Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Desarrollo de una herramienta para la generación automática de modelos de datos relacionales en sistemas BI

Autor:
Santiago Pérez Lombas

Tutores:
José Belarmino Pulido Junquera
Anibal Bregón Bregón

Resumen

En el ámbito del *Business Intelligence* es muy común el uso de aplicaciones que recogen y presentan la información del negocio para el apoyo a la toma de decisiones. Su correcto funcionamiento depende de la disposición de una capa de datos preparada para ser explotada dentro del sistema de información de la organización.

En este proyecto se propone y desarrolla una herramienta que automatiza la generación y la carga de modelos de datos relacionales que puedan servir como fuentes de datos para estas aplicaciones. Mediante la exposición de un caso de negocio concreto, se han identificado las diferentes funcionalidades que debe cubrir la solución.

El proyecto se ha desarrollado dentro de un marco de trabajo guiado por dos metodologías: Proceso Unificado y Arquitectura Dirigida por Modelos. La herramienta ha sido diseñada en la tecnología de Microsoft BI, utilizando Transact-SQL para su implementación. Finalmente, los artefactos derivados del desarrollo pueden ayudar a construir más fácilmente esta herramienta en otros sistemas de información.

Abstract

The use of applications that make insights from business information to support decision making is a really common practice in Business Intelligence. Its proper functioning depends a great deal on the existence of a layer of data within the organization's information system specifically made to be exploited.

The purpose of this project is the proposal and development of a tool which automates the generation and load of relational data models. These models can then serve as data sources to the mentioned applications. The diverse functionalities of this solution have been identified by the analysis of a particular business case.

The project has been developed within a framework guided by both Unified Process and Model-Driven Architecture methodologies. The tool has been designed using Microsoft BI technology and implemented with Transact-SQL. Finally, the artifacts driven from the development may help to build this tool straightforwardly in different information systems.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

La capacidad de tratamiento y análisis de datos para obtener información útil es un proceso que cada vez cobra mayor relevancia en muchos sectores. Hoy en día, desde pequeñas empresas a grandes instituciones mantienen un registro exhaustivo de todas las actividades que realizan, lo que da lugar a la generación de ingentes cantidades de datos. Por ello, el proceso de captura de información provechosa a partir de este mar de datos es una tarea cada vez más difícil.

El término **Inteligencia de Negocio o *Business Intelligence* (BI)** se ha consolidado como el conjunto de metodologías, técnicas o estrategias que tratan de transformar los datos en información que ayude a la toma de decisiones (Díaz, 2010). La introducción del BI ha tenido como consecuencia grandes cambios en los sistemas de almacenamiento de datos del mundo empresarial, y a día de hoy su implantación en las compañías se ha convertido en un requisito para poder competir en el mercado (Wixom y Watson, 2010). Nuevas arquitecturas y herramientas han surgido apostando en esta visión más enfocada al apoyo de la toma de decisiones, intentando optimizar los procesos de búsqueda y presentación de la información.

Este proyecto nace a raíz de un trabajo de auditoría de la plataforma BI de almacenamiento y gestión de los datos de una empresa. Esta compañía había trabajado en la implantación de un sistema de información para la integración y almacenamiento de datos internos y externos y su explotación mediante diferentes aplicaciones BI. Sin embargo, la plataforma implantada no estaba cumpliendo las expectativas de los usuarios del sistema, por lo que se resolvió solicitar la auditoría para detectar las carencias del sistema y definir acciones a realizar para solventarlas.

Tras una revisión exhaustiva del estado de esta plataforma, se determinó que resultaba ser de difícil explotación y de baja fiabilidad para los usuarios de negocio, por lo que se planteó una reingeniería para obtener un sistema más robusto y confiable. Este proyecto incide sobre uno de los puntos de mejora incorporados en el alcance de la auditoría mediante la propuesta de una herramienta que facilite la obtención de los datos para su análisis en los sistemas BI.

1.1. Sistemas BI

Antes de comenzar con la exposición de la situación actual de la plataforma y su arquitectura concreta, haremos un breve resumen de las tecnologías o técnicas más comunes para construir sistemas de la información que siguen un enfoque en línea con el BI.

Aunque cada vez es mayor el auge de las arquitecturas que siguen un esquema no relacional para las bases de datos, nos centraremos en los **Sistemas de Gestión de Bases de Datos Relacionales (RDBMS)**, que son indiscutiblemente el software más utilizado para el almacenamiento y gestión de los datos.

Tradicionalmente, los sistemas construidos por las compañías constituían bases de datos relaciones para una gestión más operacional de los datos. Éstos son llamados sistemas de **Procesamiento de Transacciones en Línea (OLTP)** y están diseñados para el procesado y almacenamiento de datos con el mayor nivel de detalle, es decir, las transacciones realizadas en el día a día de la compañía (Hernandez-Orallo, 2006). Las consultas que se realizan en el análisis operacional suelen ser simples, rápidas y repetitivas, y los registros no son almacenados durante un largo periodo de tiempo.

Sin embargo, con el paso de los años y la introducción del BI, las tecnologías de **Procesamiento Analítico en Línea (OLAP)** han crecido en importancia, con una nueva visión más centrada en utilizar la información almacenada para la toma de decisiones. Este enfoque requiere de la obtención de información en forma de resúmenes o agregados de muchas transacciones. Un **agregado** es una operación aritmética o relacional realizada sobre un conjunto de registros que contienen datos en el máximo nivel de detalle (Larson, 2017). Por ello, las consultas que se realizan sobre los sistemas OLAP son mucho más complejas e involucran un mayor número de entidades y los datos son almacenados durante largos periodos de tiempo (Alvi, 2019).

La arquitectura OLAP más extendida y totalmente aceptada en el mundo empresarial es el ***Data Warehouse* (DW)**, introducida por Kimball y Merz (1993). Esta metodología defiende un esquema para la construcción de almacenes de datos llamado modelado dimensional. La técnica de modelado dimensional divide las diferentes entidades del DW en **hechos**, que almacenan métricas de la compañía a un nivel de detalle concreto, y **dimensiones**, que contienen los datos que ponen en contexto los registros de las tablas de hechos.

Las **métricas** son indicadores numéricos que miden el rendimiento de un aspecto del negocio para facilitar la toma de decisiones, y se pueden obtener a partir de agregados (Larson, 2017). El nivel de detalle con el que se calculan los agregados en las tablas de hechos y con el que se almacenan las dimensiones se denomina **granularidad**. Cada hecho contiene múltiples claves apuntando a registros de dimensiones, que conforman el contexto del hecho registrado, por lo que los hechos se almacenan agregados con el máximo detalle posible que permiten las dimensiones según su granularidad. Las **jerarquías** son estructuras que definen la navegación entre los diferentes niveles de granularidad de las dimensiones.

Siguiendo el ejemplo de Kimball y Ross (2013), en una base de datos de un negocio la tabla de hechos de Ventas contendrá un campo con el valor de la venta (métrica) junto con múltiples campos de las dimensiones de Producto, Fecha, Tienda, Cliente, etc. Centrándonos en la dimensión Fecha, posiblemente presente una granularidad diaria, por lo que las ventas están agregadas

diariamente. Además, si la dimensión Fecha posee la suficiente información se podrían agregar con una granularidad semanal, mensual, trimestral, etc, siguiendo una jerarquía temporal. Esta organización facilita, por tanto, la comprensión y el acceso de la información almacenada, la obtención de agregados de datos, la navegación entre diferentes niveles de detalle y otras operaciones que ayudan en el análisis para la toma de decisiones.

En la Figura 1.1 podemos ver la estructura básica de la arquitectura DW. En ella se distingue el flujo de datos desde los orígenes, que suelen ser fuentes externas y bases de datos OLTP con datos generados internamente, hasta que son capturados y transformados por procesos de **Extracción, Transformación y Carga (ETL)**. Como su nombre indica, las operaciones ETL se encargan de extraer los datos de las fuentes y realizar las transformaciones necesarias sobre ellos, como corregir errores, tratar elementos ausentes, limpiar datos, resolver conflictos entre registros, etc. Una vez transformados los datos, se cargan físicamente en el modelo dimensional del DW. Las aplicaciones BI hacen sus consultas al DW para obtener los datos pertinentes en el análisis.

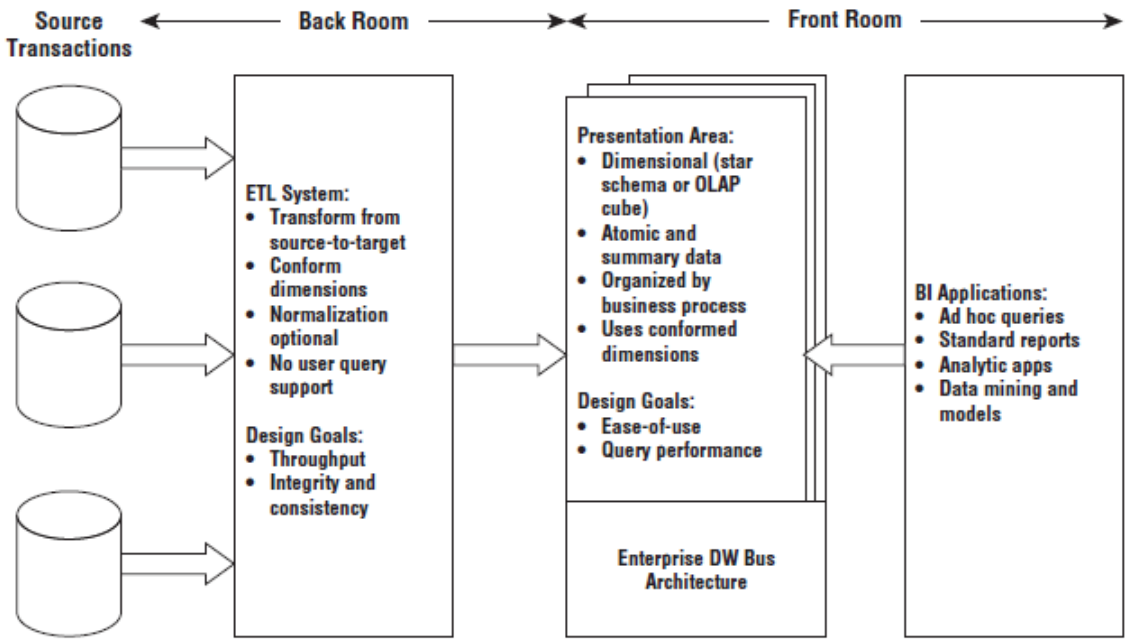

Figura 1.1.: Arquitectura básica DW (Kimball y Ross, 2013)

Sin embargo, existen múltiples variaciones de esta arquitectura que difieren en varios aspectos, como el nivel de normalización de las dimensiones, que cambia según el esquema que se sigue en la estructura de las dimensiones. Un esquema más normalizado elimina redundancias en los registros de las dimensiones y simplifica los procesos de carga y actualización de los datos, en detrimento de la eficiencia de las consultas y la simplicidad del modelo, al suponer un mayor número de entidades. Los esquemas más comunes son **estrella (*star*)**, con menor nivel de normalización y **copo de nieve (*snowflake*)**, con mayor nivel de normalización (Badia, 2006).

Igualmente, es usual encontrar arquitecturas más complejas con capas intermedias en el flujo de datos, como **zonas de aterrizaje (*landing area*)** o **áreas de pruebas (*staging area*)**. Estas capas conforman un espacio de trabajo para el apoyo de los procesos de transformación de datos que tienen lugar en los ETL, y mantienen temporalmente datos y tablas auxiliares (Ponniah, 2001). A su vez, hay quienes defienden arquitecturas con características propias tanto de sistemas OLAP como OLTP (Plattner, 2009).

El diseño de un DW está enfocado en almacenar la información del negocio de forma que facilite la toma de decisiones en base a las métricas definidas por la compañía. Debe ser construido siguiendo una visión global del negocio, más centrada en capturar los procesos que tienen lugar en el mismo y dar robustez y fiabilidad de la información almacenada. Sin embargo, falla en proporcionar agilidad y adaptarse a las necesidades cambiantes de explotación de esta información (Knabke y Olbrich, 2013).

Es por ello que algunas arquitecturas incorporan, dentro de la estructura del DW, otra capa de datos para el soporte de la explotación de la información. Esta capa, al contrario que el modelo base, puede disponer de un modelo de datos que esté completamente al servicio de las necesidades analíticas de los usuarios de la compañía. Algunos ejemplos habituales de estas necesidades son los siguientes:

- **Generar dimensiones desnormalizadas:** muchas de las consultas lanzadas sobre el almacén de datos por las tareas BI requieren la desnormalización de las dimensiones del modelo, lo que supone una alta disminución del rendimiento y la repetición de operaciones (Sanders y Shin, 2001). Estableciendo dimensiones desnormalizadas a partir de las normalizadas, se podrá reducir el coste de estas consultas.
- **Generar fotos de hechos o dimensiones:** Los *snapshots* o fotos representan una captura o instantánea del estado de los datos del sistema en un momento determinado del tiempo. Mantener fotos es muy útil cuando se necesita consultar la información que estaba almacenada en puntos anteriores del tiempo. Es diferente del concepto de hechos con fotos periódicas, que contienen hechos agregados de otras tablas a intervalos de tiempo, mientras que una instantánea de una entidad del modelo presenta los datos exactos de esa entidad en el momento que se realizó.
- **Generar nuevos hechos:** según la necesidades de explotación de datos, se podrá requerir de nuevas tablas de hechos calculadas a partir de otras del modelo. Estas tablas pueden contener resúmenes y agregados de los hechos del modelo base. Las tablas de **hechos con fotos periódicas** contienen hechos agregados en intervalos constantes del tiempo, por ejemplo diaria, semanal o mensualmente y son muy útiles para el análisis de la tendencia de una métrica (Kimball y cols., 2007). También pueden incluir cálculos de nuevas métricas y KPI (*key performance indicator*), indicadores del rendimiento de cualquier tipo de proceso en el negocio.

De esta forma, la adición de un modelo de datos en el sistema de información por encima del DW, al estar más preparado para presentar los datos que se utilizan en el análisis, podría aumentar el rendimiento de las aplicaciones que explotan esos datos en gran medida. A cambio, la implementación del modelo en base de datos necesita de un mayor espacio de almacenamiento por la replicación de la información. Esta capa en muchos sistemas BI se materializa en un conjunto de *Data Marts*.

Un ***Data Mart*** **(DM)** es una base de datos generalmente relacional y dimensional que contiene información de un aspecto concreto del negocio. Algunos autores como Larson (2017) abogan por una arquitectura de DM construidos independientemente para cada departamento de la compañía, cuya información es recogida directamente de las fuentes. Aunque esta solución es más sencilla que la implementación de un DW, ya que no requiere de una organización interdepartamental, a largo plazo es perjudicial para el negocio: no existe una gobernanza global de los datos, se desperdicia espacio al almacenar los mismos datos en diferentes departamentos y son poco extensibles y difíciles de integrar (Kimball y Ross, 2013).

Por otra parte, los DM se pueden integrar en la estructura del DW para formar otra capa de datos de la que se sirven las operaciones de BI. De esta forma, los DM, que pueden ser implementados con un enfoque más cercano a las necesidades de análisis de los usuarios, se cargan a partir del DW corporativo. El DW, aunque puede ser consultado por los departamentos, al no tener que maximizar la eficiencia en las consultas, puede estar más normalizado y contener sólo registros atómicos y no agregados de las métricas. En la Figura 1.2 se puede ver el esquema de Kimball y Ross (2013) de esta arquitectura.

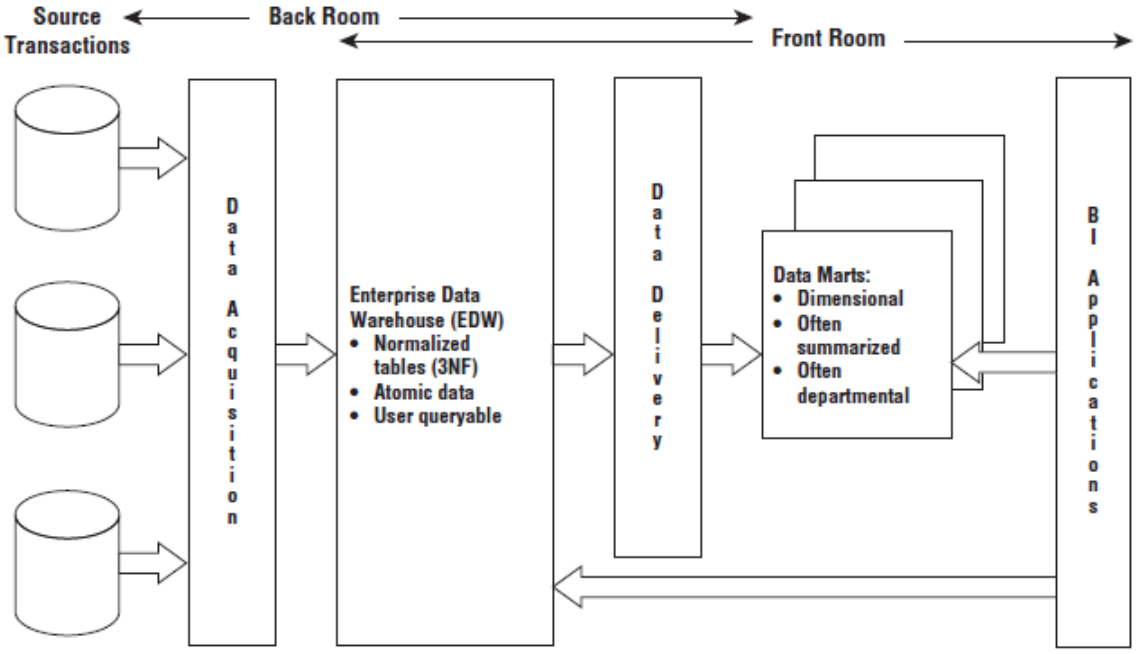

Figura 1.2.: Arquitectura DW con DM departamentales (Kimball y Ross, 2013)

Las **aplicaciones BI** permiten hacer efectiva la toma de decisiones mediante la exposición de la información obtenida a partir de los datos del DW proporcionados a través de uno o más DM. Una de las tareas más comunes de las aplicaciones BI es la presentación de la información en cuadros de mando. Un cuadro de mando permite realizar el seguimiento de la evolución del negocio agrupando y presentando de forma visual y legible la información relevante para la compañía. Otra tarea relevante es el *reporting* o sistema de generación de informes, que facilita la distribución del conocimiento empresarial personalizado a las diferentes áreas de la organización (Negash y Gray, 2008).

El término ***Business Analytics* (BA)** hace referencia al conjunto de tecnologías y técnicas de exploración y análisis de datos para obtener un visión más completa y profunda del negocio y una predicción de su estado futuro para lograr una ventaja competitiva. El BA va más allá de la visualización de datos en cuadros de mando y reporting, aplicando técnicas estadísticas, análisis predictivos y minería de datos (Albright y Winston, 2018). Las llamadas **plataformas *Analytics and Business Intelligence* (ABI)** son tecnologías que intentan aunar, en una misma herramienta, funcionalidades para la exploración visual de datos en cuadros de mando e informes y su exploración y análisis por medio de técnicas características del BA (Richardson y cols., 2020). Algunas de las funcionalidades que cada vez poseen cada vez mayor auge entre las plataformas ABI son la visualización de datos en tiempo real, la realidad aumentada o consultas en lenguaje natural.

Muchas de estas tecnologías siguen un enfoque **multidimensional** para la exploración de los datos. Conceptualmente, se puede representar este enfoque por medio de cubos multidimensionales, llamados **cubos OLAP**, donde cada dimensión del cubo supone una dimensión de la métrica que se está midiendo. La Figura 1.3 representa un cubo OLAP para los datos de las ventas según las dimensiones categoría, localización del cliente y tiempo (Post, 2004).

Este planteamiento de los datos define exactamente las dimensiones y jerarquías en los datos y facilita algunas de las operaciones más utilizadas en las tareas analíticas (Silberschatz y cols., 2011): *rollup* y *drill-down*, denominación en inglés a las operaciones de disminuir o aumentar la granularidad en las jerarquías de las dimensiones (por ejemplo, en la dimensión de tiempo, navegar entre las jerarquías día, semana, mes y año), *pivoting* (pivotar por alguna de las dimensiones) y *slice-and-dice* (seleccionar valores de la métrica correspondientes a diferentes subconjuntos de valores de las dimensiones).

Los servidores OLAP tratan de agilizar el tiempo de procesamiento de estas operaciones mediante la generación de un nuevo modelo de datos con una estructura multidimensional en base a este concepto, tomando como origen el modelo relacional del DW o los DM. Existen diferencias en la forma de implementación de este modelo (Tamayo y Moreno, 2006):

- **ROLAP:** los datos son almacenados en una base de datos relacional. El servidor se encarga de transformar dinámicamente los análisis multidimensionales realizados en las tareas de BI

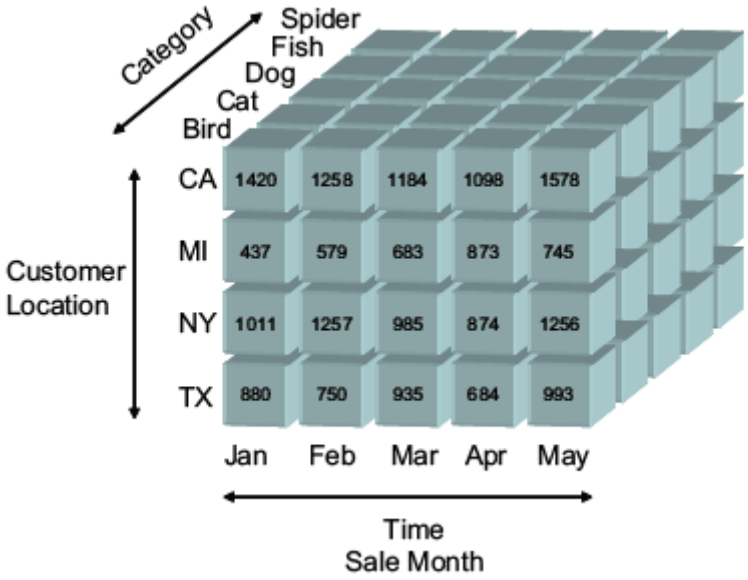

Figura 1.3.: Ejemplo de Cubo OLAP (Post, 2004)

en consultas SQL que se ejecutan sobre la base de datos relacional.

- **MOLAP:** los datos, en lugar de almacenarse en un sistema relacional, son mantenidos en una estructura de datos multidimensional. Las consultas multidimensionales pueden realizarse directamente sobre esta estructura de datos, que está enfocada en la optimización de este tipo de análisis.
- **HOLAP:** un enfoque híbrido de los dos anteriores. Los registros más detallados de datos son almacenados en una base de datos relacional, mientras que agregados de datos se mantienen en una estructura multidimensional.

Los servidores ROLAP aprovechan todos los beneficios obvios que supone utilizar una estructura relacional, en cuanto a la escalabilidad y almacenamiento (inserciones y actualizaciones). Sin embargo, en algunos casos la transformación de las consultas multidimensionales en SQL no son tan evidentes, y tienden a disminuir el rendimiento. Un intento para facilitar esta traducción es la extensión del estándar de SQL para agregar los operadores *cube* y *rollup* (Deshpande y Ramasamy, 2006).

Los sistemas MOLAP, por otra parte, consiguen resultados muy buenos en cuanto a optimización de consultas multidimensionales. MultiDimensional eXpressions (MDX) es el lenguaje de consultas multidimensionales más extendido. Sin embargo, los sistemas MOLAP resultan impracticables en bases con un número extenso de datos, ya que requieren de mayores recursos de almacenamiento al no poseer una estructura relacional. Además, como la carga de datos en la estructura multidimensional es mucho más costosa, puede que la información consultada esté desactualizada con respecto al almacén base (Larson, 2017).

En general, los datos son cargados en los servidores OLAP a partir del DW del negocio, de forma que los usuarios pueden utilizar la base de datos OLAP para realizar análisis multidimensionales y también puedan consultar el DW para otras tareas. Al contrario que con el DW, los administradores pueden tener muy poco control sobre la configuración del producto OLAP incluido en la solución (Kimball y cols., 2007).

Una de las desventajas de los modelos multidimensionales es que son complejos de implementar y se necesita invertir tiempo en traducir la estructura relacional en multidimensional. Existen servidores OLAP que permiten generar un modelo **tabular**. A cambio de una pérdida de eficiencia en el procesamiento de las consultas multidimensionales, los modelos tabulares son más sencillos de implementar porque no requieren esta conversión (Larson, 2017). Hay dos tipos de implementación:

- **Copia en memoria (*cached*):** todos los datos son cargados en memoria, por lo que su procesamiento es mucho más rápido que si estuviesen almacenados en disco.

- **Consulta directa (*direct query*):** al contrario que el resto de implementaciones, no se genera ninguna copia de los datos, porque el almacén base es consultado dinámicamente en función de las operaciones que se tienen que realizar. Usualmente, a medida que se van realizando consultas, los datos son almacenados temporalmente en caché para agilizar las consultas repetitivas.

El método de consulta directa es obviamente más lento y consume recursos del almacén base. Sin embargo, a diferencia de la copia en memoria, no hay posibilidad de que la información consultada esté desactualizada porque se obtiene directamente del almacén base.

Algunas de las compañías que desarrollan las tecnologías más utilizadas para el construcción e implantación de sistemas BI son Oracle, Microsoft, SAP, Informatica o Hitachi. Con respecto a los servidores OLAP, MicroStrategy®Intelligence Server[1], SQL Server®Analysis Services[2], Oracle®OLAP [3] y SAS®OLAP Server[4] son ejemplos de tecnologías destinadas a esta función. A su vez, existen múltiples plataformas ABI destinadas al análisis y explotación de los datos, como Power BI®[5], Qlik Sense®[6], MicroStrategy®, Tableau®[7] o Pentaho Platform®[8], entre otros.

[1] `https://www.microstrategy.com/es`,MicroStrategy.
[2] `https://docs.microsoft.com/es-es/analysis-services/analysis-services-overview`, Microsoft.
[3] `https://www.oracle.com/database/technologies/olap.html`, Oracle.
[4] `https://www.sas.com/es_es/software/olap.html`, SAS.
[5] `https://powerbi.microsoft.com`, Microsoft.
[6] `https://www.qlik.com/products/qlik-sense`, Qlik.
[7] `https://www.tableau.com`, Tableau.
[8] `https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho-platform.html`, Hitachi.

1.2. Resumen de la auditoría

Para poner en contexto el tema del proyecto, a continuación se realizará un resumen de la auditoría, empezando por la descripción de la arquitectura del sistema, los problemas encontrados y la estrategia establecida para solucionarlos.

1.2.1. Arquitectura del sistema

El sistema BI que fue objeto de análisis gestionaba y administraba toda la información de la compañía de forma centralizada, se hospedaba en un servidor de bases de datos relacional y constaba de una compleja estructura en dos capas diferenciadas. En la Figura 5.1 se puede ver un esquema de esta estructura. Explicaremos cada componente del sistema empezando por las fuentes de datos hasta finalizar con las aplicaciones de explotación de datos.

