAX` y `AVG`, objeto de estudio en una publicación bastante más reciente [Zimanyi, 06], en la cual el autor destaca la necesidad de utilizar cursores para mejorar el rendimiento de las consultas en su versión temporal, afirmando también, por cierto, que la mejor solución sería que los SGBDs proporcionaran funcionalidad temporal de manera nativa, de cara a incrementar tanto el rendimiento de la base de datos como la productividad del proceso de desarrollo de las aplicaciones.

Por tanto, siguen siendo necesarias nuevas aportaciones e investigaciones, que ayuden a determinar cuál será el estándar finalmente definido.

7.3 Incorporación de SQL temporal en los SGBDs actuales

La necesidad de funcionalidad temporal aumenta día a día en el entorno empresarial, por lo que parece lógico pensar que en el mercado de bases de datos, la línea que se seguirá por parte de los fabricantes de SGBDs relacionales será la de incorporar funcionalidad temporal dentro de sus productos relacionales.

Por el momento sólo Oracle ha anunciado la incorporación de soporte temporal en su SGBD relacional, a través de lo que ha denominado el "Workspace Manager". Debido a su reciente aparición, todavía es demasiado pronto para saber cuál será la aceptación y

la evolución de este módulo de soporte temporal o incluso si acabará marcando cuál será el estándar “de facto” ante la falta de acuerdo para definir un estándar oficial por parte de las organizaciones responsables de ello.

En este contexto, dos de los principales temas de interés sobre los que se debe seguir trabajando son el rendimiento de la base de datos, y en particular la optimización de las consultas temporales. Se trata de una tarea más compleja que en el caso de consultas “no temporales” porque los predicados utilizados en consultas temporales son más difíciles de optimizar. En las aplicaciones de base de datos tradicionales las consultas generalmente especifican predicados de igualdad, rara vez aparecen simultáneamente varios predicados de no igualdad. En contraste, en las consultas temporales es más frecuente que aparezcan varios predicados de no igualdad simultáneamente.

8 ANEXO A: Workspace Manager en Oracle 11 G

Oracle 11G ha empezado a proporcionar soporte para tiempo de validez a través de una nuevo módulo denominado “Workspace Manager”, del que se resumen en este Anexo cuáles son sus principales características relacionadas con la gestión del tiempo [Oracle, 08].

Se trata de un paquete PL/SQL que hace versionables las tablas de usuario. Cuando se activa el soporte para tiempo de validez en una tabla, ésta se renombra y se crea una vista sobre ella con el nombre de la tabla original, lo que hace que este renombrado sea transparente a las aplicaciones. Sobre la tabla renombrada se añade automáticamente una nueva columna para almacenar el periodo de validez asociado con cada una de las filas, lo que permite que existan en la tabla múltiples versiones de una fila con la misma clave principal definida por el usuario. Se puede especificar un periodo de validez para la sesión y el Workspace Manager asegurará que las operaciones de consulta, inserción, actualización y borrado reflejan correctamente el periodo de validez. El periodo de validez especificado puede estar en el pasado o en el futuro o puede incluir pasado, presente y futuro.

El Workspace Manager sólo copia una fila si ésta ha sufrido algún cambio. Esto puede reducir significativamente el hardware, el software y el tiempo necesarios para gestionar múltiples versiones de los datos, en comparación con otros escenarios en los que se llevan a cabo cargas masivas sincronizadas de datos. También incrementa la productividad, al permitir el acceso concurrente a diferentes versiones de los datos y proporcionar un único punto de acceso para gestionar todas las versiones de los datos, liberando a los desarrolladores de las aplicaciones de la tarea de escribir código propio para gestionar múltiples versiones de los datos.

El Workspace Manager también proporciona una opción de historia para tablas versionadas que etiqueta las versiones con el tiempo de transacción. Este es el instante en que realmente fueron introducidos los datos y permite a los usuarios ir a una fecha determinada y ver la base de datos tal como era para un tiempo de transacción particular.

Para proporcionar este soporte, Oracle incorpora un nuevo tipo de dato, `WM_PERIOD`, así como nuevas constantes, operadores de comprobación, operadores de asignación y tratamiento de la integridad. A continuación se describe brevemente cada una de estas características:

8.1 Tipo de dato WM_PERIOD

El tipo de dato `WM_PERIOD` se utiliza para especificar un periodo de validez para una fila en una tabla versionada. Se define de la siguiente forma:

```
CREATE TYPE WM_PERIOD AS OBJECT  (
  validFrom  TIMESTAMP WITH TIME ZONE,
  validTill  TIMESTAMP WITH TIME ZONE);
```

La fecha `validFrom` está incluida y la fecha `validTill` está excluida, esto es, el periodo de validez comienza en la fecha `validFrom` y se extiende hasta (pero sin incluir) la fecha `validTill`.