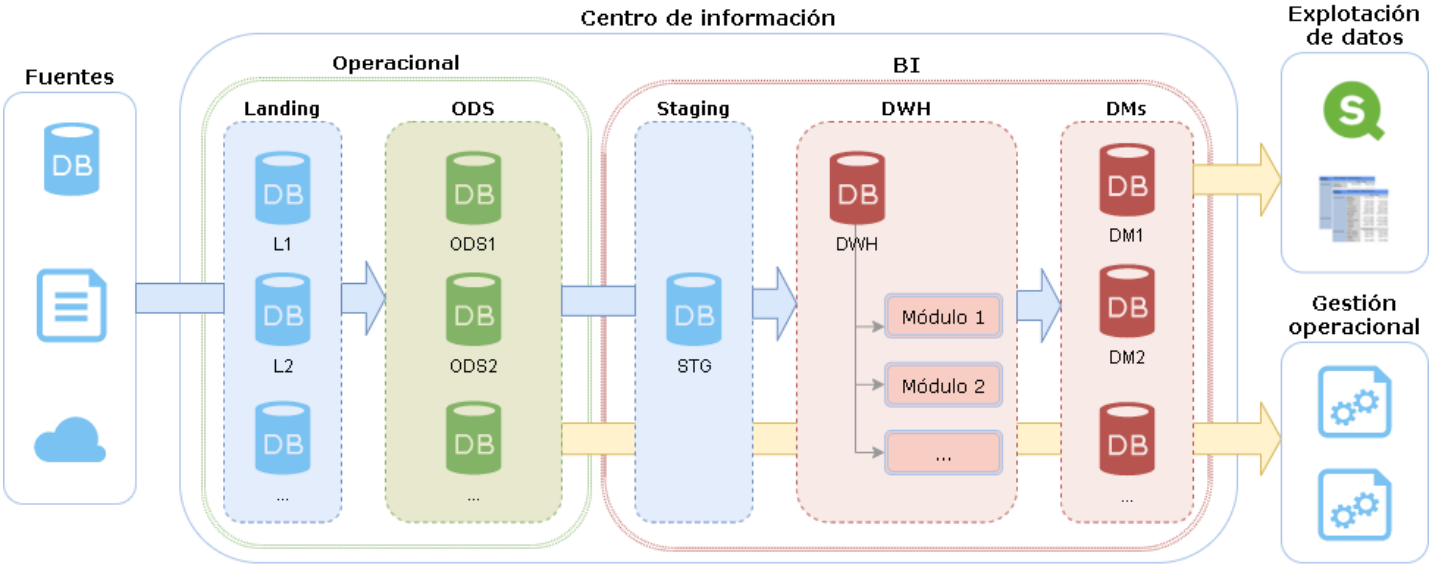

Figura 1.4.: Arquitectura del sistema informacional

Las fuentes de datos desde las que se recogía toda la información del sistema eran de dos tipos:

- **Internas:** datos generados por la propia compañía. Los datos más importantes procedían de bases de datos OLTP, utilizadas para hacer una gestión local de la información generada internamente en cada establecimiento de la empresa. Los datos de cada uno de los establecimientos eran replicados en el servidor de bases de datos central.
- **Externas:** datos públicos generalmente no estructurados e información generada por terceros.

La información pasaba desde las fuentes a una primera capa de datos por medio de procesos ETL. Esta capa tenía un esquema más acorde con la metodología OLTP. Estaba compuesta en

primer lugar por una **zona de landing** que contenía bases de datos transaccionales para el almacenamiento intermedio de los datos de las fuentes de forma temporal. Los procesos ETL se apoyaban en la zona de landing para realizar la carga entre las fuentes y el **almacén de datos operacional (ODS)**.

El ODS contaba con varias bases de datos que mantenían históricos de datos de forma temporal. Este almacén está presente en muchos sistemas de información y en este caso tenía una doble función: servía como almacén temporal entre las fuentes de datos y el DW y además era consultado por los técnicos de la compañía para realizar una gestión operacional de los datos de forma centralizada. Antes de la ingesta del ODS, los datos pasaban en el ETL por un proceso de validación técnica, para verificar identificadores únicos, integridad referencial, eliminar redundancias y limpiar los datos.

A continuación, los datos de la capa operacional se volcaban sobre la siguiente capa. Esta segunda capa contaba con una estructura acorde con los sistemas BI explicados en la Sección 1.1. Estaba compuesta de un almacén central, el **DW corporativo**, que se extendía a un serie de **DM de cada área de negocio de la compañía**. El DW corporativo contenía toda la información manejada por la compañía, mientras que los DM almacenaban los datos relevantes para cada aspecto del negocio.

Los datos pasaban sobre un **área de staging**, que contenía datos del ODS y tablas auxiliares de forma temporal. Servía de apoyo de los procesos ETL para el volcado de información entre el ODS y el DW corporativo. Los datos debían pasar por un proceso de transformación en los ETL para poder cargarse en la estructura dimensional del DW.

El DW corporativo contaba con un esquema de estrella. Estaba compuesto de varios módulos y cada módulo presentaba un modelo dimensional diferente, con tablas de dimensiones y hechos relevantes para un aspecto del negocio en específico. También contenían dimensiones conformadas, que son dimensiones con estructura equivalente en todos los módulos (Kimball y Ross, 2013), para las dimensiones del tiempo y geografía, por ejemplo.

Los datos del DW no eran explotados directamente, sino que se utilizaban para ello los DM de cada área de negocio de la compañía. Los DM, que eran cargados a partir de la información contenida en el DW, poseían a su vez una estructura de datos con un esquema de estrella. Los DM contenían tanto los hechos transaccionales del DW como resumidos o agregados con cálculos de nuevas métricas.

Finalmente, las **aplicaciones BI** explotaban los datos de los DM, cuyo acceso estaba controlado a partir de permisos definidos en las propias aplicaciones. Entre estas tecnologías la más utilizada era QlikSense, la cual estaba alojada en otro servidor y administraba los cuadros de mando y los informes con la información actualizada para las áreas de negocio. También se generaban y enviaban de forma diaria informes realizados con Reporting Services, una herramienta de Microsoft para el reporting dinámico.

La frecuencia con la que se realiza la carga de datos en un sistema BI desde las fuentes depende de los requisitos del negocio en concreto. Las fuentes de datos, ya sean internas o externas, suelen almacenar las transacciones a tiempo real, pero un sistema BI no suele requerir de actualizaciones tan frecuentes de los datos. Esto se denomina **latencia** de la información (Larson, 2017). Típicamente, como en el caso del sistema analizado, las cargas de los datos en las distintas capas se realiza diariamente y progresivamente desde la primera hasta la última capa, por lo que la latencia requerida es diaria.

Estas cargas, que eran realizadas por medio de procesos ETL, eran **incrementales**. Una carga incremental sólo propaga la información nueva por las distintas capas del sistema, de forma que en el DW eran cargados los hechos o dimensiones nuevos o modificados, disminuyendo así el tiempo de procesamiento de cada volcado (Rahman, 2012). Las zonas de landing y staging no mantenían históricos de los datos, al contrario que el ODS, DW y los DM.

La forma de almacenamiento de datos históricos en el DW se realizaba mediante el uso de dimensiones lentamente cambiantes o ***slowly changing dimensions*** **(SCD) del tipo 2**. Esto significa que los históricos se mantenían en las dimensiones mediante la adición de un campo que indicaba si el registro era actual o no. En cuanto a los DM, mantenían históricos en hechos con fotos periódicas al igual que instantáneas de tablas de hechos en momentos puntuales del tiempo. Las instantáneas se creaban y cargaban de forma manual, según las necesidades analíticas en el reporting. El DW también contenía fotos de otras tablas de hechos.

1.2.2. Problemas detectados

La auditoría estaba destinada a detectar las carencias de la capa dimensional, es decir, el DW y los DM. Los problemas detectados en el sistema que tienen relevancia para el proyecto son los siguientes:

- **Sobran tablas de hechos en el DW:** se encontraron tablas de hechos en el DW que simplemente eran instantáneas de otras tablas de hechos y dimensiones. Esto es innecesario, porque estos datos no estaban siendo explotados directamente del DW. Las fotos deberían estar sólo en los DM.
- **Modelo estrella del DW:** no tiene sentido mantener un modelo de estrella con dimensiones desnormalizadas en el DW, porque los datos que mantiene no son explotados directamente, y por tanto no se aprovechan las ventajas en el rendimiento de las consultas cuando se realizan sobre un modelo más desnormalizado. En cambio, hacen aumentar el tiempo de carga y el espacio necesario para su almacenamiento.
- **Procesos ETL de carga de los DM:** los procesos ETL que cargaban los DM no habían sido correctamente implementados. Tenían demasiada complejidad y existían tablas de algunos DM que se cargaban directamente de los datos de la zona de staging sin pasar

por el DW. Todo ello provocaba inconsistencias de datos entre el DW y los DM y un bajo rendimiento de la carga.

- **Falta de automatización en los DM:** en los DM se hacían varias operaciones de administración manualmente, como la adición o eliminación de instantáneas de otras tablas, históricos y cambios en el modelo para incluir nuevas métricas. Estas tareas repetitivas podrían estar automatizadas.

1.2.3. Estrategia

Para paliar los problemas encontrados en la plataforma, se definió una estrategia siguiendo la experiencia de otros proyectos de implantaciones de sistemas BI realizadas por la empresa auditora. Algunas de las acciones determinadas en el alcance del proyecto fueron las siguientes:

- **Definición de un modelo físico en copo de nieve para el DW:** este modelo contendría las dimensiones en un nivel más alto de normalización y las tablas de hechos necesarias. No se introducirían en el modelo tablas que contienen fotos.
- **Gestión de los DM:** se construirían los modelos de datos en los DM específicos para cada área de negocio concreto con las dimensiones desnormalizadas del DW necesarias, tablas con hechos agregados en base a métricas e instantáneas de tablas. Tanto la definición de dimensiones o hechos como la carga de los DM con respecto al DW deberían estar en gran medida automatizadas.

  Igualmente, se debería proporcionar flexibilidad para modificaciones en el modelo y tener en cuenta requisitos de latencia diferentes. Esto último se justifica porque al contrario que el DW, cuyo objetivo es almacenar de manera actualizada todos los eventos que ocurren en el negocio en un esquema comprensible, las necesidades de latencia de datos pueden ser diversas en los DM.

En base al segundo punto de esta estrategia, en el alcance funcional del proyecto de reingeniería se propuso sustituir los procesos ETL que realizaban el volcado de datos en los DM a partir del DW por otra solución que permitiese automatizar la generación de los modelos y los procesos de carga de los DM en el servidor de bases de datos.

Mediante la definición de una planificación en base a unas reglas o parametrizaciones se podría desarrollar una lógica que automatizara estos procesos. Esta planificación sería definida por el equipo de IT de la compañía a cargo de la administración del sistema. Además, la solución implantada sería lo suficientemente flexible para adaptarse a cambios en las exigencias analíticas del negocio.

De esta manera, se podría agilizar la implementación de los DM y a la vez optimizar los procesos de obtención de datos desde las aplicaciones BI, incorporando posibles modificaciones debido a las cambiantes necesidades de análisis.

1.3. Motivación

La agilización de la obtención de datos en las aplicaciones BI para la presentación de la información es uno de los problemas recurrentes en los sistemas de información. En la Sección 1.2 se ha explicado un caso real de una plataforma BI cuyos defectos tenían impacto en el acceso de los usuarios a la información del negocio.

La motivación de este proyecto es formalizar la propuesta de una herramienta que pueda ser de utilidad para mitigar este problema en sistemas de información con carencias parecidas al caso descrito. Esta herramienta, si es suficientemente genérica, puede ser de aplicación directa en multitud de escenarios.

1.4. Objetivos

El objetivo de este trabajo es el desarrollo de una herramienta con la función de generar y cargar modelos de datos relacionales dentro del servidor de bases de datos de un sistema BI. Los modelos implementados servirán como orígenes de datos para las aplicaciones que explotan la información del negocio. La herramienta deberá poder adaptar modificaciones en los modelos debido a posibles cambios en las necesidades de análisis, así como flexibilizar la latencia de la información contenida en ellos.

Actualmente existen multitud de arquitecturas y variaciones para la construcción de sistemas de información, además del continuo surgimiento de nuevas tecnologías. Teniendo en cuenta esta cuestión, un segundo propósito del proyecto es facilitar el desarrollo de la solución en otros sistemas independientemente de su arquitectura o la tecnología en la que estén implantadas.

1.5. Estructura de la memoria

La estructura del presente documento es la que sigue. En el siguiente capítulo, se hará una propuesta de la herramienta, planteando la funcionalidad que debe incorporar en el contexto de un caso de negocio concreto. En el Capítulo 3, se expondrá el plan de proyecto, incluyendo las metodologías seguidas, la planificación inicial y las herramientas que se utilizarán.

Posteriormente se describirá el desarrollo de la herramienta, empezando por los requisitos y análisis (Capítulo 4), continuando con el diseño (Capítulo 5) y la implementación y las pruebas (Capítulo 6). También se incluirá, en el Capítulo 7, el despliegue de la herramienta.

Finalmente, en el último capítulo se realizará un comentario sobre el trabajo realizado y posibles vías de mejora que se pueden llevar a cabo.

2. Planteamiento de la solución

En este capítulo se planteará una propuesta para la solución al problema expuesto en el capítulo anterior. Primeramente, se hablará brevemente del trabajo relacionado con el tema tratado. Posteriormente, se describirá un ejemplo de negocio que servirá como base para inferir la funcionalidad que debe incorporar la herramienta, la cual se explica en el último apartado.

2.1. Trabajo relacionado

La *agilidad* de un sistema BI se conoce como su capacidad para adaptarse a modificaciones no previstas de los requisitos que afectan a su funcionalidad o contenido debido al entorno altamente cambiante y dinámico en el que se sitúan (Zimmer y cols., 2012). Varios estudios se han dedicado al desarrollo del llamado **BI ágil (*agile BI*)** en base a esta idea. Knabke y Olbrich (2013) profundizan en el concepto de BI ágil agrupando y comparando muchas de las definiciones hechas sobre el tema.

En Krawatzeck y Dinter (2015) se realizó una extensa investigación sobre las estrategias y soluciones propuestas para el BI ágil, clasificándolas en cuatro categorías: principios, métodos, técnicas y tecnologías. Algunas de las técnicas analizadas tienen el objetivo de automatizar la construcción de modelos físicos a partir de abstracciones de conocimiento del negocio, como puede ser modelos conceptuales o lógicos, ontologías, reglas y requisitos.

Un ejemplo de ello se describe en Krneta y cols. (2014). Este estudio propone un algoritmo que, a partir del modelo conceptual del negocio, genera un modelo físico para el DW del tipo *Data Vault* (DV), metodología que sigue un enfoque distinto al tradicional modelado dimensional. Además, enumera una serie de tecnologías que permiten automatizar la obtención de las estructuras de DW, DM u operaciones ETL.

Muchas de estas soluciones siguen la metodología ***Model Driven Architecture* (MDA)**, como Muñoz y cols. (2009), que trabajaron en la generación automática de ETL para casos particulares mediante transformaciones QVT (*Query/View /Transform*) de su diseño conceptual general. A su vez, Zhang y cols. (2006) presentan un algoritmo que permite obtener ETL incrementales de manera automática. Igualmente, muchas de las tecnologías BI mencionadas al final de la Sección

1.1 también poseen alguna herramienta para automatizar en menor o mayor grado la construcción de los componentes de un sistema BI.

La propuesta que se describe en este capítulo también sigue un enfoque en línea con el BI ágil, ya que la herramienta se encargará de generar y administrar modelos de datos explotados por las aplicaciones de análisis partiendo de una parametrización definida por el administrador de la plataforma.

Frente a otras soluciones comentadas, la herramienta permitirá automatizar la implementación física de estos modelos y los procesos de carga de datos, agilizando el desarrollo del sistema implantado. Además, será flexible para adaptarse a cambios que puedan surgir en las necesidades de análisis.

2.2. Ejemplo de negocio

Partimos de un sencillo negocio de una empresa de bicicletas, que almacena información sobre sus ventas, productos y clientes, entre otros. Además del almacenamiento de los datos generados, se quiere explotar parte de la información del negocio para ayudar a la toma de decisiones. Más concretamente, se desea realizar un seguimiento de las ventas y el estado del stock por cada producto y tienda del negocio en el transcurso del año actual.

Diariamente, los clientes registrados realizan pedidos a un empleado en una tienda concreta. Cada pedido puede contener varios ítems, y cada ítem es una cantidad de un producto determinado. Los productos son bicicletas, cada cual posee una marca y categoría. Las tiendas poseen una cierta cantidad de todos los productos, almacenados en stock.

El número de pedidos diarios registrados es del orden de las decenas, mientras que el número de items en cada pedido es del orden de unidades. Igualmente, de forma diaria se guarda el estado del stock de cada tienda según los pedidos que han tenido lugar. Por otro lado, los productos comienzan a venderse a partir de su salida a la venta. La mayor parte de ellos se comienzan a vender al principio de año, pero otros lo harán en otras fechas más recientes. Por último, los clientes, empleados, tiendas y productos permanecen inalteradas, siendo siempre los mismos durante todo el año.

La información se almacenará siguiendo el modelo de la figura 2.1[1]. Este modelo posee varias entidades en una estructura dimensional, con características tanto de modelos en estrella como en copo de nieve. Las entidades se pueden clasificar en:

- **Dimensiones:** clientes, empleados, tiendas, productos, categorías de producto y marcas de producto.

[1]Obtenido a partir de modificaciones sobre el modelo propuesto en SQL Server Tutorial `https://www.sqlservertutorial.net/sql-server-sample-database/`, última visita: febrero de 2020.

- **Hechos:** pedidos, items por pedido y stock de productos.

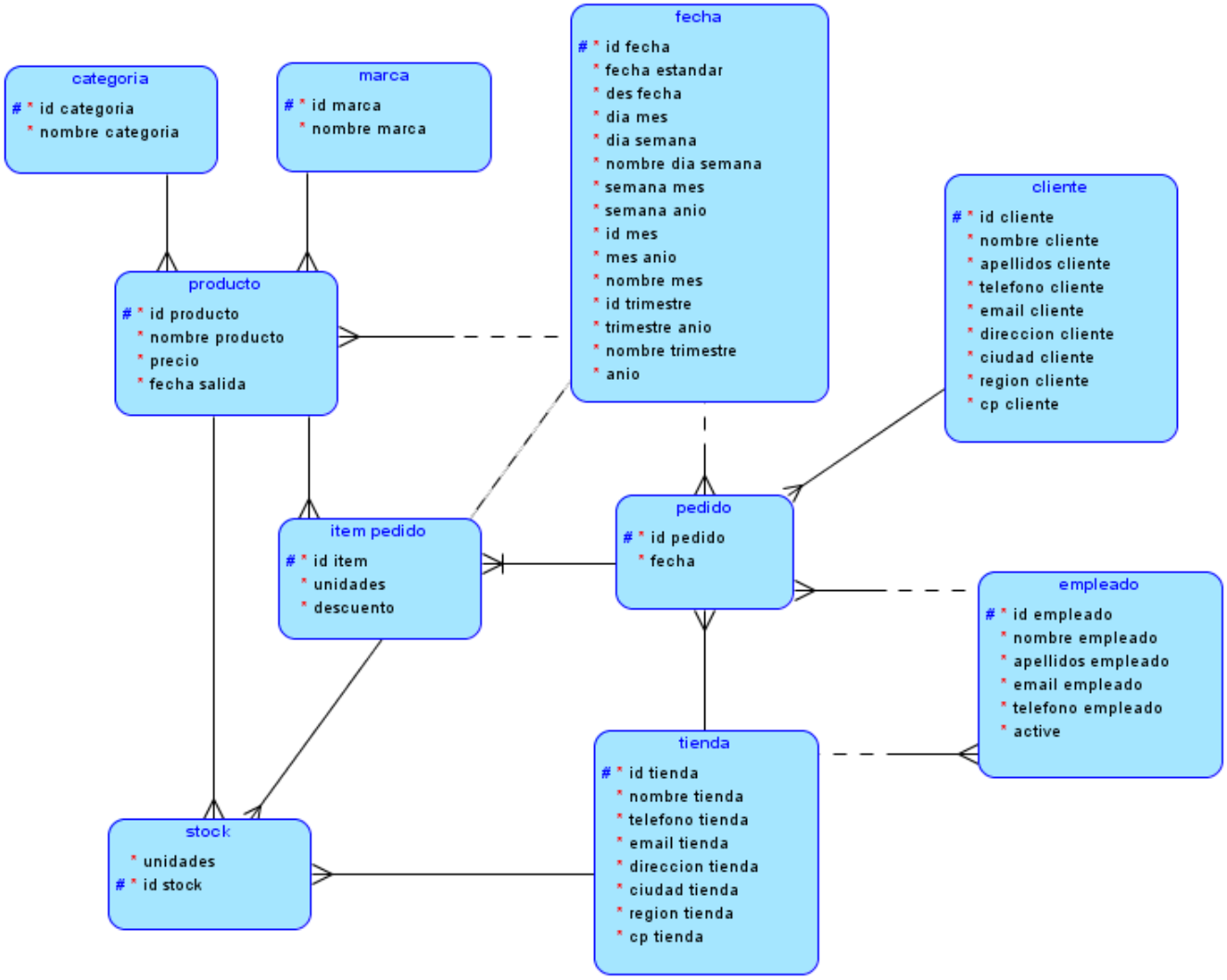

Figura 2.1.: Modelo lógico del almacén de información del negocio

Este modelo se implementará formando un almacén de la información generada en el negocio. Para facilitar la explotación de la información se quiere crear, a partir de los datos de este almacén base, otro almacén que aloje la información de forma específica para facilitar las operaciones que se realizarán desde la aplicación BI, llamado almacén de explotación. De esta forma, la aplicación podrá obtener la información contenida en el almacén de explotación, que al contrario que el base estará preparado para ello. Las entidades que deben incluirse son, en un principio:

- **Dimensión desnormalizada de los productos:** los productos aparecen en el almacén base normalizados en tres dimensiones: producto, categoría y marca. Es conveniente incluir en el almacén una dimensión que contenga toda esta información desnormalizada para que pueda ser consultada.

- **Hechos con fotos periódicas de las ventas por producto y tienda:** en el almacén base se registran todos los pedidos ocurridos diariamente. Como en el análisis no interesan los pedidos, sino las ventas, habrá que crear una entidad que almacene las ventas a nivel de detalle de producto y tienda agregando los datos de los pedidos realizados. Hay varias métricas relevantes que deben incluirse en esta entidad, como los ingresos diarios o las unidades solicitadas.
- **Capturas del stock de productos por tienda:** los datos del stock son almacenados diariamente. Sin embargo, además de que mantener toda esta información es costosa en espacio de almacenamiento, sólo se necesitaría saber el estado del stock en fechas estratégicas, como por ejemplo al final de cada mes. Para ello, se necesita una entidad que almacene instantáneas de los datos del stock en las fechas relevantes de forma que la información del stock en el almacén base pueda ser borrada periódicamente.
- **Capturas de las dimensiones pertinentes:** la dimensión de tiendas debe replicarse en el almacén de explotación para que pueda ser consultada en el análisis.

Por otra parte, hay que tener en cuenta que el número de entidades debe poder ser ampliable en función de nuevas necesidades de análisis, ya que se espera que en un futuro puedan ser explotados con mayor profundidad los datos generados en el negocio, y no sólo las ventas y el stock. La solución adoptada, por tanto, debe contemplar esta posibilidad.

Además, sería muy conveniente que la solución implantada permitiera diferentes tipos de latencia de los datos contenidos en el almacén de explotación, así como la configuración del almacenamiento de históricos. En el caso del análisis de ventas, se necesita que se mantenga la información pertinente a un nivel de detalle diario durante todo el año, pero para el stock no se requiere tanto detalle.

2.3. Propuesta

La herramienta tendrá como objetivo la implementación automática de uno o varios modelos de datos relacionales y sus procesos de carga para facilitar orígenes de datos con los que explotar la información del negocio en las plataformas BI. A partir del caso de negocio descrito, podemos resaltar los siguientes aspectos fundamentales:

- Cada modelo generado tiene que ser **configurable y ampliable** según las necesidades cambiantes de explotación de datos.
- En la carga de nuevos datos se deberá tener en cuenta que las entidades que componen cada modelo implementado pueden tener **diferentes necesidades de latencia** de la información que contienen.
- Se debe poder flexibilizar la cantidad de **datos históricos** a almacenar para cada entidad.

Toda esta configuración vendrá dada por parametrizaciones definidas por el administrador del servidor de bases de datos del sistema, en base a las exigencias analíticas. En resumidas cuentas, la solución será una herramienta que interactúa con el servidor de bases de datos como una ETL parametrizable con la función de generar y cargar modelos de datos. Como ya se comentaba en la Sección 1.1, de forma general los modelos de datos podrían albergar:

- **Dimensiones desnormalizadas:** para minimizar el tiempo de ejecución de las consultas lanzadas contra base de datos y maximizar la información disponible de la dimensión para el análisis.
- **Nuevas tablas de hechos:** calculadas a partir de otras tablas de hechos para resumir los datos, agregarlos en base a alguna métrica o modificar KPIs.
- **Instantáneas o fotos de las tablas:** pueden ser puntuales o periódicas y deberían poderse borrar si dejan de ser útiles en las consultas.

En el ejemplo propuesto en el anterior apartado, la herramienta se encargará de implementar el almacén de explotación, a partir del cual se obtendrán los datos para su visualización en la aplicación BI.

En primer lugar, la herramienta debe permitir que el administrador del sistema defina las entidades que compondrán los modelos, ya sean dimensiones, hechos o fotos. Normalmente, como en el caso del sistema analizado, los procesos ETL se encargarían de obtener los datos de una o varias bases de datos de partida con un modelo determinado y transformarlos para ajustarlos a las entidades del modelo de la base de datos donde son volcados.

Los procesos de obtención de los datos en las capas superiores de un sistema informacional se pueden traducir de forma directa en consultas realizadas sobre las bases de datos de partida. Los datos que contiene cada entidad, por tanto, se deberían poder obtener explícitamente a partir de una **consulta**. Esta consulta dará como resultado las dimensiones o hechos que contendrá la entidad.

Cada modelo de datos puede implementarse en una o varias bases de datos dentro del sistema. Por ello, también se debe proporcionar la **localización física** de la entidad en el servidor de bases de datos. La herramienta se encargará de construir para cada entidad definida la estructura de datos correspondiente (tabla, vista, o cualquier otra estructura elegida en la implementación) en la localización especificada.

La carga del sistema BI descrito se realiza de forma diaria. La fecha en la que se realiza esta carga se toma como referencia para transmitir desde la fuentes al sistema los datos nuevos o modificados. La herramienta, dentro de esta gran operación, se encargará de realizar la carga de los modelos generados en base a esta fecha. Los datos de las entidades que componen cada modelo se deberían poder actualizar con una **periodicidad** distinta y específica para cada una según las necesidades de latencia de la información que contienen.

Llamaremos **refresco** a la actualización de los datos de una entidad durante la carga. Suponiendo que la carga se realice de forma diaria, las periodicidades permitidas podrían ser, por ejemplo:

- Diaria: refresco con latencia diaria.
- Semanal: latencia semanal. Se deberá especificar el día de la semana en el que tendrá lugar el refresco.
- Mensual: latencia mensual. Se tendrá que proporcionar el día del mes del refresco.
- Ocasional: el refresco se realiza momentáneamente en una fecha especificada y no sigue ningún patrón temporal.

Además de personalizar la periodicidad de los refrescos de cada entidad, también se debería poder elegir la **frecuencia** de este refresco. Es decir, para una periodicidad semanal el refresco podría ser cada dos semanas en lugar de todas las semanas. Con ello se conseguiría flexibilizar mucho más la actualización de los datos de cada entidad.

Estos parámetros se podrán definir en **planificaciones de refresco**. Cuando se crea una nueva entidad, se podrá especificar más de una planificación de refresco para que establezcan el calendario de refrescos completo de la entidad. De esta forma, los datos de una entidad pueden ser actualizados con más de una periodicidad, lo que aporta una mayor capacidad de configuración del refresco. Durante una carga, se comprobará si cada entidad debe ser refrescada según el calendario establecido por las planificaciones a las que está asociado para la fecha en la que tiene lugar la carga.

Cuando en la carga hay una entidad a refrescar, los datos del refresco anterior podrían sustituirse por los nuevos o mantenerse almacenados en la entidad como históricos. Los datos históricos son necesarios cuando se quiere visualizar la tendencia o la evolución de una métrica a lo largo del tiempo. La herramienta debe permitir parametrizar el número de históricos a almacenar en cada entidad, así como establecer un límite para que el espacio necesario para el almacenamiento de los datos no crezca en gran medida.

Como una entidad puede estar asociada a varias planificaciones de refresco, si se define este límite para la propia entidad no se podría saber el número de históricos generados por cada planificación. Por tanto, al definir una nueva planificación se especificará el número de **refrescos máximos** cuyos datos deben mantenerse como históricos para el tipo de refresco de esa planificación concreta. La herramienta se encargará de eliminar los datos históricos en caso de que se supere el máximo cada vez que se vuelva a refrescar una entidad durante una carga.

Se debería permitir incluir nuevas entidades y planificaciones en cualquier momento, en base a las necesidades de explotación de información por las aplicaciones BI. Igualmente, puede darse el caso de que existan entidades definidas en algún modelo que pasen a ser inutilizadas. Para lidiar con estas situaciones, se tendrá que poder **desactivar el refresco** de una entidad para que la

herramienta no continúe actualizando sus datos o directamente **eliminar la entidad**, incluyendo su estructura de datos, así como **eliminar planificaciones** que no se usan. De la misma manera, debe permitir el **borrado de refrescos**, en el caso de que los datos que contienen dejen de ser necesarios o sean erróneos.

Finalmente, sería conveniente que la herramienta mantenga un registro de las acciones realizadas. Las tareas de flujo de datos son operaciones delicadas porque pueden ocasionar pérdida de información. En caso de producirse una interrupción en una carga de datos es primordial descubrir el origen de error y qué elementos se han visto afectados.

3. Plan de proyecto

En este capítulo se expone el plan de proyecto. En primer lugar, se hablará de la metodología seguida para el desarrollo del proyecto. Seguidamente, se expondrán las restricciones y la gestión de los riesgos. A continuación, se definirá la planificación inicial del proyecto y la estimación de los costes, y por último se describirán las herramientas que se utilizarán en el desarrollo.

3.1. Metodología

La metodología utilizada en el proyecto es el Proceso Unificado, pero siguiendo el enfoque de la Arquitectura Dirigida por Modelos.

3.1.1. Proceso Unificado

El **Proceso Unificado (*Unified Proccess* o UP)** es una de las metodologías de desarrollo de software más extendidas. Documentado por Kruchten (2003), se distingue por seguir una serie de 'buenas prácticas':

- Iterativo e incremental, enfocado en minimizar los riesgos.
- Dirigido por casos de uso.
- Centrado en la arquitectura.
- Dar importancia de los modelos.
- Comprobar continuamente la calidad del software.
- Controlar cambios en el software.

El ciclo de vida de un proyecto según UP se puede ver en la Figura 3.1, el cual está compuesto por dos dimensiones. Verticalmente se representan las principales disciplinas del desarrollo, mientras que las fases del proyecto con respecto del tiempo están en horizontal, las cuales son:

- **Inicio:** se establece el contexto de negocio, los principales requisitos, la planificación y una primera evaluación de los riesgos.

- **Elaboración:** se desarrolla la mayor parte del análisis y se construye una arquitectura del sistema que pueda ser validada para minimizar los riesgos.
- **Construcción:** la fase más larga y con mayor número de iteraciones en la que se construye el sistema.
- **Transición:** se dispone el producto desarrollado al usuario y se valida el alcance del proyecto.

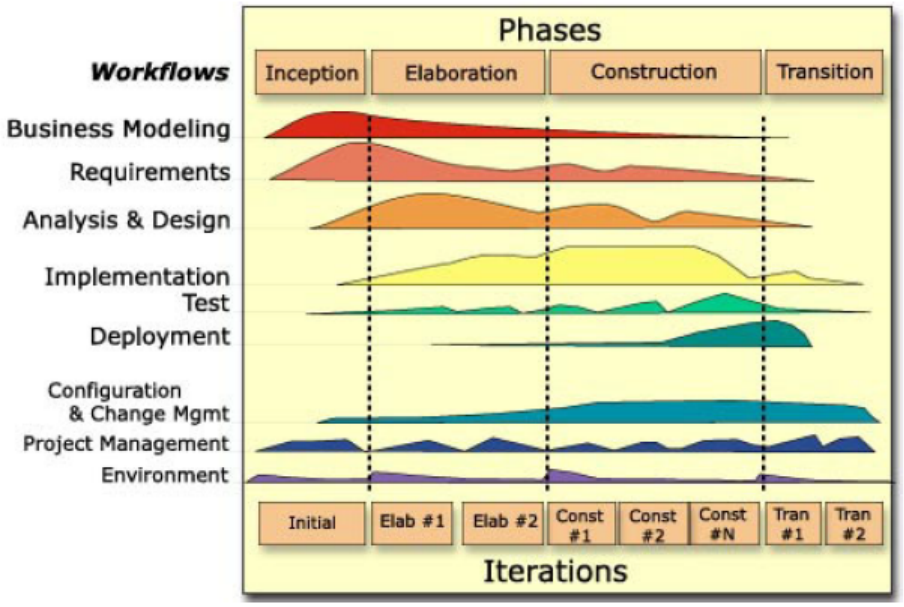

Figura 3.1.: Ciclo de vida del desarrollo de software UP (Kruchten, 2003)

3.1.2. Arquitectura Dirigida por Modelos

La **Arquitectura Dirigida por Modelos (*Model-Driven Arquitecture* o MDA®**[1]) es un enfoque del desarrollo de software propuesto por ***Object Management Group* (OMG)**. Esta metodología se centra en sacar el máximo valor a los modelos que se van generando durante las diferentes etapas del desarrollo de software. En este sentido, los modelos no sólo son parte de la documentación de un proyecto, sino artefactos aprovechables para obtener otros entregables del mismo (Siegel, 2014).

Siguiendo este enfoque en los proyectos, se podrían agilizar las etapas del desarrollo, así como mejorar las propiedades de software generado como su mantenimiento, extensibilidad o portabilidad. Para lograr este fin, son fundamentales dos aspectos:

- La estructura, semántica y notación de los modelos generados deben seguir algún estándar de lenguaje de modelado que pueda ser entendido tanto por desarrolladores y como por las tecnologías utilizadas.

[1]Model Driven Arquitecture `https://www.omg.org/mda/`, OMG. Última visita: febrero de 2020.

- Se pueden aplicar un conjunto de transformaciones a los modelos para poder derivar otros artefactos de forma parcialmente o completamente automatizada.

MDA define varios tipos de modelos de mayor a menor nivel de abstracción de una arquitectura, de los que se destacan:

- **Modelo Independiente de la Plataforma (PIM):** modelo que define el sistema cumpliendo con los requisitos pero independiente de la tecnología o plataforma en la que se ejecuta. Dos de los estándares más utilizados son Unified Modeling Language (UML®[2]), el lenguaje de modelado más extendido, o XML Metadata Interchange (XMI®[3]), para facilitar el intercambio de metadatos de modelos entre herramientas que utilizan diferentes lenguajes de modelado.
- **Modelo Específico de la Plataforma (PSM):** modelo del sistema característico de la plataforma en la que se ejecuta mediante un lenguaje propio de la tecnología utilizada, por ejemplo, SQL para los sistemas de información.

Según el MDA, mediante la utilización estándares para el modelado y herramientas de transformación de modelos, se podrían obtener artefactos con cada vez menor nivel de abstracción: partiendo de un PIM, se pueden generar varios PSM para diferentes plataformas, de los cuales se puede derivar parte del código fuente que implementa el sistema en la tecnología específica (Kleppe y cols., 2003). Esto es muy importante para el desarrollo de software en el entorno tan cambiante que envuelve al BI, donde nuevas tecnologías surgen constantemente.

En la Figura 3.2 se puede ver la aplicación de MDA en el marco del desarrollo de software tradicional, donde los modelos obtenidos pasan a ser artefactos del desarrollo.

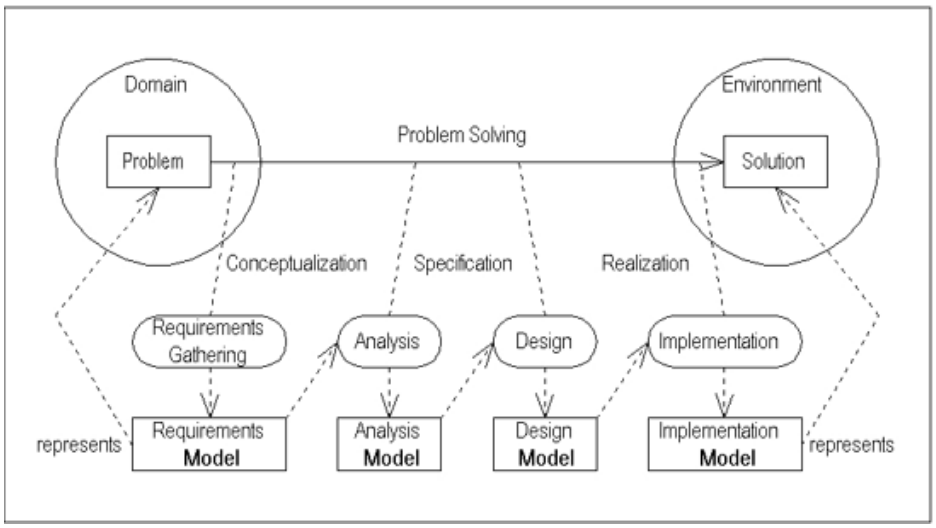

Figura 3.2.: Esquema del desarrollo de software aplicando MDA (Alhir, 2013)

[2]Unified Modeling Language `https://www.omg.org/spec/UML`, OMG.
[3]XML Metadata Interchange `https://www.omg.org/spec/XMI`, OMG.

Las características más importantes del UP (iterativo e incremental, centrado en la arquitectura) hacen que sea un desarrollo de software consistente con el MDA (Brown y Conallen, 2005). MDA incide con mayor relevancia sobre la fase de Elaboración pero no implica modificar ninguna de las actividades propias de UP. Al contrario, permite la automatización de algunas de ellas, gracias al proceso de transformación de PIM a PSM y de PSM a código. Además, proporciona una mayor diferenciación de las tareas destinadas a cada rol del proyecto.

3.2. Restricciones

Las restricciones más relevantes que afectan al alcance del proyecto son las siguientes:

- **Tiempo:** el tiempo para completar el trabajo es limitado, hasta el final del curso académico actual.
- **Recursos:** los recursos hardware y software para desarrollar el trabajo son limitados, así como el presupuesto destinado al proyecto.
- **Personal:** el alumno debe realizar todos los roles necesarios para completar el trabajo.
- **Conocimientos:** los conocimientos del alumno con respecto a las tecnologías utilizadas son limitados.

3.3. Gestión de riesgos

A continuación se listan los riesgos identificados y su gestión según el estándar PMBOK®[4].

R0	**Pérdida de datos**
Descripción	Pérdida de parte o todo el desarrollo del proyecto (documentación, entregables, etc)
Categoría	Riesgo de proyecto
Probabilidad	Baja
Impacto	Alto
Estrategia	Evitar el riesgo
Plan de acción	No aplica
Plan de contingencia	Utilizar GitLab para el control de versiones y hacer copias de seguridad con frecuencia

Tabla 3.1.: Descripción del riesgo R0

[4]PMBOK Guide and Standards `https://www.pmi.org/pmbok-guide-standards`, Project Management Institute. Última visita: marzo de 2020.

R1	**Falta de conocimiento de la tecnología**
Descripción	Tiempo dedicado a obtener los conocimientos de las tecnologías utilizadas
Categoría	Riesgo de producto
Probabilidad	Media
Impacto	Bajo
Estrategia	Aceptar el riesgo
Plan de acción	Buscar documentación y ejemplos
Plan de contingencia	No aplica

Tabla 3.2.: Descripción del riesgo R1

R2	**Falta de disponibilidad del alumno**
Descripción	Incapacidad de continuar con el proyecto debido a una enfermedad o por trabajo
Categoría	Riesgo de proyecto
Probabilidad	Baja
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Realizar nuevamente la planificación
Plan de contingencia	Monitorizar la disponibilidad del alumno

Tabla 3.3.: Descripción del riesgo R2

R3	**Falta de disponibilidad de los recursos hardware**
Descripción	Incapacidad de poder continuar con el desarrollo por falta de disponibilidad del equipo
Categoría	Riesgo de proyecto
Probabilidad	Alta
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Continuar con el desarrollo en otro equipo
Plan de contingencia	Disponer de otro equipo preparado para poder continuar con el proyecto

Tabla 3.4.: Descripción del riesgo R3

R4	**Falta de disponibilidad de los recursos software**
Descripción	Incapacidad de poder continuar con el desarrollo por falta de disponibilidad de las herramientas o tecnologías utilizadas
Categoría	Riesgo de proyecto
Probabilidad	Baja
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Encontrar el problema de la falta de disponibilidad y solucionarlo
Plan de contingencia	Disponer de las tecnologías actualizadas y configuradas

Tabla 3.5.: Descripción del riesgo R4

R5	**Retraso en la planificación**
Descripción	Mala estimación del tiempo dedicado a las tareas
Categoría	Riesgo de proceso
Probabilidad	Media
Impacto	Medio
Estrategia	Mitigar el riesgo
Plan de acción	Replanificar el proyecto
Plan de contingencia	Revisar la planificación frecuentemente

Tabla 3.6.: Descripción del riesgo R5

R6	**Retraso al disponer un entregable**
Descripción	Mala estimación de la fecha de obtención de los entregables
Categoría	Riesgo de proceso
Probabilidad	Media
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Replanificar el proyecto
Plan de contingencia	Revisar la planificación y el estado de los entregables frecuentemente

Tabla 3.7.: Descripción del riesgo R6

R7	**Cambios en los requisitos**
Descripción	Modificación de los requisitos para satisfacer el alcance del proyecto
Categoría	Riesgo de producto
Probabilidad	Baja
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Evaluación de los requisitos y replantear los cambios en la funcionalidad que supone
Plan de contingencia	Someter a una evaluación frecuente los requisitos para verificar el cumplimiento del alcance

Tabla 3.8.: Descripción del riesgo R7

R8	**Arquitectura deficiente**
Descripción	La arquitectura del sistema no cumple con los requisitos especificados
Categoría	Riesgo de producto
Probabilidad	Baja
Impacto	Alto
Estrategia	Mitigar el riesgo
Plan de acción	Rediseñar la arquitectura del sistema
Plan de contingencia	Validar la arquitectura cuando se disponga de un ejecutable de la misma

Tabla 3.9.: Descripción del riesgo R8

3.4. Planificación inicial

El proyecto se ha dividido en 7 iteraciones, como se puede ver en la Tabla 3.10. Cada semana se dedicarán 20 horas al trabajo, por lo que se hará un total de 300 horas. El parón de 2 semanas entre el 16 de febrero y el 2 de marzo es debido a un par de viajes planificados que impedirán continuar con el desarrollo del proyecto.

En la misma tabla se especifican los entregables a obtener al final de la iteración. Estos entregables se irán refinando en sucesivas iteraciones hasta llegar al producto final.

Iteración	**Fase**	**Fecha inicio**	**Fecha fin**	**Semanas**	**Horas**	**Entregable**
1	Inicio	13/01/2020	26/01/2020	2	40	
2	Elaboración	27/01/2020	16/02/2020	3	60	PIM
3	Elaboración	02/03/2020	15/03/2020	2	40	PSM
4	Construcción	16/03/2020	22/02/2020	1	20	
5	Construcción	23/03/2020	26/04/2020	5	100	Código
6	Construcción	27/04/2020	03/05/2020	1	20	
7	Transición	04/05/2020	10/05/2020	1	20	Producto final Documentación
Total		13/01/2020	10/05/2020	15	300	

Tabla 3.10.: Planificación inicial

La lista de tareas de cada iteración es la siguiente:

- **Iteración 1:**
  - Establecer la metodología de trabajo.
  - Gestión de riesgos.
  - Estimación del coste.
  - Planificación inicial.
  - Instalación de software.
  - Obtener requisitos principales.
- **Iteración 2:**
  - Recoger requisitos.
  - Desarrollar el análisis.
  - Obtener el PIM.
  - Diseñar arquitectura del sistema.

- **Iteración 3:**
  - Recoger requisitos.
  - Actualizar análisis.
  - Desarrollar el diseño.
  - Obtener el PSM.
- **Iteración 4:**
  - Implementar el almacén base.
- **Iteración 5:**
  - Implementar la herramienta.
  - Pruebas.
  - Actualizar requisitos.
  - Actualizar análisis.
  - Actualizar diseño.
  - Obtener el código.
- **Iteración 6:**
  - Implementar la plataforma de visualización.
- **Iteración 7:**
  - Despliegue del sistema.

Los entregables se componen de:

- **PIM:** Modelo del dominio y modelo de los casos de uso en UML.
- **PSM:** Modelo relacional de las tablas y procedimientos del motor en SQL.
- **Código:** Código fuente que implementa el motor.
- **Producto final y documentación**.

3.5. Coste del proyecto

Para calcular una estimación del coste del proyecto, se tienen en cuenta cuatro tipos de costes:

- **Costes de hardware**.
- **Costes de software**.
- **Costes de personal**.

- **Costes indirectos**.

Con respecto a los costes de hardware, el trabajo se realizará utilizando el ordenador personal del alumno. Debido a larga vida útil del equipo, está algo deteriorado, por lo que existe una alta probabilidad de que deje de estar disponible temporalmente. En previsión, se dispone de otro equipo preparado para retomar el trabajo. Las especificaciones de los equipos y su coste se pueden ver en la Tabla 3.11.

Hardware	Coste de compra	Vida útil	Tiempo utilizado	Coste total
HP Pavillion 15 Notebook	800 €	6 años	4 meses	45 €
HP EliteBook 840 G3	500 €	1 año	4 meses	167 €

Tabla 3.11.: Costes de hardware

En relación con el software, se han utilizado herramientas gratuitas, con licencia de estudiante o su versión de desarrollador, por lo que el coste es de 0 €.

El sueldo medio de un analista-programador en España es de 28.000 €brutos anuales[5], lo que suponiendo una jornada laboral de 40 horas son unos 14 €por hora aproximadamente. La estimación del coste de personal es, por tanto, de 4.200 €.

Por último, hay que tener en cuenta los costes indirectos derivados de la conexión a Internet (*Movistar* Fibra óptica 300 MB). El coste mensual son 60 €, lo que en 4 meses supone 240 €.

Teniendo en cuenta los cuatro tipos de costes, la estimación de coste total del proyecto es de 4.652 €.

3.6. Desviaciones con respecto a la planificación inicial

Debido a semanas en las que por incompatibilidades para compaginar la ocupación laboral con el proyecto no se han podido completar las 20 horas de trabajo, por un cambio en los requisitos y por retrasos para completar correctamente la documentación del proyecto, la duración del mismo ha sido de 5 semanas más, obteniendo el producto final y la documentación el 14/06/2020.

[5]Salarios para empleos de Analista programador/a en España `https://www.indeed.es/salaries/analista-programador-Salaries`, Indeed. Última visita: marzo de 2020.

3.7. Herramientas utilizadas

3.7.1. Herramientas de modelado

Al seguir una metodología de proyecto centrada en los modelos, se necesitan herramientas potentes de modelado en los diferentes niveles de abstracción del sistema. Éstas son Astah y Data Modeler.

3.7.1.1. Astah

Astah®[6] es un programa de modelado desarrollado por ChangeVision. Aunque centrado en la construcción de modelos en UML, proporciona herramientas para la representación de otros múltiples tipos de diagramas, como por ejemplo mapas mentales o diagramas relacionales. Además, es posible agregar dos *plug-in* para importar y exportar los modelos UML en XMI.

Existen diferentes tipos de productos de Astah, entre ellos Astah Professional, el cual se usará en este proyecto con la licencia de estudiante. Otros son Astah SysML, que permite el modelado de sistemas utilizando SysML, o Astah GSN, para construir diagramas de sistemas industriales co la notación GSN.

3.7.1.2. Data Modeler

Oracle SQL Developer Data Modeler[7] es una herramienta gráfica gratuita desarrollada por Oracle para el modelado de datos. Con Data Modeler se pueden construir modelos relacionales o multidimensionales para diversas plataformas. Además, posee capacidades de ingeniería, lo que permite, por ejemplo, automatizar la generación de código DDL (*Data Definition Language*) (ingeniería directa) para generar las estructuras de datos de un sistema u obtener el modelo lógico (ingeniería inversa) a partir de un modelo físico.

3.7.2. Microsoft BI

Microsoft es una de las empresas líder en el sector de BI gracias al software que ha desarrollado para la construcción de sistemas de información. Dos de las tecnologías más relevantes son SQL Server y Power BI.

[6] `https://astah.net/`, ChangeVision.
[7] `https://www.oracle.com/es/database/technologies/appdev/datamodeler.html`, Oracle.

3.7.2.1. SQL Server

SQL Server®[8] es uno de los RDBMS con mayor popularidad en todo el mundo, compitiendo de cerca con otras tecnologías como Oracle y MySQL, pero muy por encima de los siguientes más utilizados como PostgreSQL o MongoDB[9]. Además, su instalación es posible en sistemas Windows, Linux y contenedores Docker en un gran número de idiomas. En este trabajo se utilizará la edición Developer de SQL Server, que contiene todas las características disponibles de la versión completa para desarrolladores. Hay disponibles tres ediciones utilizables en entornos de producción: *Enterprise*, *Standard* y *Express*.

SQL Server soporta el lenguaje de desarrollo **Transact-SQL (T-SQL)**, una extensión de los estándares ISO y ANSI de SQL. Este lenguaje posee características propias de los lenguajes de programación, como lotes de instrucciones, variables locales, bucles e instrucciones específicas para el tratamiento de datos. T-SQL supone una herramienta con un gran potencial para la gestión de las bases de datos (Ben-Gan, 2012). Dentro de una base de datos de SQL Server se pueden almacenar conjuntos de instrucciones de T-SQL en forma de procedimientos almacenados y funciones.

Un **procedimiento almacenado** es un conjunto de instrucciones T-SQL que puede recibir parámetros de entrada, retornar parámetros de salida, ejecutar otros procedimientos y funciones y ser llamados desde otros procedimientos. Una **función** es una rutina que recibe parámetros y devuelve un valor resultado de una acción utilizando los valores de los parámetros. La capacidad de definir procedimientos y funciones en una base de datos proporciona diversas ventajas, como la modularidad, reutilización de código, facilidad de mantenimiento, mayor rendimiento en la ejecución o mayor seguridad.

Además del **motor de bases de datos**, que administra todos los componentes de una instancia de SQL Server en un servidor, se utilizará el servicio **SQL Server Agent**, destinado a la administración y seguimiento de tareas programadas (trabajos y alertas) en una instancia de SQL Server.

Otra tecnología muy importante es el motor de **Integration Services**, dedicado a la administración de las soluciones de **SQL Server Integration Services (SSIS)** desplegadas en una instancia de SQL Server. SSIS es una plataforma para el desarrollo de soluciones de ETL, integración y análisis de datos mediante la conexión con múltiples fuentes de datos. SSIS proporciona tanto herramientas gráficas como de programación para la implementación de soluciones.

Otros servicios que cabe mencionar son SQL Server Browser, servicio que atiende las solicitudes de conexión a todas las instancias del SQL Server de un equipo, Analysis Services (para administración de soluciones OLAP y minería de datos), Reporting Services (administración de informes

[8] `https://www.microsoft.com/es-es/sql-server/sql-server-2019`, Microsoft.
[9] Solid IT DB-Engines Ranking `https://db-engines.com/en/ranking`. Última visita: febrero de 2020.

tabulares) o Machine Learning Services (control de soluciones de aprendizaje automático). SQL Server es capaz de procesar, además de consultas T-SQL, expresiones multidimensionales (MDX) y de minería de datos (DMX), consultas XML (XMLA) y expresiones de análisis de datos (DAX).

SQL Server Management Studio (SSMS) es el entorno gráfico que facilita una interfaz común para administrar todos los servicios proporcionados por SQL Server.

Además de SQL Server, Microsoft posee otros productos para el almacenamiento y gestión de datos como Azure SQL Database[10], RDBMS basado en SQL Server pero en una plataforma de servicios *cloud* (PaaS) o Azure Cosmos DB[11], una plataforma DBMS con enfoque no relacional. De esta forma, Microsoft ha sabido adaptarse a los cambios que se están produciendo en este área, ya que se estima que en 2023 el 75 % de las bases de datos podrían estar en una plataforma en la nube (Adrian y cols., 2019).

3.7.2.2. Power BI

Power BI®[12] es una tecnología de Microsoft que proporciona múltiples herramientas para el análisis y visualización de la información del negocio. Son varios los productos englobados dentro de Power BI. **Power BI Desktop** es una aplicación de escritorio gratuita que facilita el desarrollo de informes. Un informe es una agrupación de varios objetos visuales, divididos en páginas, que exponen diferentes características de un conjunto de datos.

La información mostrada proviene de un modelo tabular de datos que Power BI genera internamente en el informe a partir de la declaración de consultas a fuentes de datos. Existen infinidad de conectores para poder obtener datos de múltiples fuentes. El modelo tabular puede implementarse en modo de importación (copia de datos) o modo *DirectQuery* (consulta directa). Además, se pueden aplicar operaciones para modificar la implementación del modelo tabular, basadas en el lenguaje de Power Query.

Power BI detecta las columnas que pueden agregarse (hechos) y las que proporcionan niveles de detalle para las agregaciones (dimensiones) en el modelo de datos. A partir de ellas define **medidas**, operaciones multidimensionales que pueden utilizarse para construir las visualizaciones del informe. El motor de consultas de Power BI es el encargado ejecutar las operaciones multidimensionales sobre el modelo tabular. Además, permite la construcción de otros objetos como jerarquías, nuevas medidas o columnas derivadas utilizando el lenguaje DAX.

Otros productos son **Power BI Pro** y **Premium**, que proporcionan servicios web en los que poder publicar y compartir los datos e informes creados en Desktop dentro de una organización. También permiten generar otros artefactos a partir de ellos, como paneles y cuadros de mando. Con

[10] `https://azure.microsoft.com/es-es/`, Microsoft Azure.
[11] `https://azure.microsoft.com/es-es/services/cosmos-db/`, Microsoft Azure.
[12] `https://powerbi.microsoft.com/`, Microsoft.

Power BI Mobile se pueden crear aplicaciones para proporcionar accesibilidad a la información desde cualquier dispositivo, y el **Servidor de informes de Power BI** facilita un sistema de reporting para la compañía.

Richardson y cols. (2020) sitúan como líder de las plataformas BI a Microsoft con Power BI, y destacan, entre otras, las capacidades en aprendizaje automático y analítica aumentada del servicio Pro.

3.7.3. Otras herramientas

- **Overleaf[13]:** plataforma online de LaTeX, editor de texto en el que se realizará la documentación. Se opta por un editor en la nube frente a otros editores de LaTeXde escritorio por la posibilidad de editar en diferentes dispositivos.
- **GitLab[14]:** plataforma en la nube para el control de versiones en el desarrollo de software. Se utilizará para hacer un seguimiento de la planificación y versionar el código generado en el desarrollo.
- **Visual Studio Code[15]:** herramienta de edición de código desarrollada por Microsoft. Tendrá una doble función en el proyecto, ya que se utilizará como entorno para el desarrollo de código y para administrar los cambios entre el repositorio remoto de GitLab y el local.
- **Draw.io[16]:** herramienta web que permite realizar múltiples tipos de diagramas. Se utilizará para dibujar esquemas del sistema o la arquitectura.
- **Microsoft One Note[17]:** aplicación de Office para apuntar notas, cambios e ideas.

[13] https://www.overleaf.com/
[14] https://about.gitlab.com/
[15] https://code.visualstudio.com/
[16] https://app.diagrams.net/
[17] https://www.microsoft.com/es-es/microsoft-365/onenote/digital-note-taking-app

4. Requisitos y Análisis

Tomando como base el ejemplo del caso de negocio y el planteamiento de la funcionalidad realizado en el Capítulo 2, en este capítulo se listarán los requisitos de la herramienta y se desarrollará el análisis, produciendo como resultado el PIM.