8.2 Constantes

Constantes que se pueden utilizar como fechas `validFrom` y `validTill` en la especificación de una variable `WM_PERIOD` son:

- **`DBMS_WM.MIN_TIME`:** Es el valor mínimo definido para un timestamp. Su valor es el día 1 de Enero del año -4712 (4712 BCE).
- **`DBMS_WM.MAX_TIME`:** Es el valor máximo definido para un timestamp. Su valor es el instante final (11:59.999999000 pm) del día 31 de Diciembre del año 9999.
- **`DBMS_WM.UNTIL_CHANGED`:** Es un timestamp que es tratado como `DBMS_WM.MAX_TIME` hasta que es sobrescrito por una modificación posterior.

8.3 Operadores

Se definen operadores de comprobación y de asignación que aceptan dos periodos como parámetros y que se pueden utilizar para aplicar filtros de tiempo de validez en las consultas.

Los operadores de comprobación de relaciones devuelven el valor entero 1 si existe la relación entre los dos periodos y devuelven 0 en caso contrario:

- **`WM_OVERLAPS`** comprueba si dos periodos se solapan.
- **`WM_CONTAINS`** comprueba si el primer periodo contiene al segundo periodo.
- **`WM_MEETS`** comprueba si el final del primer periodo es el comienzo del segundo periodo.
- **`WM_EQUALS`** comprueba si los dos periodos son iguales.
- **`WM_LESSTHAN`** comprueba si el final del primer periodo es anterior al comienzo del segundo periodo.
- **`WM_GREATERTHAN`** comprueba si el comienzo del primer periodo es posterior al final del segundo periodo.

Los operadores de asignación devuelven el periodo que refleja la relación entre los dos periodos, o valor `NULL` si los dos periodos no cumplen la relación especificada:

- **WM_INTERSECTION** devuelve la intersección de los dos periodos, es decir, el rango de tiempo común a ambos periodos.
- **WM_LDIFF** devuelve la diferencia por la izquierda de los dos periodos.
- **WM_RDIFF** devuelve la diferencia por la derecha de los dos periodos.

Es posible utilizar los operadores de comprobación de relación en lugar de los atributos `wm_valid.validFrom` y `wm_valid.validTill` de la fila.

A continuación se muestra información adicional sobre cada uno de los operadores, que se listan en orden alfabético.

WM_CONTAINS

El operador `WM_CONTAINS` comprueba si el primer periodo contiene al segundo periodo. `WM_CONTAINS(p1,p2)` devuelve 1 si el periodo `p1` contiene por completo al periodo `p2`, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_CONTAINS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1988', 'MM-DD-YYYY'))) = 1
```

```
WM_CONTAINS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0
```

WM_EQUALS

El operador `WM_EQUALS` comprueba si el primer periodo es igual al segundo periodo. `WM_EQUALS(p1,p2)` devuelve 1 si el periodo `p1` es igual al periodo `p2`, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_EQUALS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY'))) = 1
```

```
WM_EQUALS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0
```

WM_GREATERTHAN

El operador `WM_GREATERTHAN` comprueba si el primer periodo es posterior al segundo periodo. `WM_GREATERTHAN(p1, p2)` devuelve 1 si el periodo completo `p1` ocurre después del periodo `p2`, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_GREATERTHAN(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1970', 'MM-DD-YYYY'),
      TO_DATE('01-01-1980', 'MM-DD-YYYY'))) = 1
```

```
WM_GREATERTHAN(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1970', 'MM-DD-YYYY'),
      TO_DATE('01-01-1981', 'MM-DD-YYYY'))) = 0
```

WM_INTERSECTION

El operador `WM_INTERSECTION` devuelve la intersección de los dos periodos, es decir, el periodo común a ambos. `WM_INTERSECTION(p1,p2)` devuelve el periodo común a `p1` y `p2`.

El siguiente ejemplo devuelve el periodo que se extiende desde 01-Enero-1985 hasta 01-Enero-1988:

```
WM_INTERSECTION(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1988', 'MM-DD-YYYY')))
```

El siguiente ejemplo devuelve el periodo entre 01-Enero-1985 y 01-Enero-1990:

```
WM_INTERSECTION(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

El siguiente ejemplo devuelve `NULL` porque no existe intersección entre los periodos:

```
WM_INTERSECTION(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1992', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

WM_LDIFF

El operador `WM_LDIFF` devuelve la diferencia por la izquierda entre los dos periodos. `WM_LDIFF(p1,p2)` devuelve el periodo que `p1` sobrepasa a `p2` por la izquierda.

El siguiente ejemplo devuelve el periodo entre el 01-Enero-1980 y 01-Enero-1985:

```
WM_LDIFF(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1988', 'MM-DD-YYYY')))
```

El ejemplo siguiente devuelve un valor `NULL` porque `p1.validFrom` es mayor que `p2.validFrom`:

```
WM_LDIFF(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1975', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

El siguiente ejemplo devuelve un valor `NULL` porque `p2` está completamente separado de (en este caso, es posterior a) `p1`:

```
WM_LDIFF(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1992', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

WM_LESSTHAN

El operador `WM_LESSTHAN` comprueba si el primer periodo es anterior al segundo periodo. `WM_LESSTHAN(p1,p2)` devuelve 1 si el periodo completo `p1` ocurre antes del periodo `p2`, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_LESSTHAN(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1991', 'MM-DD-YYYY'),
      TO_DATE('01-01-1992', 'MM-DD-YYYY'))) = 1
```

```
WM_LESSTHAN(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1989', 'MM-DD-YYYY'),
      TO_DATE('01-01-1992', 'MM-DD-YYYY'))) = 0
```

WM_MEETS

El operador `WM_MEETS` comprueba si el final del primer periodo es el comienzo del segundo periodo. `WM_MEETS(p1,p2)` devuelve 1 si `p1.validTill = p2.validFrom`, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_MEETS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1990', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 1
```

```
WM_MEETS(
   WM_PERIOD(
```

```
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1992', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0
```

WM_OVERLAPS

El operador `WM_OVERLAPS` comprueba si los dos periodos se solapan. `WM_OVERLAPS(p1,p2)` devuelve 1 si los periodos `p1` y `p2` se solapan, y devuelve 0 en caso contrario. Por ejemplo:

```
WM_OVERLAPS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 1
```

```
WM_OVERLAPS(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1970', 'MM-DD-YYYY'),
      TO_DATE('01-01-1980', 'MM-DD-YYYY'))) = 0
```

WM_RDIFF

El operador `WM_RDIFF` devuelve la diferencia por la derecha entre los dos periodos. `WM_RDIFF(p1, p2)` devuelve el periodo que `p1` sobrepasa a `p2` por la derecha.

El siguiente ejemplo devuelve el periodo entre 01-Ene-1988 y 01-Enero-1990:

```
WM_RDIFF(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
```

```
   WM_PERIOD(
      TO_DATE('01-01-1985', 'MM-DD-YYYY'),
      TO_DATE('01-01-1988', 'MM-DD-YYYY')))
```

El siguiente ejemplo devuelve un valor `NULL` porque `p1.validTill` es menor que `p2.validTill`:

```
WM_RDIFF(
   WM_PERIOD(
      TO_DATE('01-01-1980', 'MM-DD-YYYY'),
      TO_DATE('01-01-1990', 'MM-DD-YYYY')),
   WM_PERIOD(
      TO_DATE('01-01-1975', 'MM-DD-YYYY'),
      TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

8.4 Gestión de restricciones

Esta sección describe consideraciones relativas al soporte de tiempo de validez que afecta a las restricciones de integridad referencial y de unicidad.

8.4.1 Restricciones de integridad referencial

Si existe una restricción de integridad referencial entre dos tablas versionadas, entonces se tienen en cuenta los periodos de validez a la hora de forzar dicha restricción. Por ejemplo, se asume que la tabla `DEPARTAMENTS` tiene una columna `MANAGER_ID` que es una clave ajena que hace referencia a la columna `EMPLOYEE_ID` de la tabla `EMPLOYEES` (es decir, el manager del departamento debe ser un empleado existente). Si ambas tablas son versionadas y una operación de inserción o actualización da como resultado un nuevo valor para `DEPARTAMENTS.MANAGER_ID`, esta operación fallará si el valor de `DEPARTAMENTS.WM_VALID` no está contenido dentro del valor `EMPLOYEES.WM_VALID` para el empleado que se está considerando manager del departamento. Es decir, la operación fallará si el nuevo manager del departamento no es un empleado válido durante el periodo especificado para la operación de inserción o actualización.

Sólo se tienen en consideración los periodos de validez cuando ambas tablas son versionadas.