4.1. Actores

Se distinguen dos actores:

- El **Administrador** del servidor de bases de datos, que realiza todas las funciones de la herramienta.
- El **Servidor de bases de datos**, que realiza varias acciones en función de su interacción con la herramienta.

4.2. Requisitos de usuario

El diagrama de los casos de uso se puede ver en la Figura 4.1, junto con una descripción detallada de cada uno.

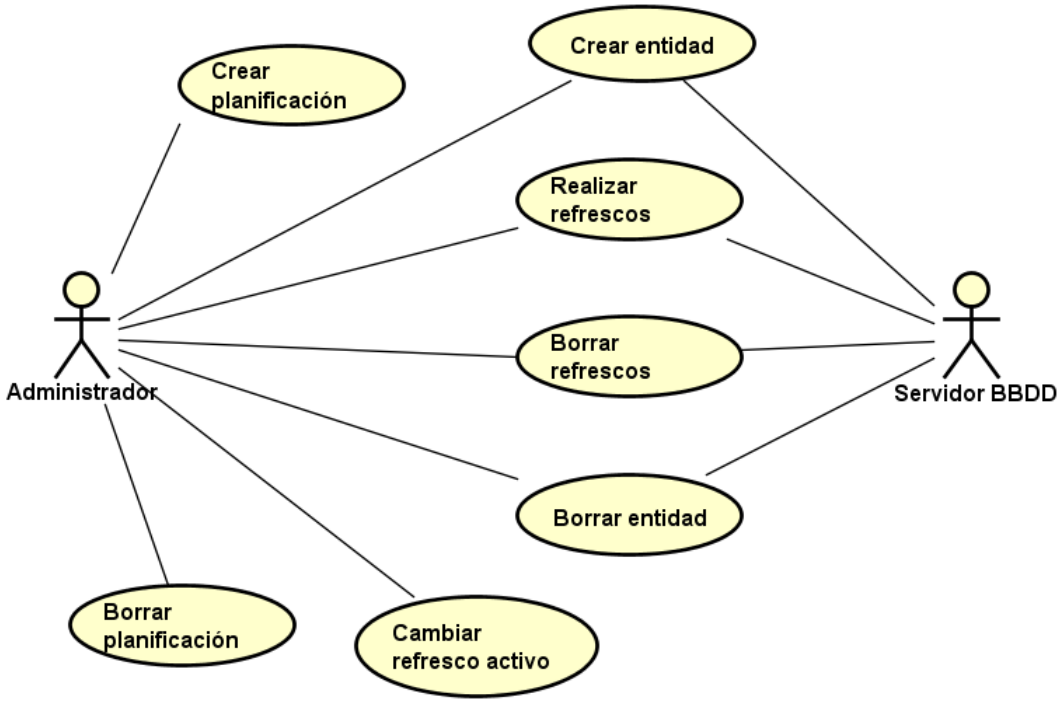

Figura 4.1.: Diagrama de casos de uso

CU1	**Crear entidad**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite crear una nueva entidad
Actores	Administrador, Servidor BBDD
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere crear una nueva entidad especificando los valores de los atributos determinados y los nombres de las planificaciones de refresco asociadas 2 El sistema indica al Servidor BBDD que hay que crear la estructura de datos de la entidad en la localización especificada 3 El Servidor BBDD crea la estructura de datos 4 El sistema crea la entidad con los valores de atributos introducidos
Flujos alternativos	-
Flujos de excepción	2a Los valores introducidos no son correctos 2b Ya existe una entidad con el mismo nombre y localización 2c Alguno de los nombres introducidos no se corresponde a una planificación del sistema
Postcondiciones	La entidad y su estructura de datos se han creado
Dependencias	RF1, RF2

Tabla 4.1.: Descripción del caso de uso CU1 Crear entidad

CU2	**Crear planificación**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite crear una nueva planificación
Actores	Administrador
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere crear una nueva planificación especificando los valores para los atributos 2 El sistema crea la planificación con los valores introducidos
Flujos alternativos	-
Flujos de excepción	2a Los valores introducidos no son correctos 2b Ya existe una planificación con el mismo nombre 2c Ya existe una planificación con los mismos valores para los atributos
Postcondiciones	La planificación se ha creado
Dependencias	RF3, RF4, RF5, RF6, RF7, RF8

Tabla 4.2.: Descripción del caso de uso CU2 Crear planificación

CU3	**Realizar refrescos**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite realizar nuevos refrescos
Actores	Administrador, Servidor BBDD
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere realizar nuevos refrescos especificando la fecha de carga 2 El sistema recoge todas las entidades a refrescar para la fecha introducida 3 El sistema indica al Servidor BBDD que hay que eliminar los datos del refresco más antiguo para una entidad a refrescar 4 El Servidor BBDD elimina los datos 5 El sistema indica al Servidor BBDD que hay que obtener los datos del nuevo refresco 6 El Servidor BBDD obtiene los datos 7 El sistema indica al Servidor BBDD que hay que cargar los datos del nuevo refresco en la estructura de datos de una entidad a refrescar 8 El Servidor BBDD añade los datos 9 El sistema crea el nuevo refresco y vuelve al paso 3
Flujos alternativos	3a No existen entidades sin refrescar para la fecha introducida. El caso de uso termina
Flujos de excepción	-
Postcondiciones	La carga de los datos se ha llevado a cabo y los refrescos se han creado
Dependencias	RF9, RF10

Tabla 4.3.: Descripción del caso de uso CU3 Realizar refrescos

CU4	**Borrar refrescos**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite borrar refrescos de una entidad
Actores	Administrador, Servidor BBDD
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere borrar refrescos para una entidad, especificando su nombre y localización y las fechas de inicio y fin de borrado 4 El sistema indica al Servidor BBDD que hay que eliminar los datos de todos los refrescos de la entidad en el rango de fechas especificado 5 El Servidor BBDD elimina los datos 6 El sistema elimina los refrescos de la entidad en el rango de fechas especificado
Flujos alternativos	1a El Administrador introduce todos los datos excepto la fecha fin de borrado. El sistema asigna a la fecha fin el valor introducido para la de inicio
Flujos de excepción	2a No existe una entidad con el nombre y localización dados 2b La fechas introducidas no son un rango de fechas correcto
Postcondiciones	Los datos de los refrescos y los refrescos han sido eliminados
Dependencias	RF11, RF12

Tabla 4.4.: Descripción del caso de uso CU4 Borrar refrescos

CU5	**Cambiar refresco activo**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite cambiar el refresco activo de una entidad
Actores	Administrador
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere cambiar el refresco activo de una entidad especificando su nombre y localización y valor del atributo activo 2 El sistema cambia el valor del atributo activo de la entidad especificada
Flujos alternativos	-
Flujos de excepción	2a No existe una entidad con nombre y localización dados
Postcondiciones	Se ha cambiado el valor del atributo activo para la entidad
Dependencias	RF13, RF14

Tabla 4.5.: Descripción del caso de uso CU5 Cambiar refresco activo

CU6	**Borrar entidad**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite eliminar una entidad
Actores	Administrador, Servidor BBDD
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere eliminar una entidad especificando nombre y localización de la entidad 2 El sistema indica al Servidor BBDD que hay que eliminar la estructura de datos de la entidad 3 El Servidor BBDD elimina la estructura de datos 4 El sistema elimina todos los refrescos de la entidad 5 El sistema elimina la entidad
Flujos alternativos	-
Flujos de excepción	2a No existe una entidad con nombre y localización especificados
Postcondiciones	Se ha eliminado la entidad y su estructura de datos
Dependencias	RF15

Tabla 4.6.: Descripción del caso de uso CU6 Borrar entidad

CU7	**Borrar planificación**
Descripción	El sistema deberá comportarse como se describe cuando el Administrador solicite eliminar una planificación
Actores	Administrador
Precondiciones	-
Flujo normal	1 El Administrador indica al sistema que quiere eliminar una planificación especificando su nombre 2 El sistema elimina la planificación
Flujos alternativos	-
Flujos de excepción	2a No existe una planificación con nombre especificado 2b La planificación está asociada a una o varias entidades existentes
Postcondiciones	Se ha eliminado la planificación
Dependencias	RF16

Tabla 4.7.: Descripción del caso de uso CU7 Borrar planificación

4.3. Requisitos funcionales y no funcionales

4.3.1. Requisitos funcionales

De CU1 *Crear entidad*:

- **RF1:** El sistema permitirá definir nuevas entidades.
- **RF2:** El sistema permitirá definir entidades con localización en el almacén de explotación.

De CU2 *Crear planificación*:

- **RF3:** El sistema permitirá definir nuevas planificaciones de refresco.
- **RF4:** El sistema permitirá definir planificaciones de refresco con periodicidad diaria, semanal, mensual y ocasional.
- **RF5:** El sistema permitirá definir planificaciones con refresco semanal en cualquier día de la semana.
- **RF6:** El sistema permitirá definir planificaciones con refresco mensual en cualquier día del mes.
- **RF7:** El sistema permitirá definir planificaciones con refresco ocasional en cualquier fecha.
- **RF8:** El sistema comprobará que las planificaciones no superen una frecuencia superior a la anual.

De CU3 *Realizar refrescos*:

- **RF9:** El sistema realizará para una fecha concreta el refresco de todas las entidades cuyas planificaciones establezcan un refresco en esa fecha.
- **RF10:** El sistema eliminará el refresco más antiguo si una entidad supera el máximo de refrescos almacenados de una planificación concreta.

De CU4 *Borrar refrescos*:

- **RF11:** El sistema permitirá eliminar el refresco de una entidad en una fecha concreta.
- **RF12:** El sistema permitirá eliminar todos los refrescos de una entidad en el rango entre dos fechas concretas.

De CU5 *Desactivar refresco activo*:

- **RF13:** El sistema permitirá desactivar el refresco de una entidad.
- **RF14:** El sistema permitirá activar el refresco de una entidad.

De CU6 *Borrar entidad*:

- **RF15:** El sistema permitirá eliminar una entidad.

De CU7 *Borrar planificación*:

- **RF16:** El sistema permitirá eliminar una planificación.

4.3.2. Requisitos funcionales de información

- **RF17:** El sistema deberá almacenar información de las entidades que componen el modelo de datos y sus características (nombre, tipo, localización física, si tiene el refresco activo, consulta y la fecha a partir de la cual se refresca).
- **RF18:** El sistema deberá almacenar información de las planificaciones que establecen el refresco de las entidades y sus características (nombre, periodicidad del refresco, día si es semanal o mensual, frecuencia, máximo de refrescos, fecha de refresco ocasional).
- **RF19:** El sistema deberá almacenar información de la fecha de los refrescos que se han realizado en cada entidad.

4.3.3. Requisitos no funcionales

- **RNF1:** Se utilizará la tecnología de Microsoft BI para implementar el sistema.
- **RNF2:** El motor de consultas de SQL Server llevará a cabo todas las funcionalidades del sistema.
- **RNF3:** Se utilizará la interfaz de SSMS para acceder a todas las funcionalidades.
- **RNF4:** No pueden existir dos entidades con el mismo nombre y localización.
- **RNF5:** No pueden existir dos planificaciones con el mismo nombre.
- **RNF6:** No pueden existir dos planificaciones con las mismas características.
- **RNF7:** Una entidad no puede estar asociada a más de 10 planificaciones.
- **RNF8:** El sistema creará una tabla por cada entidad con el nombre y en la localización especificados.
- **RNF9:** Las consultas de las entidades definidas podrán consultar cualquier base de datos y esquema dentro de la instancia de SQL Server.
- **RNF10:** Las consultas de las entidades estarán sujetas a las restricciones propias de las subconsultas.
- **RNF11:** Se mantendrá un log de las ejecuciones de los principales procedimientos almacenados.

4.4. Modelo del dominio

El diagrama del modelo del dominio se puede ver en la Figura 4.2. Las entidades que contiene son las siguientes:

- **Entidad:** almacena la información de las entidades que componen los modelos.
- **Planificación:** contiene la información que establece un calendario de refrescos determinado. Una planificación no está asociada a una única entidad, sino que puede utilizarse para programar refrescos de varias entidades. Los posibles valores de los atributos *día*, *frecuencia* y *fecha ocasional* dependen de la periodicidad escogida. Esta dependencia se puede ver en la Tabla 4.8.
- **Refresco:** almacena la fecha de los refrescos realizados a cada Entidad.

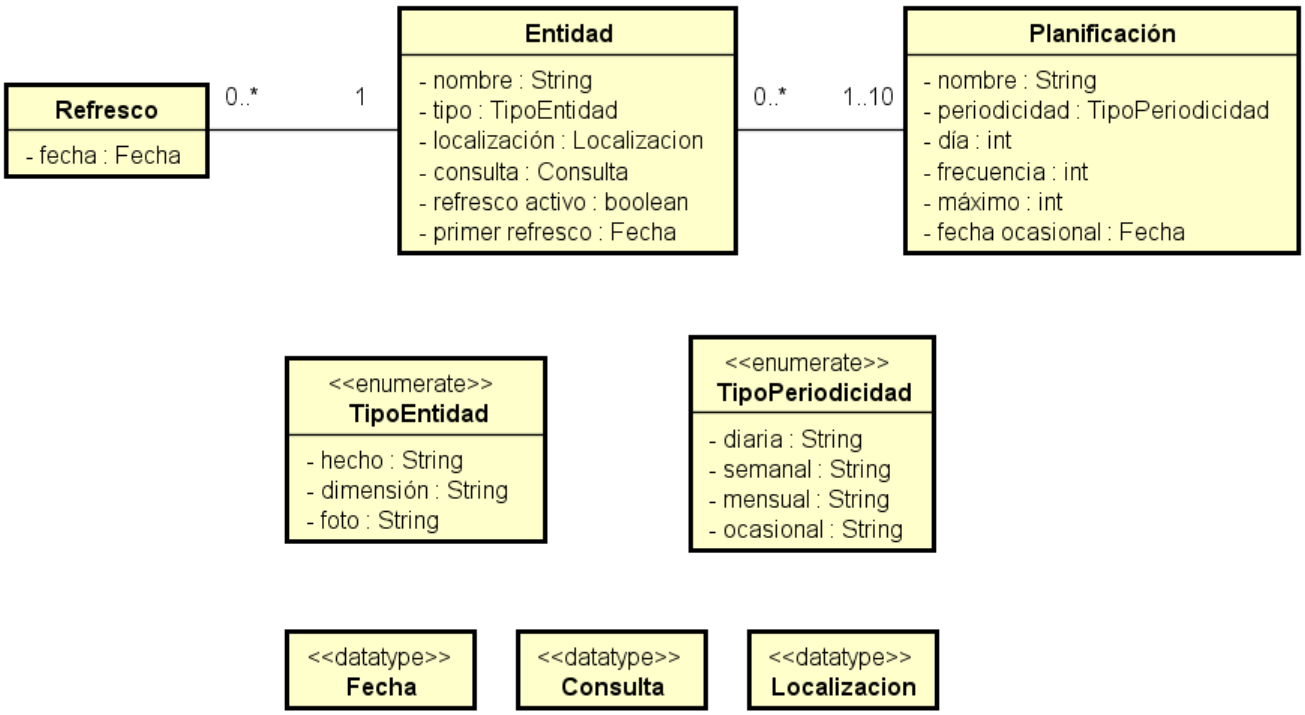

Figura 4.2.: Diagrama del modelo del dominio

El diccionario de datos se puede ver en la Tabla 4.9.

4.5. Realización en análisis de los casos de uso

Las figuras 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 y 4.9 incluyen los diagramas de secuencia que representan la realización en análisis de los casos de uso.

periodicidad	*día*	*frecuencia*	*fecha ocasional*
'diaria'	-	1 a 365	-
'semanal'	1 a 7	1 a 52	-
'mensual'	1 a 31	1 a 12	-
'ocasional'	-	-	Fecha

Tabla 4.8.: Dependencia de los atributos *día*, *frecuencia* y *fecha ocasional* según *periodicidad*

Tipo de datos	**Descripción**
boolean	puede tomar dos valores, que representan verdadero o falso
int	valores numéricos de tipo entero
String	cadenas de caracteres
Fecha	valores que representan una fecha
TipoEntidad	puede tomar 3 valores String, especificados en el modelo del dominio
TipoPeriodicidad	puede tomar 4 valores String, especificados en el modelo del dominio
Localizacion	valores que representan una localización dentro del servidor de bases de datos
Consulta	valores que representan una consulta en el servidor de bases de datos

Tabla 4.9.: Diccionario de datos

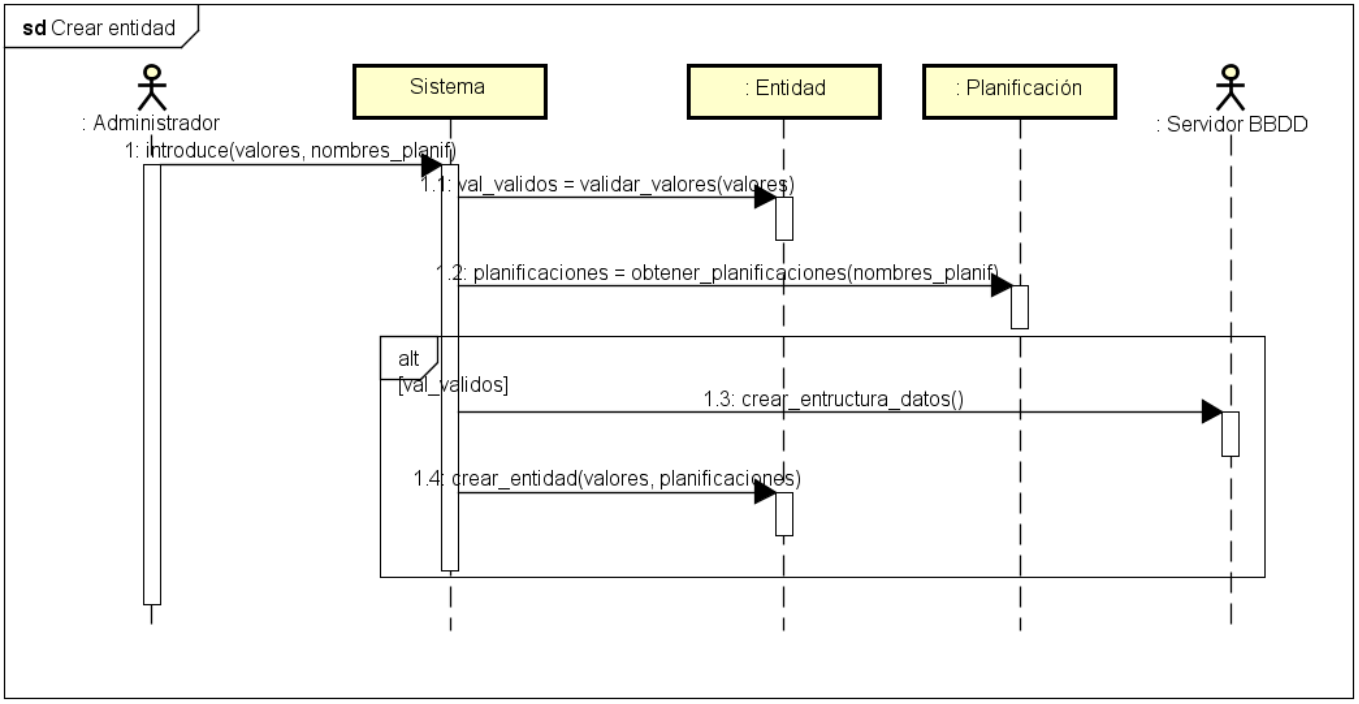

Figura 4.3.: Diagrama de secuencia CU1 *Crear entidad*

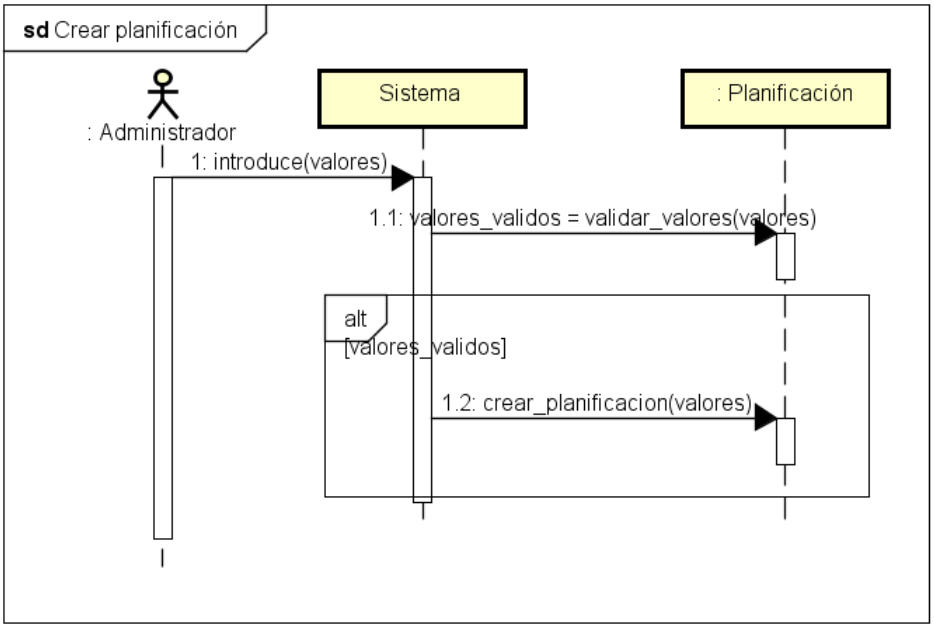

Figura 4.4.: Diagrama de secuencia CU2 *Crear planificación*

4.6. Modelo independiente de la plataforma

El archivo **PIM.xmi** contiene el PIM, formado por el modelo de dominio en el estándar XMI, el cual puede ser importado por cualquier herramienta de modelado. La especificación en XMI de los modelos incluye la estructura de entidades, atributos y operaciones, pero no los diagramas ni los estereotipos de las entidades[1].

Por ello, también se ha creado el archivo **PIM.xml**. Éste contiene el modelo completo codificado en XML, incluyendo los diagramas, y puede ser importado por cualquier versión de Astah y de otras herramientas que lo soporten.

[1]XMI Export `https://astah.net/product-plugins/xmi-export/`, Astah. Última visita: marzo de 2020.