8.4.2 Restricciones de unicidad

Si existe una restricción de unicidad en una tabla versionada, entonces se tienen en cuenta los periodos de validez a la hora de forzar dicha restricción. Por ejemplo, se asume que la tabla `EMPLOYEES` tiene una columna `EMPLOYEE_ID` que tiene una restricción de unicidad. Si una operación de inserción o actualización da como resultado un `EMPLOYEE_ID` ya existente, la operación fallará si se solapan los valores de `WM_VALID` existente e insertado. Es decir, la operación fallará si el nuevo empleado y un empleado existente tiene el mismo identificador de empleado y sus periodos de validez se solapan. Y la operación será correcta si ambos periodos no se solapan.

9 BIBILIOGRAFÍA

[1] [Allen, 83] J. F. Allen. “Maintaining knowledge about temporal intervals”. Communications of the ACM,26(11):832–843, 1983.

[2] [Bischoff, 97] Joyce Bischoff. “Data Warehouse. Practical Advice from the Experts”, Prentice Hall, 1997

[3] [Date, 02] C. J. Date, Hugh Darwen and Nikos Lorentzos. “Temporal Data and the Relational Model”. San Francisco: Morgan-Kaufmann, 2002.

[4] [Inmon, 94] W.H. Inmon, Richard D. Hackathorn. “Using the Data Warehouse", John Wiley, 1994

[5] [Inmon, 95] W.H. Inmon. “What is a Data Warehouse?" Prism, Volume 1, Number 1, 1995

[6] [Johnston, 07] Tom Johnston and Randall Weis. “Managing time in relational databases”, DM Review, Mayo 2007

[7] [Kimball, 96] Ralph Kimball. “The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses”, John Wiley, 1996

[8] [Kimball, 97] Ralph Kimball, “A dimensional modelling manifesto”, DBMS Magazine, 1997

[9] [Oracle, 08] Oracle® Database Workspace Manager Developer's Guide 11g Release 1 (11.1), Agosto 2008

[10] [Snodgrass, 94] R. T. Snodgrass et al. “TSQL2 Language Specification”. SIGMOD Record, Marzo 1994.

[11] [Snodgrass, 00] R. T. Snodgrass. “Developing Time-Oriented Applications in SQL”. San Francisco: Morgan-Kaufmann, 2000. (ISBN: 1558604367)

[12] [Steiner, 98] Andreas Steiner. “A Generalisation Approach to Temporal Data Models and their Implementations”, Ph. D. thesis, 1998.

[13] [Wuu, 93] Wuu, G.T.J and Dayal, U. “A Uniform Model for Temporal and Versioned Object-Oriented Databases”, In: Temporal Databases: Theory, Design, and Implementation, Edited by A.Tansel et al, Benjamin/Cummings, p. 230-247. 1993.

[14] [Zimanyi, 06] Zimanyi, E. “Temporal Aggregates and Temporal Universal Quantification in Standard SQL”, SIGMOD Record 35 (2), Junio 2006

9.1 Referencias de Internet

Se listan a continuación las referencias de Internet consultadas para la elaboración de este trabajo:

- [inmon] http://www.inmoncif.com/

  Sitio web de Bill Inmon sobre Data Warehouse.

- [datawarehousing] http://www.datawarehousing.com

  Sitio web de IBM sobre Inteligencia de Negocio y Data Warehouse.

- [dw-institute] http://www.tdwi.org

  TDWI (The Data Warehousing Institute™) proporciona formación, certificación, noticias e investigaciones para profesionales de las TI. Fundado en 1995, TDWI es la principal institución educativa sobre Inteligencia de Negocio y Data Warehousing. Cuenta con el soporte de numerosos Partners entre los que se encuentran HP, IBM, Microsoft, Microstrategy, SAP, SAS, Sybase y Teradata.

- [kimball] http://www.rkimball.com

  Sitio web del "Kimball Group", este grupo fue fundado en 2003 para responder a la creciente demanda de consultoría y formación sobre Data Warehouse. Liderado por Ralph Kimball, está formado por un equipo de profesionales que durante más de dos décadas han estado dedicados al desarrollo, la enseñanza y la aplicación técnicas de diseño de Data Warehouse a través de sus publicaciones y actividades de formación y consultoría.

- [jtemporal] http://jtemporal.sourceforge.net/

  Página de inicio del proyecto de código abierto JTemporal. Se trata de un marco de trabajo de código abierto de componentes que proporciona funcionalidad relativa al tiempo. Basado en la teoría de [Snodgrass, 00].