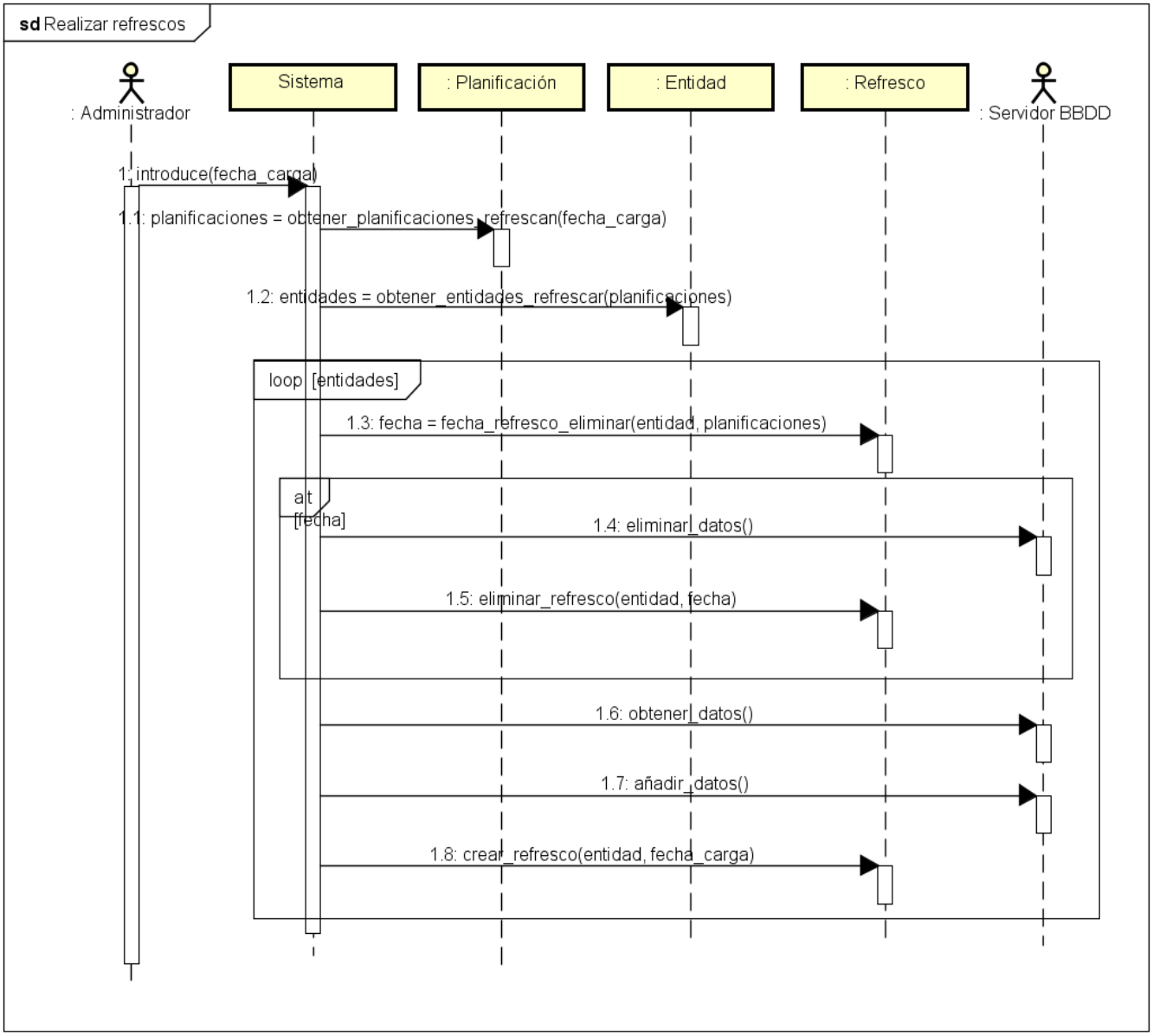

Figura 4.5.: Diagrama de secuencia CU3 *Realizar refrescos*

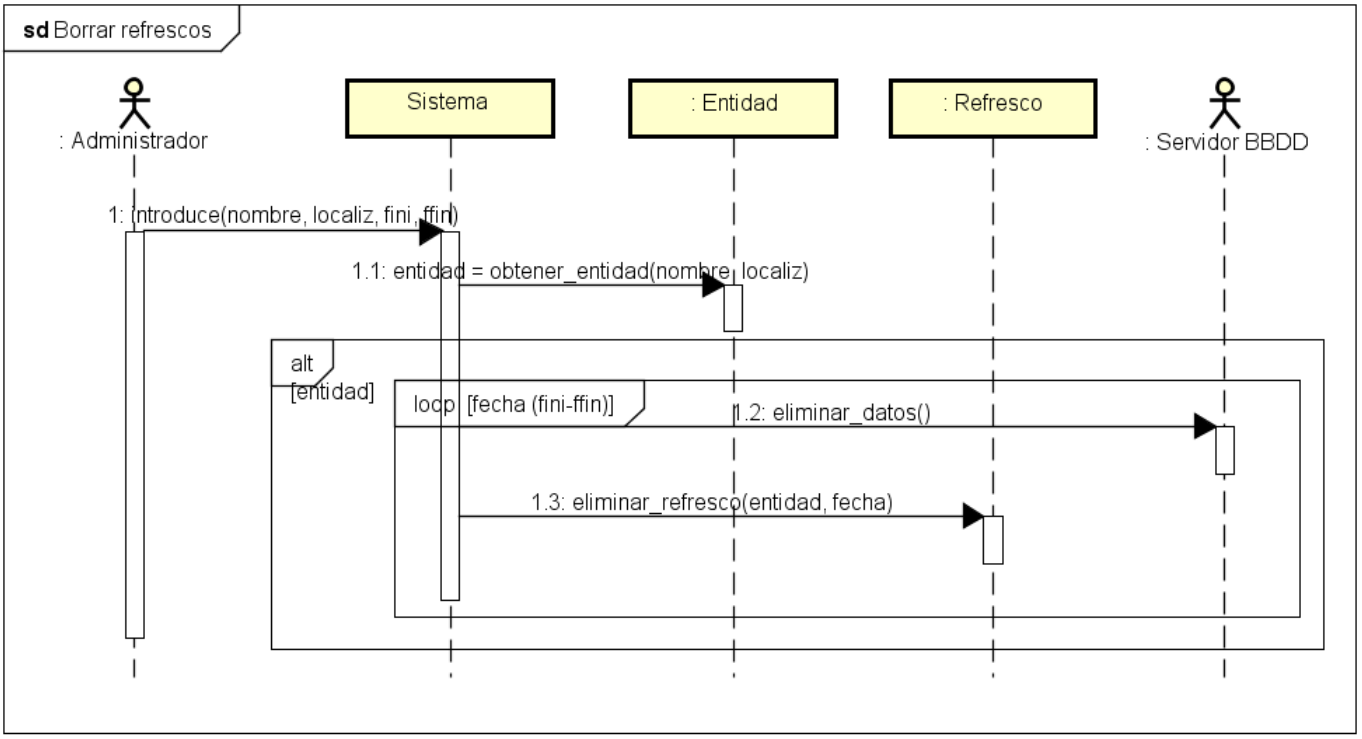

Figura 4.6.: Diagrama de secuencia CU4 *Borrar refrescos*

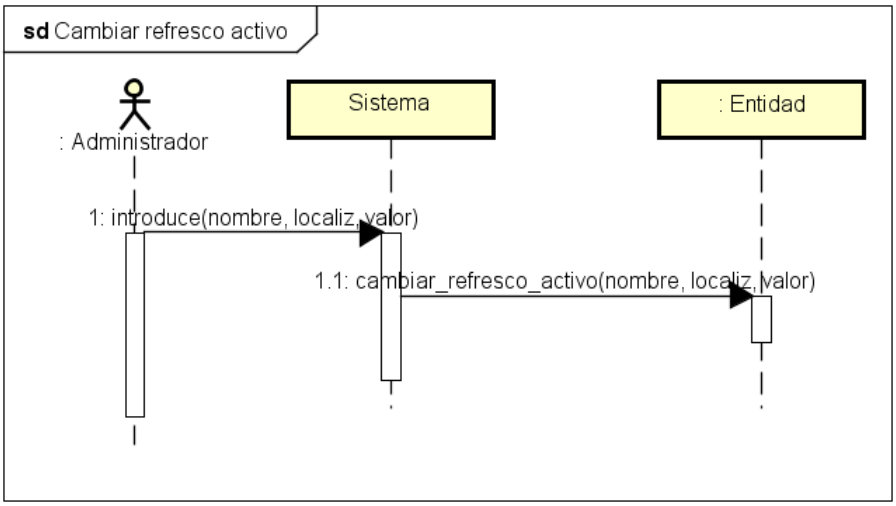

Figura 4.7.: Diagrama de secuencia CU5 *Cambiar refresco activo*

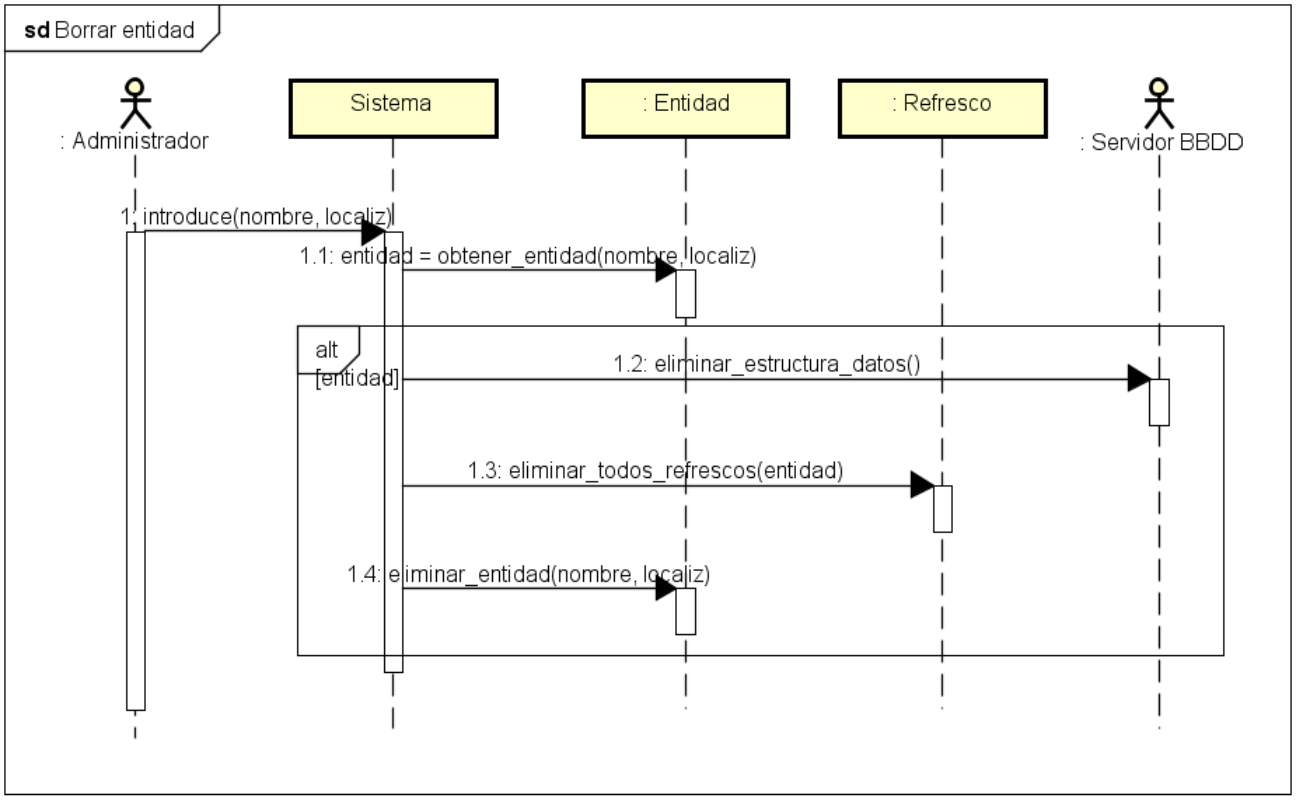

Figura 4.8.: Diagrama de secuencia CU6 *Borrar entidad*

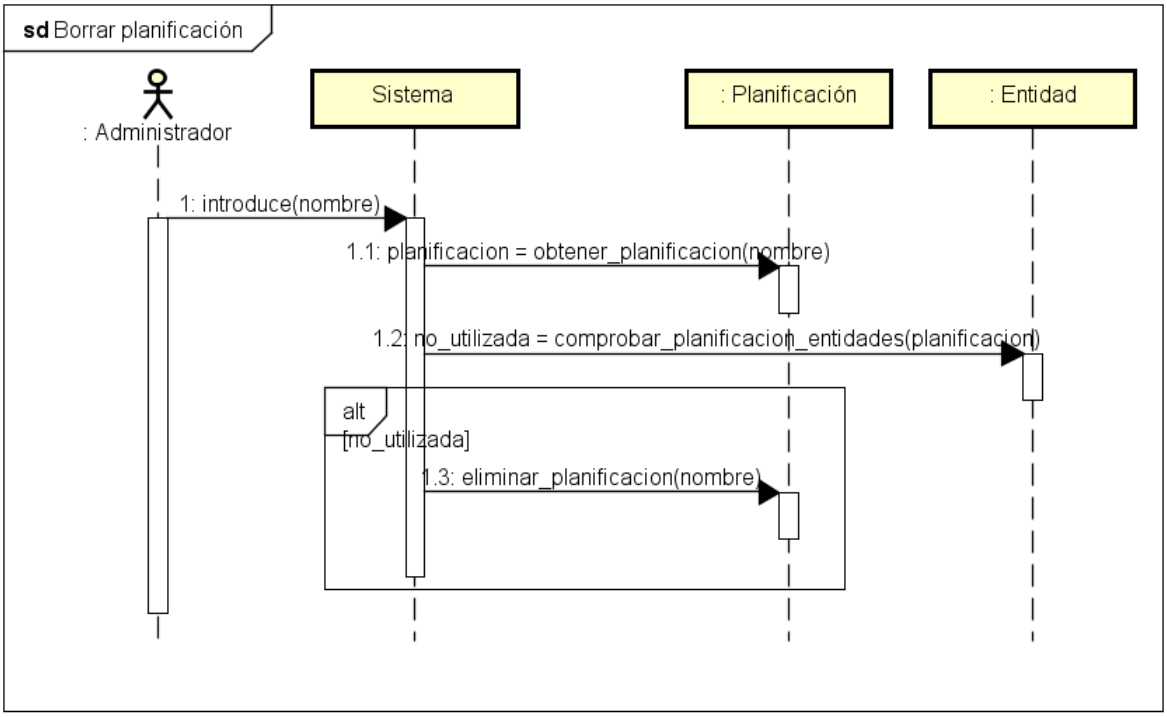

Figura 4.9.: Diagrama de secuencia CU7 *Borrar planificación*

5. Diseño

En este capítulo se describirá detalladamente el diseño de la herramienta. En primer lugar, se mostrará la arquitectura escogida para implementar la solución. Seguidamente, se explicará el diseño de las consultas y las cargas de datos al efectuar los refrescos. A continuación, se describirá la realización de los casos de uso en diseño. Por último, se obtendrá el PSM de la herramienta.

5.1. Arquitectura

La realización de los casos de uso de la herramienta requiere de diversas interacciones con el servidor de bases de datos que almacena la información del negocio, sobre todo en el caso de uso CU3 *Realizar refrescos*: después de seleccionar las entidades de los modelos que deben refrescarse para la fecha indicada por el Administrador, la herramienta debe ejecutar para cada entidad la consulta sobre el servidor, que puede apuntar a una o varias bases de datos.

A continuación deberá cargar los datos devueltos por el servidor en la tabla de la entidad. Dependiendo de la localización especificada para cada entidad, la tabla puede estar en diferentes bases de datos de destino dentro del servidor.

Si hay varios modelos de datos generados en varias bases de datos, y cada modelo posee un alto número de entidades, las operaciones de datos sobre el servidor al realizar los refrescos podrían ser muy costosas. Por ello, se necesita enfocar el diseño de la herramienta en la optimización de estas operaciones.

Microsoft BI es la tecnología escogida para construir el sistema. Gracias a la programación procedimental incluida en el lenguaje de desarrollo T-SQL[1], se puede implementar toda la funcionalidad de la herramienta en el propio servidor de bases de datos de SQL Server para que el motor de consultas del servidor realice las operaciones. Es decir, se implementará el sistema como un **motor propio** dentro del servidor de bases de datos.

Dentro de una instancia de SQL Server se creará una base de datos llamada MOTOR, donde se persistirá la información del motor y se ejecutarán los procedimientos y funciones que implementan la funcionalidad del motor. A su vez, la instancia contendrá la base de datos BASE, al que apuntarán las consultas de las entidades del modelo que el motor debe generar en la base de datos

[1]Transact-SQL, extensión de SQL desarrollada por Microsoft (página 33).

EXPL, que formará el almacén de explotación. El modelo implementado en esta última base de datos será utilizado como origen de datos para la plataforma de visualización en Power BI.

El Administrador del servidor de bases de datos utilizará la interfaz que proporciona SSMS[2] para realizar los casos de uso. Esta interfaz es mucho más completa que cualquier otra que se construya desde cero, ya que ofrece múltiples herramientas de administración del servidor, y acceder a las funcionalidades del motor será una de ellas. Se supondrá que el Administrador de SQL Server sabrá utilizar la interfaz, además de poseer conocimientos avanzados de SQL y de bases de datos.

Un esquema del sistema de información se puede ver en la Figura 5.1.

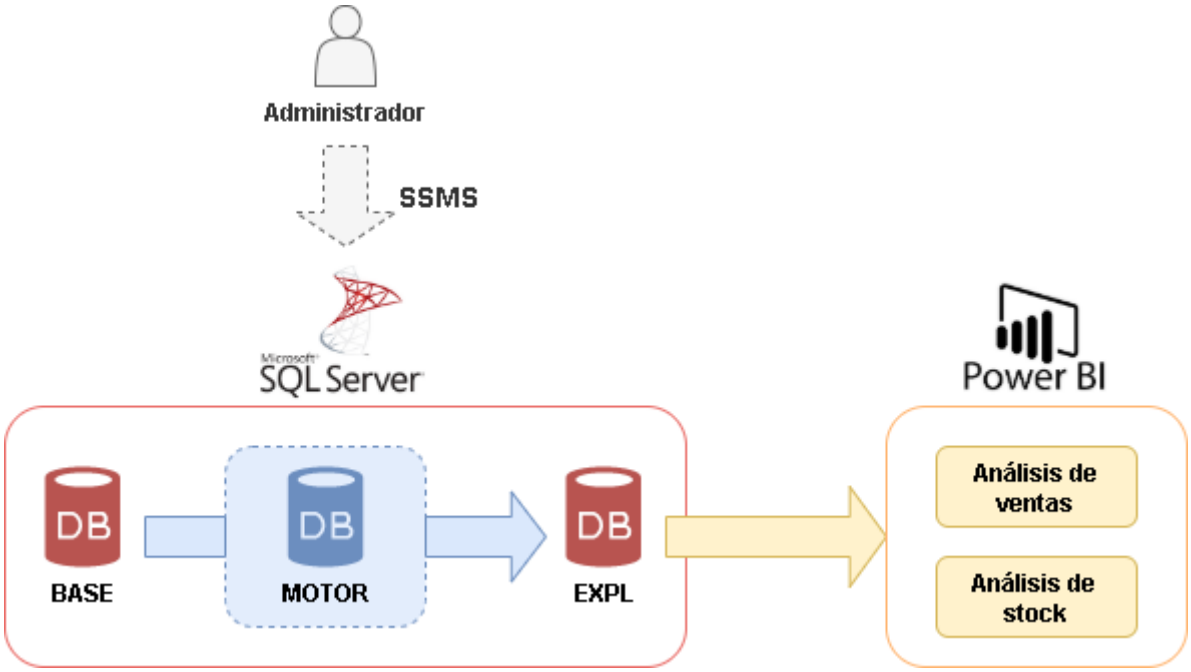

Figura 5.1.: Esquema general del sistema de información

Entre las ventajas de utilizar procedimientos almacenados para realizar la funcionalidad del motor, se puede señalar:

- No hace falta incluir otro tipo de software que lleve a cabo las operaciones del motor. Esto supone que no se necesita depender de otras herramientas ni configurar conexiones externas al servidor de bases de datos, lo cual facilita el desarrollo y simplifica la implementación del motor.
- El lenguaje T-SQL está optimizado expresamente para ejecutar tareas sobre los datos almacenados en la instancia de SQL Server, por lo que el rendimiento de las operaciones sobre el servidor será muy alto. Además, incluye instrucciones específicas de administración de datos que facilitan la implementación del motor.

[2]SQL Server Management Studio, interfaz de usuario para la administración de una instancia de SQL Server (página 33).

- El movimiento de datos queda localizado dentro de SQL Server, lo que aumenta la seguridad.

Para la implementación del motor se ha escogido una arquitectura por capas, con lo que se consigue desacoplar las distintas funcionalidades del mismo. Las tres capas de las que se compone el sistema son:

- **Lógica:** controla la información almacenada dentro de la base de datos del motor y las operaciones que debe realizar. Utiliza los servicios de la capa externa.
- **Externa:** prepara y realiza las consultas T-SQL al resto de bases de datos del servidor, en este caso, BASE y EXPL.
- **Común:** capa relajada que presta servicios a las otras dos capas.

SSMS será el subsistema que se utilizará como interfaz de usuario y utilizará los servicios de la capa lógica.

Como T-SQL no es un lenguaje de programación orientado a objetos, no se pueden implementar clases ni construir instancias. Sin embargo, se pueden utilizar las propiedades de los esquemas de SQL Server para generar un **espacio de nombres** en los que agrupar los procedimientos almacenados y funciones en módulos según la funcionalidad que llevan a cabo.

Esquemas que constituyen la capa lógica:

- ***main*:** con los procedimientos que iniciará SSMS según la interacción del Administrador, uno por cada caso de uso.
- ***controlador*:** con los procedimientos que realizan los casos de uso, y el log de las operaciones realizadas.
- ***modelo*:** con los procedimientos que realizan consultas y modificaciones sobre la información del motor (entidades, planificaciones y refrescos), y las tablas donde se almacena esta información.

Esquemas que constituyen la capa externa:

- ***fachada*:** procedimientos que construyen las instrucciones T-SQL que van a lanzarse sobre otras bases de datos.
- ***consultas*:** procedimientos que ejecutan las instrucciones T-SQL sobre otras bases de datos (EXPL y BASE).

La capa relajada se compone del esquema por defecto que contienen todas las bases de datos de SQL Server, ***dbo***. Los objetos de MOTOR a los que no se especifique esquema estarán incluidos en dbo.

Como el esquema *modelo* contendrá las tablas donde se almacenan los datos de las entidades, planificaciones y refrescos, no necesita utilizar los servicios de la capa externa para obtener y

comunicar cambios sobre la información del motor. Los procedimientos de este esquema podrán consultarlos y modificarlos directamente.

En la Figura 5.2 se puede ver el diagrama de la arquitectura, donde vienen representadas las capas, los módulos y sus dependencias entre sí.

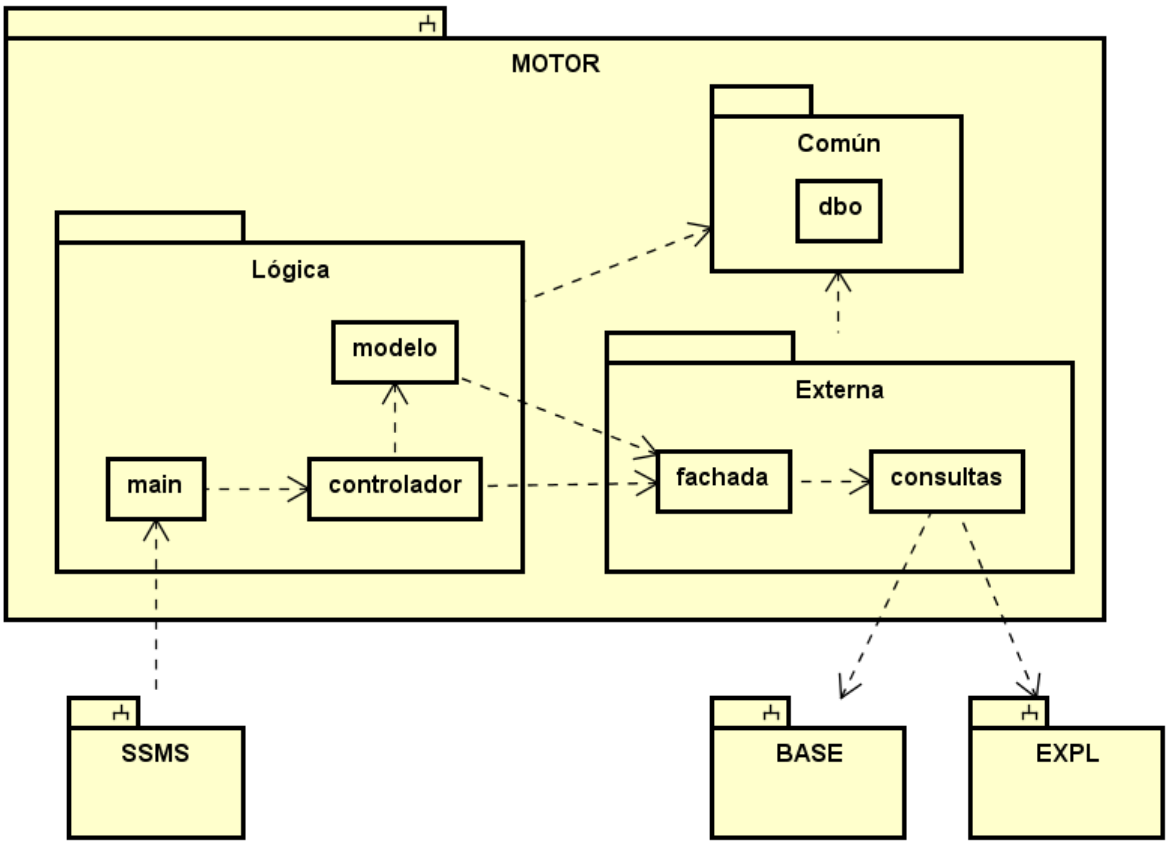

Figura 5.2.: Diagrama de la arquitectura del sistema

5.2. Diseño de las consultas y las cargas

La realización de varios de los casos de uso implican operaciones de inserciones y borrado de datos sobre las estructuras de datos de las entidades, que pueden ser muy complejas dependiendo de las consultas y el tipo de estas estructuras de datos.

Para simplificar las operaciones, se han tomado dos decisiones:

- El Administrador será el encargado de construir las consultas de las entidades, las cuales especificará como **cadenas de texto**. De esta forma se disminuye en gran medida la complejidad en el tratamiento de las consultas y proporciona la posibilidad de especificar cualquier tipo de consulta.

- Todas las entidades se materializarán físicamente en **tablas básicas**, ya que otros tipos de tablas o estructuras necesitan de tratamientos específicos para la inserción y borrado.

El motor, en un nuevo refresco de la entidad, ejecutará la consulta sobre el servidor para obtener los datos a cargar en la tabla. Sería muy conveniente que la consulta pueda acceder a la fecha indicada por el Administrador al iniciar el CU3 *Realizar refrescos*. Además de proporcionar mayor flexibilidad, permite que el motor pueda realizar **cargas incrementales** (Rahman, 2012) en las tablas de las entidades. Esto es muy importante para aumentar el rendimiento del motor cuando trabaja con grandes volúmenes de datos.

Poniendo un ejemplo dentro del caso de negocio tratado, se quiere obtener una dimensión desnormalizada de los productos con las marcas y categorías. Los productos pueden salir a la venta en cualquier fecha, por lo que esta entidad se debe asociar con una planificación diaria, que todos los días cargue los datos de BASE a la tabla de la entidad.

En una carga normal se cargarían de forma diaria todos los productos que han salido hasta la fecha. La consulta sería la siguiente:

```
SELECT p.id_producto, p.nombre_producto, p.precio,
p.fecha_salida, m.nombre_marca, c.nombre_categoria
FROM BASE.produccion.producto p
JOIN BASE.produccion.marca m ON m.id_marca = p.id_marca
JOIN BASE.produccion.categoria c ON c.id_categoria = p.id_categoria
```

En este caso se estarían realizando muchas inserciones de filas innecesarias, porque los productos no se modifican y sólo habrá nuevos productos en día sueltos. Sin embargo, si estuviera disponible la fecha en el que se realiza la carga para poder ser utilizada en las consultas, se puede añadir una cláusula WHERE en la consulta que filtre los datos devueltos:

```
SELECT p.id_producto, p.nombre_producto, p.precio,
p.fecha_salida, m.nombre_marca, c.nombre_categoria
FROM BASE.produccion.producto p
JOIN BASE.produccion.marca m ON m.id_marca = p.id_marca
JOIN BASE.produccion.categoria c ON c.id_categoria = p.id_categoria
WHERE p.fecha_salida = [fecha carga]
```

De esta forma, en cada refresco sólo se insertarán los productos cuya fecha de salida sea equivalente a la fecha de carga.

Para ello, el Administrador podrá **indicar con el símbolo '$' y la palabra clave FECHA_CARGA** que quiere utilizar la fecha de carga en esa posición de la consulta. El motor se encargará de sustituir el texto '$FECHA_CARGA' por el valor actual de la fecha de carga de forma dinámica, en formato numérico *yyyymmdd*.

La instrucción INSERT INTO ... SELECT es una práctica recomendada para la inserción masiva de datos [3]. Se puede usar para insertar los datos de un nuevo refresco de la forma:

```
INSERT INTO [tabla] SELECT * FROM ([consulta]) AS sq
```

Utilizando esta forma de inserción no hace falta crear ninguna tabla auxiliar intermedia, sino que en una sola instrucción se obtienen e insertan los datos. Sin embargo, tiene dos implicaciones sobre las consultas y las tablas:

1. La consulta se utiliza como una **subconsulta** dentro de la instrucción.
2. La tabla de destino tiene que tener las mismas columnas y del mismo tipo que las que devuelve la consulta.

La primera implicación pone restricciones sobre el tipo de consultas que se pueden especificar. Estas restricciones son[4]:

- WITH: no se pueden incluir expresiones de tabla común (CTE)[5] en una subconsulta, por lo que no se puede utilizar la cláusula WITH. Si se quiere añadir una entidad cuya consulta contiene WITH, se deberá modificar primero para transformarla en una consulta equivalente sin esta cláusula, añadiendo la expresión de tabla común como una subconsulta en la cláusula FROM.
- ORDER BY: sólo se puede incluir la cláusula ORDER BY cuando también hay una cláusula TOP en la consulta. Esto en realidad no supone ningún problema, porque la tarea de ordenación de las filas dependerá de la plataforma de explotación de datos. Por otra parte, sí se puede incluir ORDER BY si completa otra cláusula, como PARTITION.

Con respecto a la segunda, la consulta que se especifica al crear una nueva entidad definirá la estructura de columnas que contendrá la tabla de esa entidad.

El borrado de los datos de un refresco a priori no se podría realizar, ya que no sería posible vincular qué filas de la tabla de una entidad han sido generadas por cada refresco. Esto se puede solucionar añadiendo una **columna de control** llamada **fecha_refresco** en la tabla, que almacenará la fecha de carga (equivalente a la fecha del refresco concreto) que provocó la inserción de cada fila, lo que hace a las filas localizables a partir de los refrescos. Además de posibilitar el borrado de datos de un refresco, esta columna también es conveniente para explotar los datos de la tabla de una forma más completa.

La instrucción de borrado es, por tanto:

[3]INSERT (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/statements/insert-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: abril de 2020.

[4]Subconsultas (SQL Server) `https://docs.microsoft.com/es-es/sql/relational-databases/performance/subqueries?view=sql-server-ver15`, Microsoft Docs. Última visita: abril de 2020.

[5]WITH common_table_expression (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: abril de 2020.

```
DELETE FROM [tabla] WHERE fecha_refresco = [fecha de refresco a eliminar]
```

Esta columna se deberá añadir posteriormente a la creación de la tabla. Modificando la instrucción de inserción se pueden insertar los valores de la columna junto con los datos del nuevo refresco:

```
INSERT INTO [tabla] SELECT *, [fecha carga] FROM ([consulta]) AS sq
```

De esta forma se consigue que la inserción y borrado de los datos de un nuevo refresco sean simples y uniformes independientemente de las características de la entidad y la complejidad de su consulta.

5.3. Realización en diseño de los casos de uso

El almacenamiento de logs de las ejecuciones es general a todos los casos de uso. Cada vez que comience la ejecución de un procedimiento del esquema *controlador* que realiza un caso de uso, se almacenará en la tabla **controlador.ejecucion**, y los mensajes de error o información de la ejecución en **controlador.mensaje**. Cuando la ejecución termina, ya sea con éxito o error, se actualizarán los valores de la ejecución almacenada al inicio de la realización del caso de uso.

A continuación se describirá la realización en diseño de los casos de uso, entrando en el detalle de los pasos que sean más complejos.

5.3.1. Crear entidad

El procedimiento **main.CrearEntidad** inicia el CU1 *Crear entidad*. El Administrador podrá indicar los valores de los atributos de la entidad enviándolos como parámetros:

- *nombre*: texto que corresponderá al nombre de la entidad. Obligatorio.
- *tipo*: texto que indica el tipo de datos que contendrá la entidad (hechos, dimensiones, fotos). Obligatorio.
- *bd*: nombre de la base de datos donde se materializará la entidad. Obligatorio.
- *esquema*: nombre del esquema de la base de datos donde se materializará la entidad. Opcional, valor por defecto: dbo.
- *consulta*: cadena de texto de la consulta T-SQL cuyo resultado son los datos que almacenará la entidad. Obligatorio.
- *activo*: indica si el refresco de la entidad está activo. Opcional, valor por defecto: Y (sí).
- *primer_refresco*: fecha del primer refresco. Opcional, valor por defecto: 01/01/2000.

- *descripcion*: texto con la descripción de la entidad. Opcional.
- *planificacion_0 ... planificacion_9*: nombres de las planificaciones a las que se quiere asociar la entidad. Es obligatorio introducir al menos 1 de ellas.

Este procedimiento llama a **controlador.crear_entidad**, que se encargará de la realización del caso de uso:

1. Validar los valores de la entidad especificados (**modelo.validar_valores_entidad**).
2. Validar los nombres de planificaciones especificados (**modelo.validar_planificaciones**).
3. Almacenar la localización si es nueva (**modelo.nueva_localizacion**).
4. Crear la tabla de la entidad (**fachada.crear_tabla**).
5. Añadir la columna *fecha_refresco* a la tabla (**fachada.agregar_columna**).
6. Almacenar la nueva entidad (**modelo.nueva_entidad**).
7. Almacenar las relaciones entre la entidad y las planificaciones (**modelo.nueva_entidad_planificacion**).

Con respecto a la localización de la entidad, el Administrador debe especificar el nombre del esquema y de la base de datos de la instancia de SQL Server. Las localizaciones válidas (pares de nombres de esquema y base de datos) se irán almacenando en la tabla **modelo.localizacion**. No se permitirán localizaciones cuya base de datos sea MOTOR.

Para comprobar si la localización especificada es válida, el procedimiento **modelo.validar_valores_entidad** comprobará primero si ya está almacenada en la tabla. Si no es el caso, deberá llamar al procedimiento **fachada.existe_localizacion** para que realice la consulta que compruebe si el esquema y la base de datos existen dentro de la instancia.

Si la comprobación es correcta, **modelo.validar_valores_entidad** devolverá un bit, indicando si se debe almacenar la nueva localización, en cuyo caso *controlador*, después de superar las validaciones de valores y planificaciones, llamará al procedimiento **modelo.nueva_localizacion**.

También se debe comprobar si ya existe una tabla con el mismo nombre y en la misma localización. El procedimiento **fachada.existe_tabla** se encarga de construir esta consulta.

Para comprobar que la consulta sea válida, deberá llamar al procedimiento **fachada.ejecutar_subconsulta**, que la introducirá como subconsulta en una instrucción para que la ejecute *consultas*. Si no se produce ningún error, la consulta será válida.

El Administrador podrá especificar entre un mínimo de 1 y un máximo de 10 nombres de planificaciones. El procedimiento **controlador.crear_entidad** incluirá todos los nombres entre separadores en una cadena de texto, el cual enviará a **modelo.validar_planificaciones** para que los valide uno por uno.

Para crear la tabla de la entidad, el procedimiento **fachada.crear_tabla** deberá, en primer lugar, llamar a *consultas* para ejecutar la consulta de la entidad y depositar los resultados en una tabla auxiliar. En otra llamada a *consultas* extraerá las columnas y su tipo de datos de la tabla, con las que construirá la instrucción CREATE. Cuando no haya más columnas, llamará de nuevo a *consultas* para ejecutar la instrucción.

Con respecto al control de excepciones, hay varios casos de excepción que si ocurren interrumpirán la ejecución:

- El esquema o la base de datos no existen, o la base de datos es MOTOR.
- Ya existe otra entidad con el mismo nombre y localización.
- Ya existe una tabla con el mismo nombre en la localización especificada.
- La consulta no es válida.
- Alguno de los nombres de planificaciones no es válido.
- Se produce un error al crear la tabla.
- Se produce un error añadir la columna **fecha_refresco** a la tabla. En este caso, también se eliminaría la tabla creada en el paso anterior.

5.3.2. Crear planificación

El procedimiento **main.CrearPlanificacion** inicia el CU2 *Crear planificación*. El Administrador podrá indicar los valores de los atributos de la planificación enviándolos como parámetros:

- *nombre*: nombre identificativo de la planificación. Obligatorio.
- *periodicidad*: código de la periodicidad de la planificación. Obligatorio.
- *dia*: día de la planificación. Opcional, valor por defecto: -1.
- *frecuencia*: frecuencia de la planificación. Opcional, valor por defecto: -1.
- *maximo*: máximo de refrescos. Opcional, valor por defecto: 1.
- *fecha_ocasional*: fecha de refresco ocasional. Opcional, valor por defecto: '01/01/2000'.
- *descripcion*: descripción de la planificación. Opcional.

Este procedimiento llama a **controlador.crear_planificacion**, que se encargará de la realización del caso de uso:

1. Validar los valores especificados (**modelo.validar_valores_planificacion**).
2. Almacenar la nueva planificación (**modelo.nueva_planificacion**).

Las periodicidades disponibles estarán almacenadas en la tabla **modelo.periodicidad**. Para cada periodicidad almacenada, se incluirán los valores mínimos y máximos permitidos para el resto de atributos de la planificación. El procedimiento **modelo.validar_valores_planificacion** los contrastará para comprobar si los valores especificados por el Administrador son válidos.

Control de excepciones:

- Ya existe otra planificación con el mismo nombre.
- La periodicidad no es una de las de la tabla **periodicidad**.
- El día no está entre los valores permitidos.
- La frecuencia no está entre los valores permitidos.
- El máximo no está entre los valores permitidos.
- La fecha ocasional no está entre los valores permitidos.
- Ya existe otra regla con los mismos valores para todos los atributos, excepto el nombre.

5.3.3. Realizar refrescos

El procedimiento **main.RealizarRefrescos** inicia el CU3 *Realizar refrescos*. El Administrador pasa como parámetro *fecha_carga*, la fecha en la que realizar los refrescos.

Este procedimiento llama a **controlador.realizar_refrescos**, que se encargará de la realización del caso de uso:

1. Obtener las entidades que deben refrescarse para la fecha especificada (**modelo.seleccionar_entidades_refresco**).
2. Realizar el refresco de cada entidad (**controlador.aplicar_refresco**).
   a) Para cada planificación que establece un refresco en la fecha especificada, comprobar si se debe eliminar un refresco antiguo de la entidad y los datos de ese refresco de la tabla de la entidad (**modelo.comprobar_maximo_refrescos**). Si es el caso, eliminar los datos (**fachada.eliminar_datos**) y el refresco (**modelo.actualizar_refrescos**).
   b) Obtener y almacenar los datos del nuevo refresco (**fachada.insertar_datos**).
   c) Para cada planificación que establece un refresco en la fecha especificada, almacenar un nuevo refresco (**modelo.nuevo_refresco**).

Para obtener las entidades a refrescar se deberán consultar las tablas **modelo.entidad**, **modelo.planificacion** y **modelo.refresco**. Una entidad será seleccionada si se cumplen las siguientes condiciones:

- La entidad tiene el refresco activo y la fecha de primer refresco es más antigua o igual a la fecha de carga.

- Está asociada a una planificación que establece un refresco para la fecha de carga.
- No existe ningún refresco de la entidad por la planificación que establece el refresco en la fecha de carga (para no originar duplicados de refrescos).
- La fecha del refresco más antiguo existente de la entidad establecido por la planificación es también más antiguo que la fecha de carga en caso de se haya alcanzado el máximo de refrescos de la planificación (para no eliminar refrescos más recientes al efectuar refrescos más antiguos).

Las comprobaciones para averiguar si se cumple la segunda condición son diferentes según la periodicidad de la planificación:

- **Ocasional:** la fecha ocasional de la planificación coincide con la fecha de carga.
- **Diaria:** la diferencia en el número de días entre la fecha de carga y la fecha del último refresco de la entidad establecido por la planificación es un múltiplo de la frecuencia de la planificación.
- **Semanal:** el día de la semana de la fecha de carga es igual al valor de día de la planificación y la diferencia en el número de semanas entre la fecha de carga y la fecha del último refresco de la entidad establecido por la planificación es múltiplo de la frecuencia de la planificación.
- **Mensual:** en el caso del refresco mensual, las comprobaciones son más complejas debido al desigual número de días de los meses. Por ejemplo, cuando una planificación señala el refresco el día 31 de cada mes, este refresco se debería realizar el último día de todos los meses independientemente del número de días que contiene el mes. Se seleccionará una entidad si cumple al menos una de las condiciones siguientes:

  1. La diferencia en el número de meses entre la fecha de carga y la fecha del último refresco de la entidad establecido por la planificación es un múltiplo de la frecuencia de la planificación y el día del mes de la fecha de carga coincide con el día del mes de la planificación (situación normal).
  2. La diferencia en el número de meses entre la fecha de carga y la fecha del último refresco de la entidad establecido por la planificación es un múltiplo de la frecuencia de la planificación, la fecha de carga es el último día del mes y el día del mes de la planificación es el 31 (situación en la que el refresco está programado en día 31 y el mes actual tiene menos de 31 días).
  3. La diferencia en el número de meses entre la fecha de carga y la fecha del último refresco de la entidad establecido por la planificación es un múltiplo de la frecuencia de la planificación, la fecha de carga es el último día de febrero y el día del mes de la planificación es menor que 31 y mayor que 28 (situación en la que el refresco está programado en día 29 o 30 y el mes actual es febrero).

Cuando la entidad aún no ha sido refrescada por esa planificación, no se deberá comprobar si la diferencia en el número de días, semanas o meses del último refresco es múltiplo de la frecuencia para periodicidad diaria, semanal o mensual.

Es importante ver que una entidad puede tener más de un refresco programado por distintas planificaciones para la misma fecha, por lo que una entidad puede ser seleccionada varias veces para una misma fecha. Para diferenciar cuáles son las planificaciones que establecen el refresco en la misma fecha para la misma entidad, el procedimiento **modelo.seleccionar_entidades_refresco** devolverá tuplas (entidad, planificación), entre las cuales puede haber alguna entidad repetida.

Los refrescos se almacenarán en la tabla **modelo.refresco** a un nivel de detalle formado por la fecha, la entidad y la planificación. Los datos del refresco de una entidad sólo deben cargarse una vez en su tabla, independientemente del número de planificaciones asociadas a la entidad que establezcan un refresco en esa fecha.

Esto debe tenerse en cuenta al realizar el refresco de cada entidad, del cual se encarga el procedimiento **controlador.aplicar_refresco**. En primer lugar, deberá comprobar si hay que eliminar un refresco antiguo de la entidad, en caso de que se haya alcanzado el máximo de refrescos para la o las planificaciones que establecen un refresco en la fecha de carga.

Si se alcanza el máximo de refrescos, se debe eliminar el refresco más antiguo de la entidad establecido por la planificación. Sin embargo, puede existir otra planificación que no haya alcanzado el máximo y tenga también un refresco en la misma fecha y entidad. En ese caso, no hay que eliminar los datos de esa fecha de refresco en la tabla de la entidad.

El procedimiento **modelo.comprobar_maximo_refrescos** realiza a cabo estas comprobaciones para una entidad y planificación y devuelve:

- Un bit que indica si hay que eliminar un refresco.
- Un bit que indica si hay que eliminar los datos de la tabla con fecha igual a la de ese refresco.
- La fecha del refresco a eliminar.

El borrado y la inserción de los datos del refresco en la tabla de la entidad se efectúan como se ha explicado en la Sección 5.2. Por último, se debe almacenar un nuevo refresco para cada planificación. Todo este proceso se realiza para cada una de las entidades seleccionadas.

Aunque el uso esperado del motor es que el Administrador realice este caso de uso de forma diaria, justo después de la carga de los almacenes a los que apuntan las consultas de las entidades, pueden darse lugar otras situaciones:

- La fecha de carga corresponde a una fecha en la que ya se ha realizado anteriormente una carga. Esto puede ocurrir cuando la última carga terminó con errores o sin completarse y se repite para realizar el refresco de las entidades que faltan para esa fecha.

- La fecha de carga es más de un día superior a la fecha de la última carga. En este caso el sistema no ha efectuado la carga de datos para una o más fechas.
- La fecha de carga es anterior a la fecha de la última carga. Tiene lugar cuando no se ha realizado la carga en una fecha pero posteriormente se quieren cargar los datos faltantes producidos en esta fecha.

El diseño del caso de uso asegura que el motor funcionará correctamente y de la forma esperada en los tres casos.

Control de excepciones: si se produce alguna de las siguientes situaciones, el refresco de la entidad correspondiente se detiene pero se continúa con el refresco de la siguiente entidad:

- Se produce un error al borrar los datos de un refresco antiguo en la tabla de la entidad.
- Se produce un error al insertar los datos del nuevo refresco en la tabla de la entidad.

5.3.4. Borrar refrescos

El procedimiento **main.BorrarRefrescos** inicia el CU4 *Borrar refrescos*. Los parámetros que el Administrador especifica son:

- *nombre*: nombre de la entidad. Obligatorio.
- *bd*: nombre de la base de datos de la entidad. Obligatorio.
- *esquema*: nombre del esquema de la entidad. Obligatorio.
- *fecha_inicio*: fecha de inicio de borrado. Obligatorio.
- *fecha_fin*: fecha fin de borrado. Opcional, valor por defecto: *fecha_inicio*.

Este procedimiento llama a **controlador.borrar_refrescos**, que se encargará de la realización del caso de uso:

1. Comprobar que existe una entidad con los datos especificados (**modelo.comprobar_datos_entidad**) y que el rango de fechas es válido.
2. Para cada una de las fechas del rango, si existe un refresco de la entidad, eliminar los datos de la tabla de la entidad (**fachada.eliminar_datos**) y eliminar todos los refrescos almacenados en esa fecha y entidad (**modelo.actualizar_refrescos**).

Control de excepciones:

- No existe una entidad con el nombre y localización especificados.
- La fecha de inicio es más reciente que la fecha fin de borrado.
- Se produce un error al borrar los datos de un refresco de la tabla de la entidad.

5.3.5. Cambiar refresco activo

El procedimiento **main.CambiarRefrescoActivo** inicia el CU5 *Cambiar refresco activo*. Los parámetros que el Administrador especifica son:

- *nombre*: nombre de la entidad. Obligatorio.
- *bd*: nombre de la base de datos de la entidad. Obligatorio.
- *esquema*: nombre del esquema de la entidad. Obligatorio.
- *activo*: indica si hay que activar o desactivar el refresco. Obligatorio.

Este procedimiento llama a **controlador.cambiar_refresco_activo**, que se encargará de la realización del caso de uso:

1. Comprobar que existe una entidad con los datos especificados (**modelo.comprobar_datos_entidad**).
2. Actualizar el valor de la columna *activo* para la entidad almacenada (**modelo.actualizar_entidad**).

Control de excepciones:

- No existe una entidad con el nombre y localización especificados.

5.3.6. Borrar entidad

El procedimiento **main.BorrarEntidad** inicia el CU6 *Borrar entidad*. Parámetros:

- *nombre*: nombre de la entidad. Obligatorio.
- *bd*: nombre de la base de datos de la entidad. Obligatorio.
- *esquema*: nombre del esquema de la entidad. Obligatorio.

El procedimiento hace una llamada a **controlador.borrar_entidad**, que se encargará de la realización del caso de uso:

1. Comprobar que existe una entidad con los datos especificados (**modelo.comprobar_datos_entidad**).
2. Eliminar la tabla de la entidad (**fachada.eliminar_tabla**).
3. Eliminar los refrescos almacenados para esa entidad (**modelo.actualizar_refrescos**).
4. Eliminar la entidad (**modelo.actualizar_entidad**).

Para eliminar la tabla, se construye una instrucción DROP, la cual se enviará a *consultas* para ejecutarla.

Control de excepciones:

- No existe una entidad con el nombre y localización especificados.
- Se produce un error al eliminar la tabla.

5.3.7. Borrar planificación

El procedimiento **main.BorrarPlanificacion** inicia el CU7 *Borrar planificación*. Tiene el parámetro *nombre*, el nombre de la planificación.

El procedimiento llama a **controlador.borrar_planificacion**, que se encargará de la realización del caso de uso:

1. Comprobar que existe una planificación con ese nombre y no está asociada a ninguna entidad existente (**modelo.comprobar_datos_planificacion**).
2. Eliminar la planificación (**modelo.actualizar_planificacion**).

Control de excepciones:

- No existe una planificación con ese nombre.
- La planificación está asociada a alguna entidad existente.

5.4. Base de datos del motor

A continuación se resume la estructura de la base de datos en la que se implementa el motor, sus tablas y procedimientos almacenados.

5.4.1. Tablas

El diagrama del modelo relacional de la base de datos MOTOR se puede ver en la Figura 5.3. En el esquema *modelo* se encuentran las tablas:

- **entidad**: almacena las entidades creadas.
- **planificacion**: almacena las planificaciones de refresco creadas.
- **entidad_planificacion**: almacena las relaciones entre entidad y planificación.
- **refresco**: cada fila es un refresco realizado.
- **periodicidad**: almacena las posibles periodicidades y los valores máximos y mínimos que pueden tomar el resto de atributos de la planificación según la periodicidad.
- **localizacion**: almacena las posibles localizaciones (base de datos y esquema) de una entidad.

En *controlador*:

- **procedimiento**: cada fila es una ejecución de un procedimiento de *controlador*.
- **mensaje**: almacena los mensajes generados en cada ejecución.

Por último, la dimensión temporal **fecha** está en el esquema *dbo*.

Algunos detalles de diseño de las tablas son los siguientes:

- **Persistencia de entidades, planificaciones y refrescos:** las entidades, planificaciones y refrescos pueden borrarse al realizar los casos de uso *Borrar entidad*, *Borrar planificación* y *Borrar refrescos*. Para no perder información, se ha añadido a las tablas una columna bit ***existe*** que indica si existe (1) o se ha borrado (0).
- **Utilización de claves subrogadas:** las columnas cuyo nombre comienza por **id_**, que forman la clave primaria de la tabla, constituyen claves subrogadas. Las claves subrogadas son columnas autoincrementales de tipo entero que toman valores a partir del 1. En SQL Server estas columnas se llaman identidad (IDENTITY).

  Para la tabla **fecha**, sin embargo, **id_fecha** es una columna numérica que contiene la fecha en formato *yyyymmdd*. Este formato, además de identificar cada fecha de forma única, permite ordenar las fechas de forma creciente o decreciente y realizar comparaciones rápidamente.
- **Tratamiento de nulos:** todas las columnas de las tablas excepto las de descripción son obligatorias, es decir, no pueden tener valores nulos. Cuando se inserte una fila con un valor no informado o innecesario para una columna, se cargará el valor -1 si es entera o decimal, '-1' si es varchar, 0 si es bit y 20000101 si hace referencia a una fecha.
- **Cumplimiento de la integridad referencial:** la política ante el borrado de filas de una tabla referenciadas por filas en otras tablas por una clave foránea es NO ACTION. Esta política directamente no permite el borrado de filas referenciadas.

Los valores iniciales de **modelo.periodicidad** se pueden ver en la Tabla 5.1. La tabla **dbo.fechas** se cargará con una fila por cada fecha desde el 01/01/2000 al 31/12/2030.

nombre	**codigo**	**min/max dia**	**min/max frecuencia**	**min/max maximo**	**min/max fecha**
'Diaria'	'D'	NA(-1)/NA(-1)	1/365	1/999999	NA(20000101)/NA(20000101)
'Semanal'	'S'	1/7	1/52	1/999999	NA(20000101)/NA(20000101)
'Mensual'	'M'	1/31	1/12	1/999999	NA(20000101)/NA(20000101)
'Ocasional'	'O'	NA(-1)/NA(-1)	NA(-1)/NA(-1)	1/1	20000101/20303131

Tabla 5.1.: Filas de la tabla modelo.periodicidad

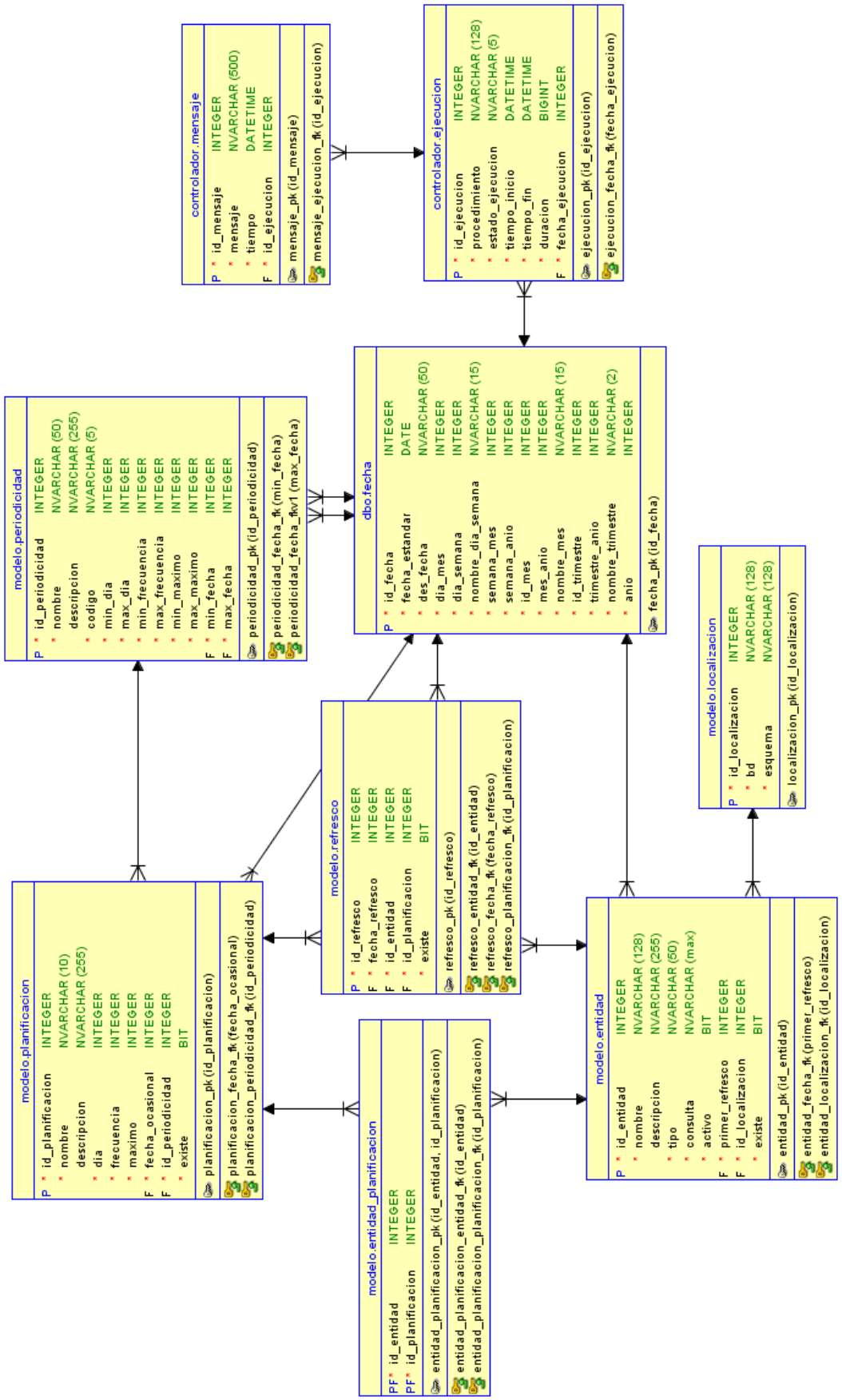

Figura 5.3.: Modelo relacional de la base de datos MOTOR

5.4.2. Procedimientos almacenados

5.4.2.1. Esquema *main*

Contiene los procedimientos almacenados que dan inicio a los casos de uso. Estos son:

- **CrearEntidad**.
- **CrearPlanificacion**.
- **RealizarRefrescos**.
- **BorrarRefrescos**.
- **CambiarRefrescoActivo**.
- **BorrarEntidad**.
- **BorrarPlanificacion**.

5.4.2.2. Esquema *controlador*

El esquema *controlador* contiene un procedimiento por cada caso de uso, que lleva a cabo la realización del mismo:

- **crear_entidad**.
- **crear_planificacion**.
- **realizar_refrescos**.
- **borrar_refrescos**.
- **cambiar_refresco_activo**.
- **borrar_entidad**.
- **borrar_planificacion**.

Además, contiene:

- **nueva_ejecucion:** inserta una fila en controlador.ejecucion para almacenar una nueva ejecución con los valores de columna dados.
- **actualizar_ejecucion:** actualiza el estado, duración y fecha fin de una fila de controlador.ejecucion dado su id.
- **nuevo_mensaje:** inserta una fila en controlador.mensaje para almacenar un nuevo mensaje con los valores de columnas dados.
- **aplicar_refresco:** realiza el refresco de una entidad dado su id para una fecha de carga.

5.4.2.3. Esquema *modelo*

Con respecto a las entidades:

- **validar_valores_entidad:** comprueba que los valores de los atributos pasados por parámetro son válidos para una entidad.
- **comprobar_datos_entidad:** comprueba que existe una entidad con el nombre y localización proporcionados.
- **nueva_localizacion:** inserta una fila en modelo.localizacion para almacenar una nueva localización con los valores de esquema y base de datos proporcionados.
- **nueva_entidad:** inserta una fila en modelo.entidad para almacenar una nueva entidad con los valores de las columnas dados.
- **nueva_entidad_planificacion:** inserta filas en modelo.entidad_planificacion para una entidad y varias planificaciones.
- **obtener_datos_entidad:** devuelve el nombre, base de datos, esquema y consulta de una entidad dado su id.
- **actualizar_entidad**: actualiza las columnas *existe* y *activo* de la fila donde se almacena la entidad con nombre, esquema y base de datos dados.
- **seleccionar_entidades_refresco:** devuelve un conjunto de filas con tuplas (id entidad, id planificación) con los id de las entidades a refrescar en la fecha de carga proporcionada y los id de las planificaciones que establecen ese refresco.

Con respecto a las planificaciones:

- **validar_planificaciones:** comprueba que los nombres de las planificaciones, pasadas en una cadena de texto y separadas por ';', son válidos.
- **validar_valores_planificacion:** comprueba que los valores de los atributos de una planificación pasados por parámetro son válidos.
- **nueva_planificacion:** inserta una fila en modelo.planificacion para almacenar una nueva planificación con los valores de las columnas dados.
- **comprobar_datos_planificacion:** comprueba que existe una planificación con el nombre proporcionado y que no está asociada a ninguna entidad.
- **actualizar_planificacion:** actualiza la columna *existe* de la fila donde se almacena la planificación indicada por su nombre.

Con respecto a los refrescos:

- **comprobar_maximo_refrescos:** devuelve, dados los id de una entidad y una planificación, si hay que eliminar el refresco más antiguo establecido por la planificación en la

entidad, en cuyo caso también devuelve la fecha de ese refresco y si no hay más refrescos en esa fecha para la misma entidad.

- **nuevo_refresco:** inserta una fila en modelo.refresco para almacenar un nuevo refresco con los valores de las columnas dados.
- **actualizar_refrescos:** dependiendo de los id de la entidad y planificación dados y la fecha de refresco, actualiza el valor de la columna *existe* de una o varias filas de modelo.refresco.

5.4.2.4. Esquema *fachada*

- **existe_tabla:** construye la consulta que comprueba si ya existe en el servidor una tabla con nombre y en la localización dados.
- **existe_localizacion:** construye la consulta que comprueba si existe en el servidor el esquema y base de datos proporcionados.
- **preparar_consulta:** sustituye la cadena '$FECHA_CARGA' por la fecha de carga actual en formato numérico.
- **ejecutar_subconsulta:** construye la instrucción que comprueba si una consulta dada es válida como subconsulta.
- **crear_tabla:** construye la instrucción CREATE de una tabla con las columnas y tipo de datos iguales a las columnas devueltas por una consulta dada, con nombre y en la localización proporcionados.
- **agregar_columna:** construye la instrucción ALTER que añade la columna *fecha_refresco* con el tipo *int* en la tabla dada.
- **eliminar_tabla:** construye la instrucción DROP que elimina la tabla con nombre y en la localización proporcionados.
- **insertar_datos:** construye la instrucción INSERT que inserta los datos de un refresco sobre una tabla proporcionando nombre, localización, consulta que devuelve los datos y fecha del refresco.
- **eliminar_datos:** construye la instrucción DELETE que elimina los datos de un refresco de una tabla proporcionando nombre, localización y fecha del refresco.

5.4.2.5. Esquema *consultas*

- **resultado_int:** ejecuta las consultas SELECT que devuelven un resultado de tipo *int*.
- **sin_resultado:** ejecuta las instrucciones que modifican datos u objetos del servidor y no devuelven resultado (CREATE, ALTER, DROP, etc).

5.4.2.6. Esquema *dbo*

El esquema *dbo* no contiene procedimientos almacenados, sino una serie de funciones de uso común para el resto de procedimientos de los otros esquemas.

5.5. Modelo específico de la plataforma

El archivo **PSM.sql** contiene el PSM, formado por el modelo físico de la base de datos MOTOR específico para SQL Server, que contiene las instrucciones CREATE de los esquemas, las tablas y los procedimientos almacenados. De estos últimos se incluye su declaración, pero no su implementación. Su contenido se puede ver en el Anexo A.

Este fichero puede ser importado como archivo DDL por las herramientas de modelado, para poder generar el modelo relacional de la base de datos y más modelos físicos característicos de otras plataformas.

6. Implementación y pruebas

En este capítulo se explican los detalles de implementación del motor y las pruebas realizadas para validar el funcionamiento del mismo. También se describirá resumidamente la implementación de la plataforma de visualización de la información.

6.1. Implementación del motor

6.1.1. Codificación

Se seguirá la siguiente convención para la codificación en T-SQL:

- Palabras clave de T-SQL y bases de datos en MAYÚSCULA.
- Objetos de SQL Server (esquemas, tablas, columnas, índices, procedimientos, funciones) excepto bases de datos en minúscula.
- Parámetros y variables de procedimientos y operadores en minúscula.
- Utilizar snake_case para los nombres de objetos y variables sin incluir caracteres especiales.

Con respecto a las variables y parámetros de T-SQL, para favorecer la claridad y facilitar el desarrollo del código, se utilizará una nomenclatura concreta compuesta de un prefijo que varía según el tipo de datos que contiene:

- Numérico (int, decimal): *nm_*.
- Cadena de texto (varchar, nvarchar): *st_*.
- Fecha y hora (date, datetime): *dt_*.
- Booleano (bit): *bl_*.
- Cadena de texto que corresponde a una consulta SQL generada dinámicamente: *sq_*.
- Cursores: *cr_*.

Por último, los nombres de las funciones de T-SQL incorporarán el prefijo **fn_** para diferenciarlas de los procedimientos.

6.1.2. Control de entrada y salida

Todos los parámetros de entrada de los procedimientos del esquema *main* serán del tipo *nvarchar*, para asegurar que no se producen errores de tipado al procesar los valores introducidos por el Administrador. Estos valores se comprobarán y convertirán al tipo correcto al iniciarse el procedimiento correspondiente de *controlador*. De ello se encargarán los procedimientos **controlador.int_valido**, **controlador.bit_valido** y **controlador.date_valido**.

Si no se especifica un valor para los parámetros o se pasa el valor *null*, se cambiarán a sus valores por defecto o a una cadena vacía antes de pasárselos al procedimiento de *controlador*. En el caso de los parámetros indicadores, el valor 1/0 se indicará con los caracteres 'Y'/'N' respectivamente.

Para proporcionar información durante la realización del caso de uso, todos los mensajes generados en la ejecución se imprimirán también por la salida de SSMS con la instrucción RAISERROR[1].

6.1.3. Parámetros de salida de procedimientos

Muchos de los procedimientos explicados en el capítulo de diseño necesitan devolver uno o varios valores al procedimiento que los invocó. Para ello, se han utilizado **parámetros OUTPUT**. Los parámetros OUTPUT son variables declaradas en el procedimiento invocador, que se envían como parámetros en la llamada del procedimiento invocado junto con la palabra OUTPUT.

Al declarar los parámetros de entrada del procedimiento invocado también deben declararse estos parámetros, especificando con la palabra OUTPUT que se tratan de parámetros de salida. Todos los cambios sufridos en el parámetro dentro del procedimiento invocado se incluirán en la variable del procedimiento invocador una vez que el invocado finalice.

6.1.4. Funciones y constantes

Las funciones realizan un conjunto de instrucciones sencillas, y siempre devuelven un valor de un tipo que se indica al declarar la función. Tienen restricciones con respecto a las instrucciones y cláusulas que se pueden utilizar, como EXEC o OUTPUT. En este caso, se han utilizado funciones escalares para tareas recurrentes, como el tratamiento de valores de fechas y su conversión entre tipos *int* y *date* en el esquema *dbo*.

El esquema *dbo* también contiene todas las constantes utilizadas por los otros esquemas. Como no se pueden definir variables globales en T-SQL, se han utilizado funciones escalares sin parámetros de entrada que sólo incluyen una instrucción RETURN con el valor de la constante.

[1]RAISERROR (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/language-elements/raiserror-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: mayo de 2020

6.1.5. Ejecución de consultas generadas dinámicamente

Las consultas e instrucciones T-SQL construidas por los procedimientos del esquema *fachada* son generadas dinámicamente e introducidas dentro de una variable de tipo *nvarchar* para poder ser enviadas a *consultas* en las llamadas a sus procedimientos desde *fachada*.

Para ejecutar las instrucciones T-SQL se ha utilizado el componente **sp_executesql**[2], contenido dentro del esquema *sys* de todas las bases de datos. En el caso de las instrucciones que no devuelven resultado (procedimiento **consultas.sin_resultado**) el componente se usa de la siguiente forma, donde *sq_consulta* es la variable *nvarchar* que contiene la instrucción enviada por *fachada*.

```
EXEC sys.sp_executesql @sq_consulta
```

Por otro lado, hay que ejecutar consultas que devuelven un resultado, el cual debe recogerse y enviarse de vuelta al procedimiento de *fachada*. En nuestro caso, sólo se necesita realizar consultas que devuelven el resultado de un COUNT. Para ello, el procedimiento de *fachada* debe:

- Construir la consulta de forma que el resultado del COUNT de la cláusula SELECT se incluyan en una variable:

```
DECLARE @sq_select nvarchar(500)
SET @sq_select = 'SELECT @nmOUT = COUNT(*) FROM ...'
```

- Enviar un parámetro *nvarchar* que contiene la especificación de la variable incluida en la consulta para depositar el valor devuelto:

```
DECLARE @sq_param nvarchar(100)
SET @sq_param = '@nmOUT int OUTPUT'
```

- Enviar el parámetro de tipo *int* donde se almacena el valor de la variable incluida en la consulta:

```
DECLARE @nm_sq_out int
```

La utilización del componente para las consultas es, entonces:

```
EXEC sys.sp_executesql @sq_consulta, @sq_param, @nm_sq_out OUTPUT
```

Como *nm_sq_out* contiene el valor numérico del COUNT, éste debe especificarse como OUTPUT en la llamada a **consultas.resultado_int** desde *fachada*.

[2] sp_executesql (Transact-SQL) https://docs.microsoft.com/es-es/sql/relational-databases/system-stored-procedures/sp-executesql-transact-sql?view=sql-server-ver15, Microsoft Docs. Última visita: mayo de 2020.

6.1.6. Tratamiento de errores y excepciones

Las condiciones de error se producen cuando alguna de las entradas especificadas por el Administrador no es correcta. Para capturarlas, todos los procedimientos invocados desde *controlador* tendrán como parámetros de salida *bl_error* y *st_mensaje*. El primero, de tipo *bit*, indicará si hay un error, y el segundo, de tipo *nvarchar*, una descripción del error.

Si se comprueba que hay un error, el procedimiento modificará *bl_error* y *st_mensaje* y devolverá la llamada inmediatamente con la instrucción RETURN. El procedimiento invocador del esquema *controlador* almacenará el mensaje (**controlador.nuevo_mensaje**) asociado al id de la ejecución correspondiente, contenido dentro de una variable cuyo valor es devuelto como parámetro de salida del procedimiento **controlador.nueva_ejecucion**.

Después de almacenar el mensaje, se realiza la instrucción GOTO *Completada*. La etiqueta *Completada* estará al final del procedimiento y llamará a **controlador.actualizar_ejecucion**.

Al ejecutar las consultas e instrucciones desde los procedimientos del esquema *consultas*, pueden ocurrir situaciones de excepción, ya que no se posee control sobre las bases de datos exteriores a MOTOR, que darían lugar a la interrupción del caso de uso. Por ello, la ejecución del componente *sp_executesql* se ha incluido dentro de un bloque TRY-CATCH. Si se produce una excepción, el bloque la capturará, y recogerá el mensaje de error (ERROR_MESSAGE) en una variable que envía al procedimiento invocador como parámetro de salida.

6.1.7. Tablas temporales y cursores

Para realizar algunas operaciones se han utilizado **tablas temporales**[3], como por ejemplo para depositar los id de las entidades y planificaciones seleccionadas en **modelo.seleccionar_entidades_refrescar** o las columnas y su tipo para crear la tabla de la entidad en **fachada.crear_tabla**. Las tablas temporales, identificadas con el prefijo '#', pueden ser usadas por los procedimientos invocados desde el procedimiento que la creó, y se eliminan cuando éste termina.

Junto con las tablas temporales se han utilizado **cursores**[4], objetos que sirven para recorrer todas las filas de la tabla y almacenar su contenido en variables. De esta forma, se pueden utilizar los valores de esas variables en varias operaciones. Se ha utilizado la función @@FETCH_STATUS para controlar el final de lectura cuando no hay más filas.

[3]Tablas `https://docs.microsoft.com/es-es/sql/relational-databases/tables/tables?view=sql-server-ver15`, Microsoft Docs. Última visita: abril de 2020.

[4]Cursores (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/language-elements/cursors-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: mayo de 2020.

6.1.8. Otros detalles de implementación

- Se han aplicado las instrucciones SET ANSI_NULLS ON[5] y SET QUOTED_IDENTIFIER ON[6] para que T-SQL siga el comportamiento ISO con respecto al tratamiento de nulos y de las comillas.
- Antes de la instrucción CREATE de cada objeto se incluye una instrucción DROP para eliminar el objeto en caso de que ya existiera en la base de datos.
- Al principio del cuerpo de cada procedimiento se ha incluido la instrucción SET NOCOUNT ON[7] para evitar que devuelvan otros mensajes tras su ejecución.

6.2. Pruebas

Para comprobar el funcionamiento de cada procedimiento implementado se han realizado una serie de **pruebas unitarias**. En los procedimientos del esquema *controlador* que llevan a cabo la funcionalidad de cada caso de uso se han hecho un mayor número de pruebas, para validar que el flujo es correcto en los casos normales y que los casos de excepción son capturados adecuadamente.

Los procedimientos **controlador.crear_entidad** y **controlador.crear_planificación** han sido más costosos de validar, debido a la cantidad de parámetros de entrada y la necesidad de probar diferentes combinaciones de valores para ellos. Además, por su dificultad se ha puesto especial énfasis en probar que el procedimiento **fachada.crear_tabla** construya de forma adecuada la instrucción CREATE de la tabla a partir de diferentes consultas.

Una vez superadas estas pruebas, se han realizado un conjunto de **pruebas de integración** con las que aseguraremos el correcto funcionamiento del sistema en su totalidad. Para ello, se ha construido en SQL Server un entorno de pruebas constituido una base de datos donde se generarán las tablas del modelo (EXPL_DEV) y la base de datos del motor (MOTOR_DEV). Las consultas de las entidades apuntarán a una base de datos donde se almacena la información del negocio (BASE), cuyo diseño se explica en el Anexo B.

Por el gran número de pruebas unitarias efectuadas, especificarlas ocuparía mucho espacio en la documentación, así que a continuación sólo se listarán las pruebas de integración.

[5]SET ANSI_NULLS (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/statements/set-ansi-nulls-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: junio de 2020.

[6]SET QUOTED_IDENTIFIER (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/statements/set-quoted-identifier-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: junio de 2020.

[7]SET NOCOUNT (Transact-SQL) `https://docs.microsoft.com/es-es/sql/t-sql/statements/set-nocount-transact-sql?view=sql-server-ver15`, Microsoft Docs. Última visita: mayo de 2020.

P1	**Crear planificaciones**
Propósito	Comprobar que se crean varias planificaciones correctamente
Precondiciones	-
Salida esperada	Las nuevas planificaciones son almacenadas con los valores especificados
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.1.: Descripción de la prueba de integración P1

P2	**Crear entidad con nueva localización**
Propósito	Comprobar que se crea una entidad asociada a varias planificaciones con una localización que no está almacenada
Precondiciones	Las planificaciones existen
Salida esperada	La nueva localización es almacenada, se crea la tabla de la entidad y se almacena la nueva entidad con los valores especificados
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.2.: Descripción de la prueba de integración P2

P3	**Realizar refrescos para fechas consecutivas**
Propósito	Comprobar se realiza correctamente el refresco de una entidad en varias fechas consecutivas
Precondiciones	La entidad existe
Salida esperada	Se refresca la entidad si así lo establecen las planificaciones a las que se ha asociado para la fecha de carga, agregando los datos del nuevo refresco a la tabla (sustituyendo de forma correcta $FECHA_CARGA) y creando el refresco. Se eliminan los refrescos antiguos y los datos de la tabla de la entidad cuando se alcanza el máximo.
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.3.: Descripción de la prueba de integración P3

P4	**Realizar refrescos para fechas no consecutivas**
Propósito	Comprobar que se realiza correctamente el refresco de una entidad en varias fechas alternas
Precondiciones	La entidad existe
Salida esperada	Igual que P4
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.4.: Descripción de la prueba de integración P4

P5	**Crear entidad con localización almacenada**
Propósito	Comprobar que se crea una entidad asociada a varias planificaciones con una localización que ya está almacenada
Precondiciones	Ya existe una entidad con la misma localización
Salida esperada	Se crea la tabla de la entidad y se almacena la nueva entidad con los valores especificados
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.5.: Descripción de la prueba de integración P5

P6	**Realizar refrescos para fechas repetidas**
Propósito	Comprobar que se refresca una entidad recién creada pero no se refresca otra entidad que ya se ha refrescado en las mismas fechas
Precondiciones	Las dos entidades existen, una de ellas asociada a refrescos almacenados
Salida esperada	Se refresca sólo la entidad no refrescada anteriormente en las fechas de carga
Salida obtenida	También se ha refrescado la entidad ya refrescada anteriormente en las fechas de carga cuyo refresco ya no existía, eliminando refrescos recientes y creando refrescos más antiguos.
Acción	Se ha añadido otra comprobación en **modelo.seleccionar_entidades_refrescar** para que sólo se seleccionen entidades cuyo refresco más antiguo por la planificación sea menor a la fecha de carga en caso de que se haya alcanzado el máximo de refrescos para esa planificación

Tabla 6.6.: Descripción de la prueba de integración P6

P7	**Eliminar refrescos más recientes de una entidad**
Propósito	Comprobar que se eliminan los refrescos de una entidad para un rango de fechas
Precondiciones	La entidad existe y está asociada a refrescos existentes en el rango de fechas
Salida esperada	Se eliminan los refrescos y los datos correspondientes de la tabla de la entidad en las fechas especificadas
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.7.: Descripción de la prueba de integración P7

P8	**Realizar refrescos en las fechas que se han eliminado**
Propósito	Comprobar que se vuelven a realizar los refrescos en las fechas en las que se habían eliminado
Precondiciones	La entidad existe y está asociada a refrescos almacenados pero que no existen en las fechas especificadas
Salida esperada	Se vuelven a realizar los refrescos de la entidad en la que habían sido eliminados
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.8.: Descripción de la prueba de integración P8

P9	**Eliminar entidad**
Propósito	Comprobar que se elimina una entidad anteriormente refrescada
Precondiciones	La entidad existe y está asociada a refrescos existentes
Salida esperada	Se elimina la tabla de la entidad, todos sus refrescos y la entidad
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.9.: Descripción de la prueba de integración P9

P10	**Eliminar planificación que estaba asociada a la entidad eliminada**
Propósito	Comprobar que se elimina una planificación que está asociada a una entidad que ya no existe
Precondiciones	La planificación existe y está asociada a una entidad almacenada pero que no existe
Salida esperada	Se elimina la planificación
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.10.: Descripción de la prueba de integración P10

P11	**Realizar refrescos después de eliminar una entidad**
Propósito	Comprobar que se realizan refrescos sólo para la entidad que existe
Precondiciones	Las dos entidades están almacenadas, una existe y la otra no
Salida esperada	Se realizan los refrescos sólo de la entidad existente
Salida obtenida	Se ha obtenido la salida esperada

Tabla 6.11.: Descripción de la prueba de integración P11

6.3. Código

La carpeta *Codigo* contiene los ficheros donde se encuentra el código fuente del motor en T-SQL. Está dividido en tres subcarpetas según las capas:

- **Logica:** *main.sql*, *controlador.sql* y *modelo.sql*.
- **Externa:** *fachada.sql* y *consultas.sql*.
- **Comun:** *dbo.sql*.

6.4. Implementación de la plataforma de visualización

Una vez obtenido el código depurado del motor, se puede pasar a construir la plataforma de visualización de la información en un informe de Power BI. El primer paso en la construcción del informe es la implementación del modelo de datos tabular, el cual necesita de un origen de datos.

Por ello, se han creado las planificaciones y entidades para implementar el modelo de datos que tomará como origen el informe. Las entidades son:

- **productos:** dimensión desnormalizada de los productos. Se refresca incrementalmente según la fecha de salida del producto de forma diaria.
- **ventas_producto_tiendas:** almacena las métricas de unidades vendidas e ingresos de los pedidos a nivel de detalle de tienda, producto y fecha. Se refresca incrementalmente según la fecha del pedido de forma diaria.
- **stock:** almacena fotos de stock para el día 15 y el último día de cada mes durante todo el año, y los domingos de las últimas 4 semanas.
- **tiendas:** foto de la dimensión de tiendas. Se refresca en 1 de enero.
- **tiempo:** foto de la dimensión temporal desde la fecha 01/01/2020. Se refresca el 1 de enero.

La conexión a la instancia de SQL Server se ha configurado para que sea dinámica mediante la utilización de dos parámetros para el nombre de la instancia y el nombre de la base de datos. De esta forma, se puede cambiar fácilmente el origen de datos sin necesidad de realizar modificaciones en el modelo.

Una vez establecida la conexión se ha creado el modelo tabular en modo de importación de datos[8]. El modelo se puede ver en la Figura 6.1.

Las tareas de modelado que se han realizado son:

- Establecer relaciones entre las tablas del modelo.

[8]El modo de importación genera una copia de los datos de origen (página 35).

- Eliminar, modificar el tipo de datos o cambiar nombre de columnas.
- Cambiar ordenaciones de valores de columnas.
- Desactivar la actualización de los datos de tiempo y tiendas.

Además, se han creado los siguientes objetos multidimensionales:

- Una jerarquía personalizada de la dimensión temporal en la tabla de tiempo.
- Una columna calculada que indica la criticidad del stock en texto (*Falta de stock*).
- Una medida que devuelva las unidades de stock de la última fecha para todos los productos (*Último stock*).

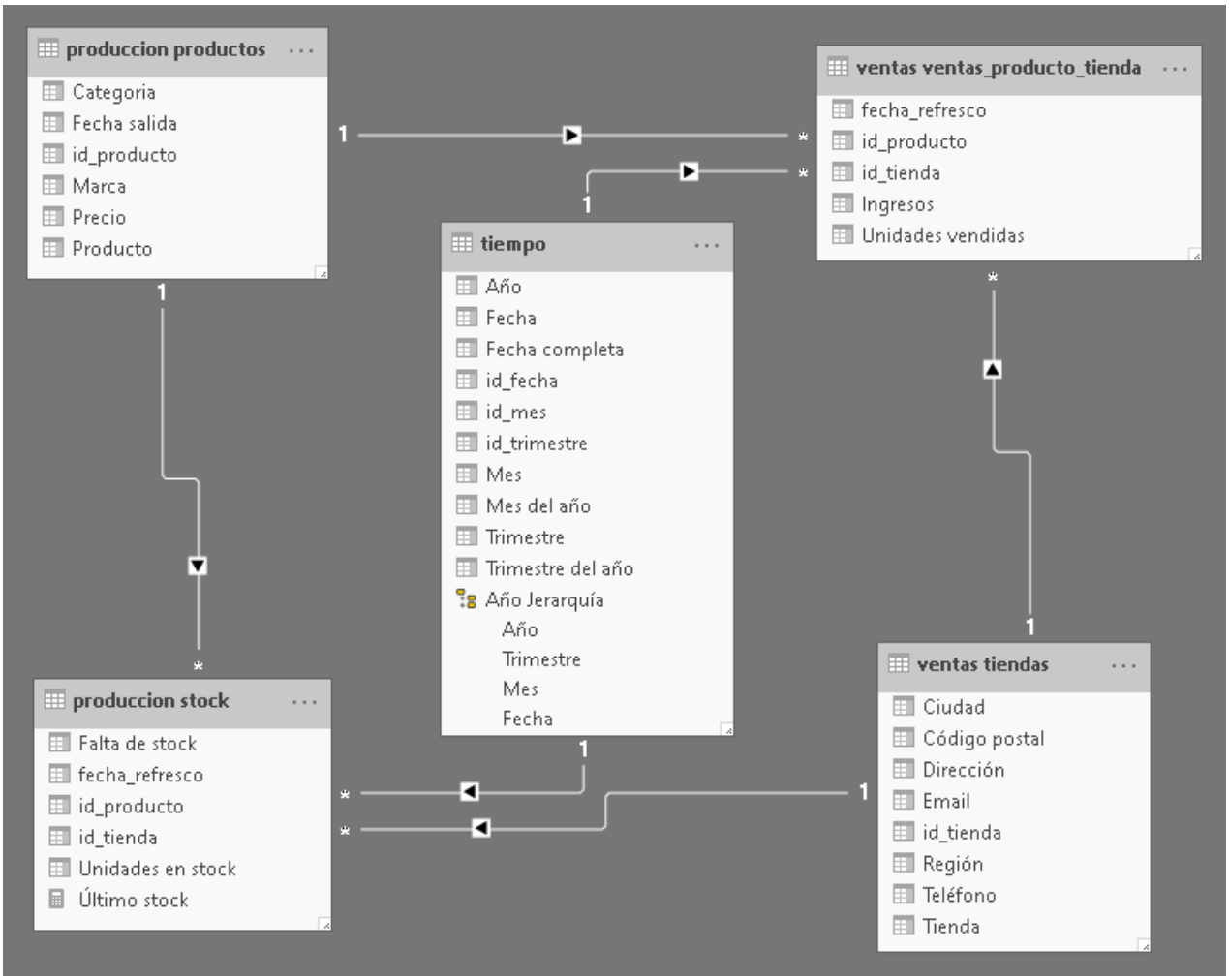

Figura 6.1.: Modelo tabular implementado en Power BI

El informe de Power BI se compone de dos páginas: **Ventas**, que muestra la información de los ingresos y las unidades vendidas en cada tienda, y **Stock**, que muestra la información del stock por tienda. Ambas páginas contienen objetos visuales que muestran diferentes indicadores de las

métricas para la tienda seleccionada. Todos ellos son interactivos y reaccionan ante la selección realizada en otros objetos visuales.

Algunas de las exploraciones que se pueden hacer en la primera página son (Figura 6.2):

- Elegir la tienda de la que visualizar los datos.
- Navegar entre la jerarquía temporal para visualizar los datos con mayor o menor detalle, tanto de toda la serie temporal como de un punto en el tiempo concreto.
- Visualizar la información de una marca y categoría en concreto.

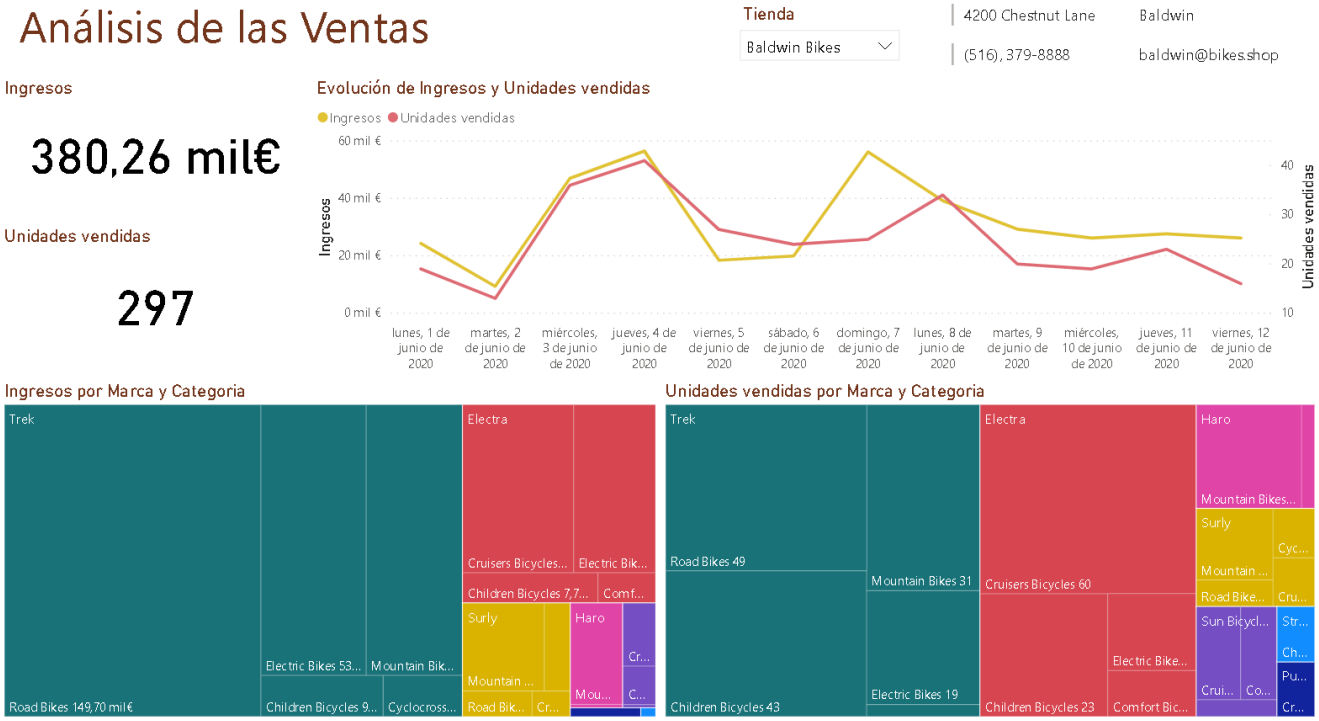

Figura 6.2.: Página Ventas del informe de Power BI

Con respecto a la página de Stock (Figura 6.3):

- Elegir la tienda de la que visualizar los datos.
- Visualizar la información de un producto concreto.
- Visualizar la información en una fecha concreta.

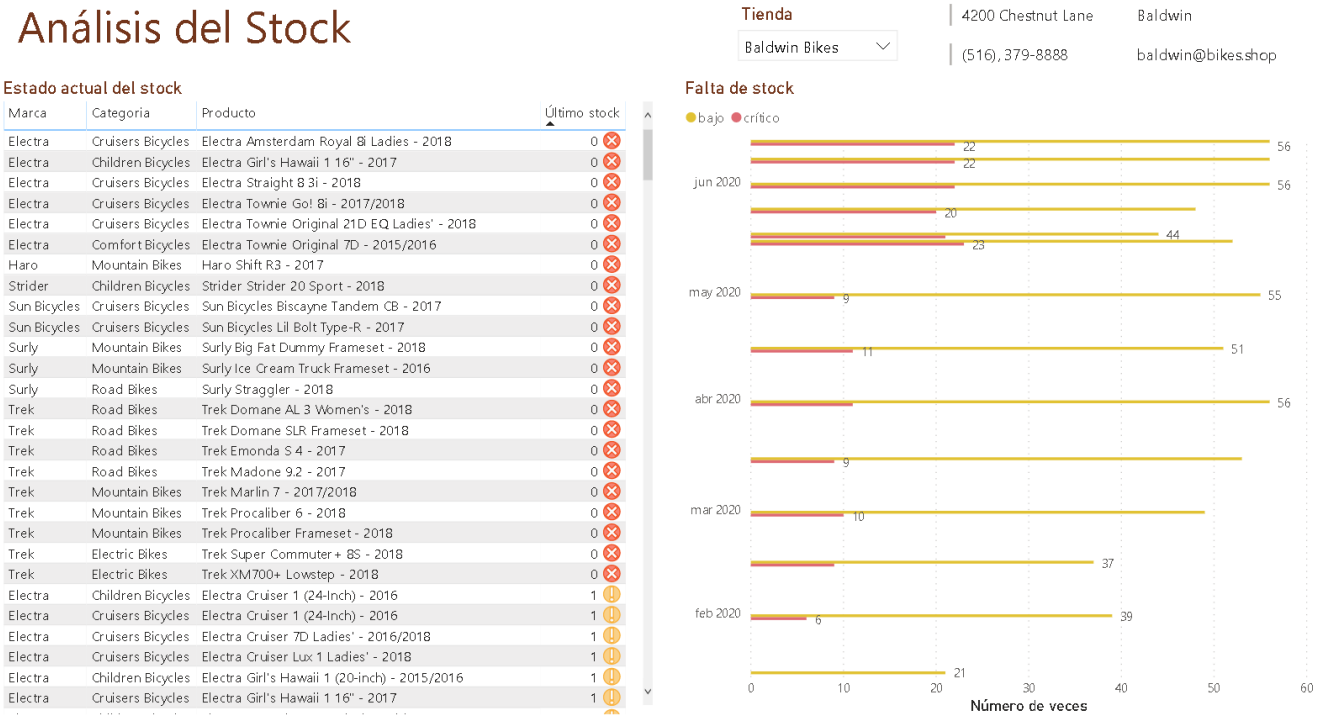

Figura 6.3.: Página Stock del informe de Power BI

7. Despliegue y manual de usuario

En este capítulo se proporcionan las instrucciones para desplegar el motor en un servidor de SQL Server y su manual de usuario. También se describirá el despliegue del sistema de información del caso de negocio descrito en el capítulo 2.

7.1. Despliegue del motor

Se ha creado un fichero *batch* llamado *despliegue.bat* y ubicado dentro del directorio *Codigo*, que realiza el despliegue del motor en una instancia local de SQL Server ejecutando en el orden correcto los scripts de T-SQL que implementan el motor con la utilidad **sqlcmd**[1]. El nombre de la instancia de SQL Server en el que realizar el despliegue se envía como argumento.

Los requisitos de software que deben cumplirse para completar el despliegue son los siguientes:

- Instancia de SQL Server 2017 o superior.
- Utilidad sqlcmd 13.x o 15.x.
- Controlador ODBC para SQL Server 13 (si la versión de sqlcmd es 13.x) o 17 (si la versión de sqlcmd es 15.x).

Para ejecutar correctamente el fichero, hay que estar dentro del directorio *Codigo*. El argumento puede ser de la forma *maquina**instancia*, .*instancia* (despliega en la instancia indicada de la máquina) o *maquina* (despliega en la instancia predeterminada de la máquina).

Precondiciones:

- El servicio de motor de bases de datos de SQL Server está operativo.
- El usuario con *login* con las credenciales de Windows está habilitado y asociado al rol *sysadmin*.
- El script se ejecuta localmente a la instancia y el protocolo de conexión por memoria compartida está habilitado.

[1] Utilidad sqlcmd `https://docs.microsoft.com/es-es/sql/tools/sqlcmd-utility?view=sql-server-ver15`, Microsoft Docs. Última visita: mayo de 2020

Si el argumento es incorrecto o no se puede conectar con la instancia de SQL Server especificada, se notificará por la salida del error. Si se produce un error al ejecutar cualquiera de los scripts T-SQL, se notificará del error y a continuación se desharán los cambios efectuados.

7.2. Manual de usuario del motor

Una vez desplegado el motor, aparecerá la base de datos MOTOR en la lista de bases de datos del menú de navegación de SSMS. Para iniciar alguno de los casos de uso, hay que abrir MOTOR >Programabilidad (*Programmability*) >Procedimientos almacenados (*Stored Procedures*).

Es conveniente filtrar los procedimientos para que aparezcan sólo los que interesan del esquema *main*. Para ello, se pulsa con el botón derecho el directorio de procedimientos almacenados >Filtro (*Filter*) >Configuración de filtro (*Filter settings*) como se ve en la Figura 7.1.

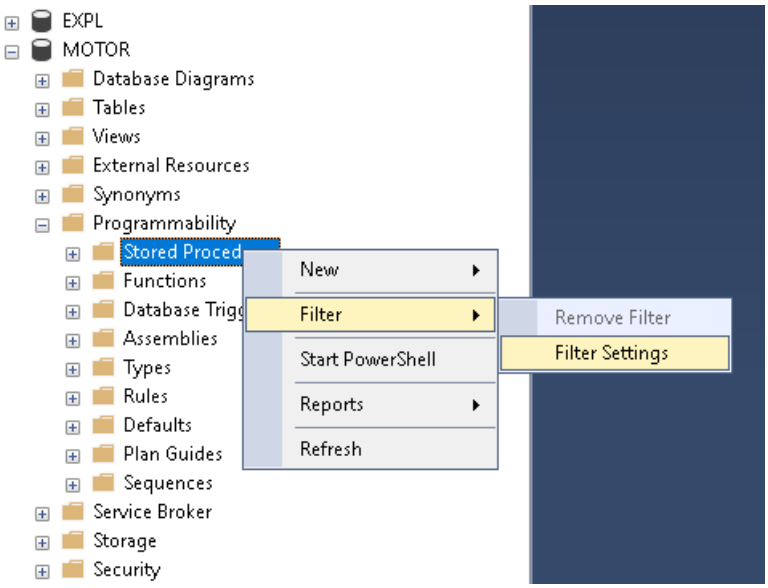

Figura 7.1.: Manual de usuario: Configuración de filtro

En la ventana que aparece, se escribe 'main' en el criterio Esquema (*Scheme*) Contiene (*Contains*) como se puede ver en la Figura 7.2. A continuación se pulsa *OK*.

Ahora, abriendo la carpeta de los procedimientos almacenados, aparecerán los que inician los casos de uso. Para realizar cualquiera, se pulsa botón derecho sobre éste >Ejecutar Procedimiento Almacenado (*Execute Stored Procedure*), como en la Figura 7.3.

Aparecerá una ventana para introducir los parámetros. Para ello, hay que tener en cuenta lo siguiente:

- Los valores si/no (como en el parámetro activo al crear una entidad) se deben introducir con los caracteres Y/N.

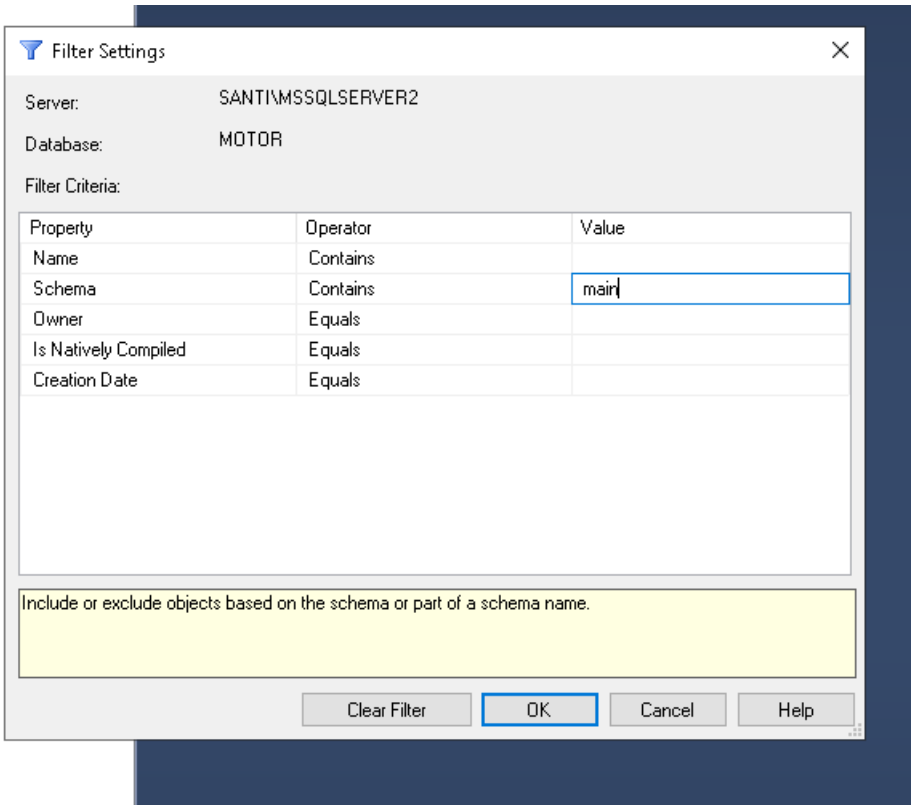

Figura 7.2.: Manual de usuario: Aplicar filtro

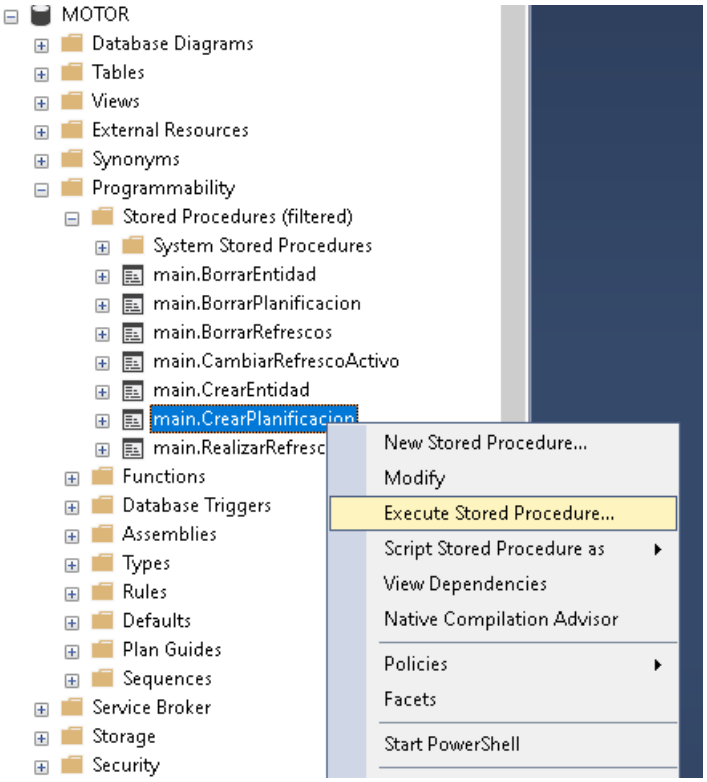

Figura 7.3.: Manual de usuario: Ejecutar procedimiento almacenado

- Las fechas deben introducirse con cualquier formato de fechas válido, como por ejemplo dd/mm/yyyy, d/m/yy, dd.mm.yyyy, d.m.yy, dd-mm-yyyy, d-m-yy, yyyy-mm-dd, yyyy.mm.dd, etc.
- Cuando no se quiere informar de un parámetro, se deja el cuadro de texto vacío o se indica en el *checkbox* que se quiere enviar valor nulo (*Pass Null Value*), ambas opciones son válidas.

En la Figura 7.4 se han rellenado los parámetros para crear una nueva planificación.

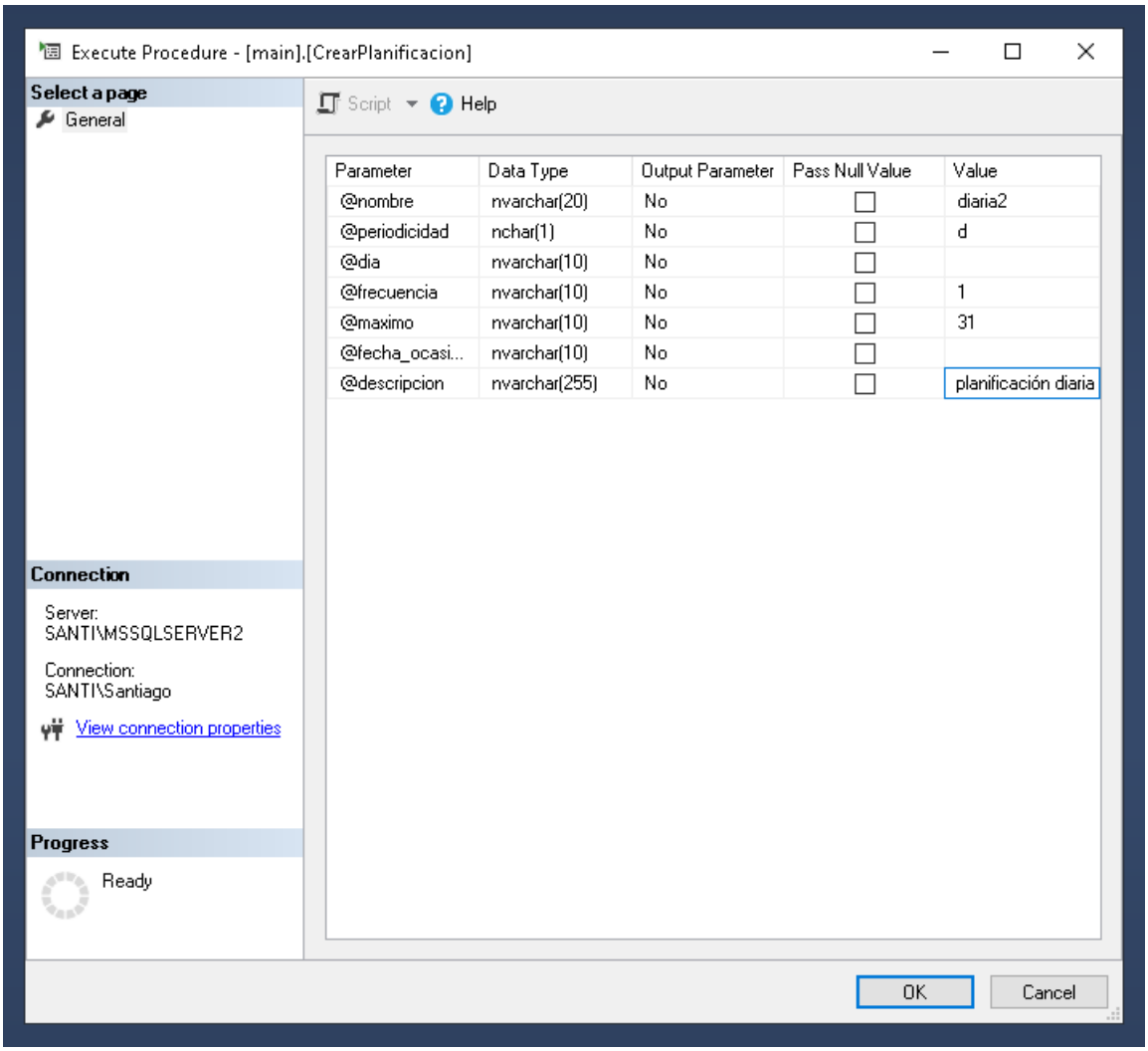

Figura 7.4.: Manual de usuario: Introducir parámetros

Pulsando *OK*, comenzará la ejecución del procedimiento, abriéndose una nueva ventana de edición. En la salida Mensajes (*Messages*) de SSMS irán apareciendo los mensajes que indican el estado de la ejecución (ver Figura 7.5).

También se ha dispuesto una plantilla para realizar cada caso de uso, dentro del directorio *Recursos\Plantillas*. Si se abre la plantilla en el editor de SSMS, se puede introducir el valor de cada parámetro, que inicialmente está asignado a *null*, entre comillas simples (’). Los parámetros

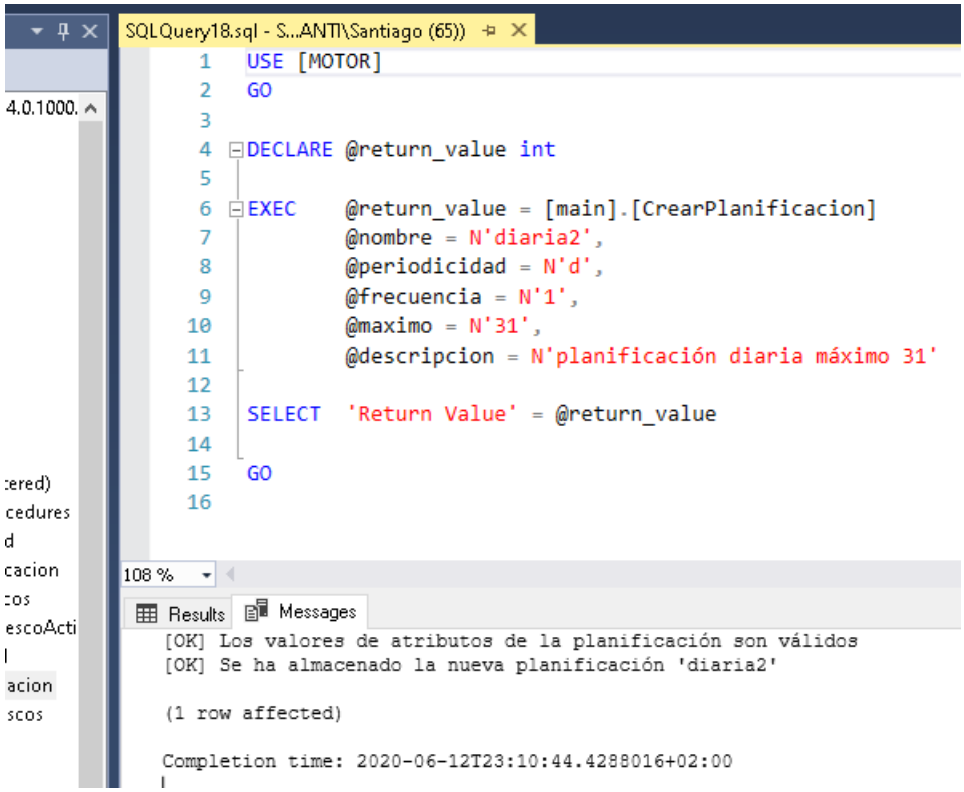

Figura 7.5.: Manual de usuario: Estado de la ejecución

no informados se dejan a *null*.

Una vez escritos los valores de los parámetros, se pulsa el botón Ejecutar (*Execute*) para realizar el caso de uso, como se puede ver en la Figura 7.6 con la plantilla *CrearEntidad.sql*, donde se crea un entidad llamada *productos* con la dimensión desnormalizada de los productos almacenados en la base de datos BASE.

Al finalizar la ejecución, se puede comprobar que se ha creado la tabla *EXPL.produccion.productos* en la Figura 7.7.

7.3. Despliegue del sistema de información

El sistema de información completo se ha desplegado en la instancia de SQL Server del ordenador personal del alumno y en el servicio de Power BI con la cuenta de Microsoft de estudiante.

El despliegue comienza por levantar las bases de datos BASE, EXPL y MOTOR en la instancia de SQL Server. Para automatizar la carga de hechos en BASE, se han creado dos tareas de SQL Server Agent:

- Una tarea realiza la carga de hechos (ventas y stock) y los nuevos productos desde el 01/01/2020 hasta la fecha en la que se crea la tarea. Sólo se ejecuta una vez a los 5 minutos

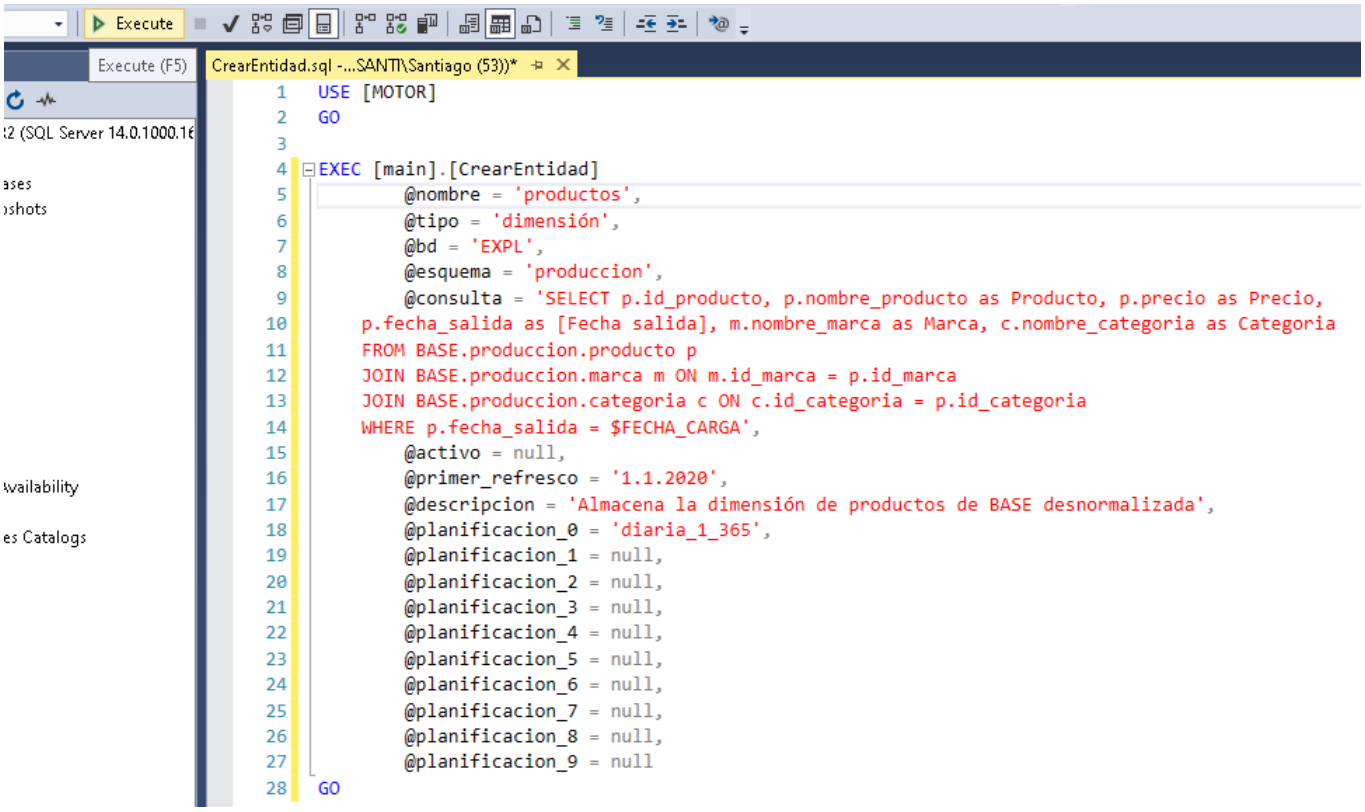

Figura 7.6.: Manual de usuario: Uso de plantillas

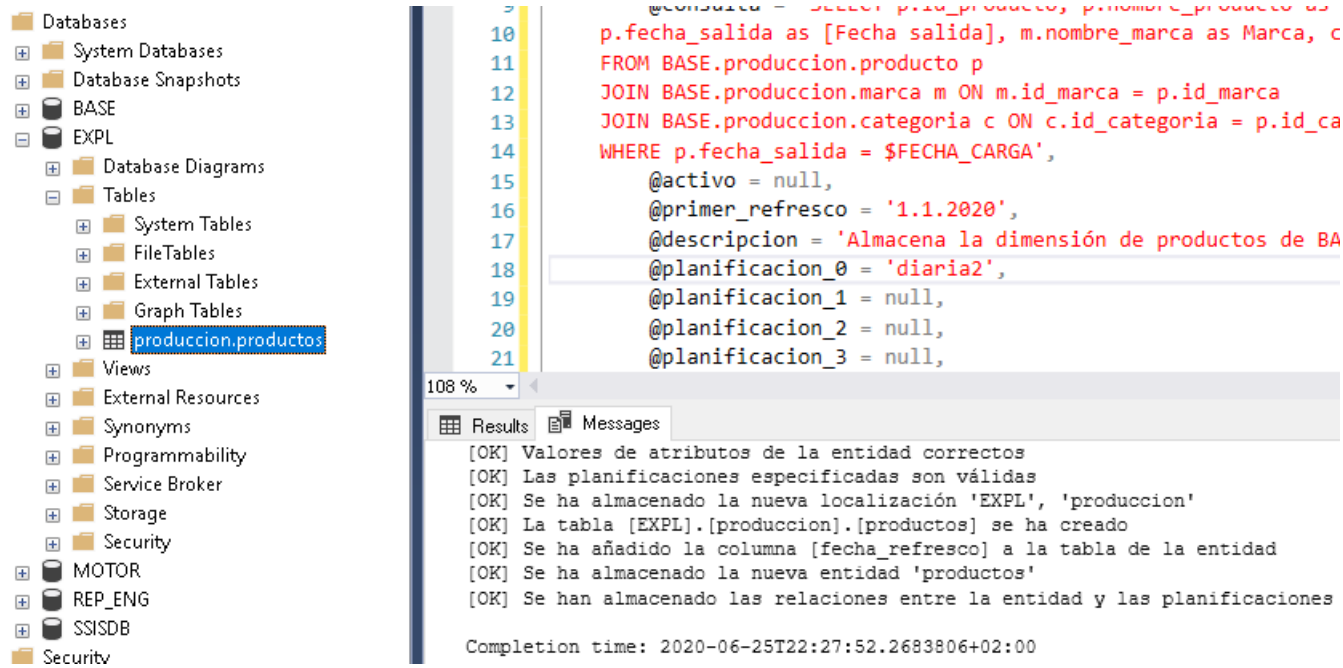

Figura 7.7.: Manual de usuario: Ejecución terminada

de crear la tarea.

- La otra tarea realiza todos los días la carga de hechos para la fecha actual.

A continuación, se han creado las planificaciones y entidades para implementar el modelo de datos en la base de datos EXPL y dos tareas en SQL Server Agent en las que se delega el trabajo de realizar el refresco de las entidades del modelo:

- Una tarea realiza el refresco de las entidades desde el 01/01/2020 hasta la fecha en la que se crea la tarea. Sólo se ejecuta una vez a los 30 minutos de crear la tarea.
- La otra tarea realiza todos los días el refresco de las entidades para la fecha actual.

Una vez que el motor ha implementado el modelo en EXPL, se puede desplegar la plataforma de visualización de la información. Para ello, se ha generado un nuevo informe a partir de la plantilla del informe implementado ajustando los parámetros de origen de datos. A continuación, se ha publicado el informe en el servicio de Power BI y se ha configurado una puerta de enlace de datos local[2], que puede acceder a los datos utilizando unas credenciales con permisos de lectura sobre EXPL. Además, se ha programado una actualización automática del conjunto de datos del informe para tener siempre una copia actualizada de los datos de EXPL en el servicio.

Con las cargas y actualizaciones configuradas, en el sistema se realiza diariamente el flujo de datos que se mostraba en la Figura 5.1. Este flujo comienza por la generación de nueva información en BASE. A continuación, el motor refresca las entidades de EXPL, añadiendo a las tablas la nueva información. Por último, el servicio de Power BI, mediante la puerta de enlace, recoge los datos de EXPL para actualizar el conjunto de datos del informe.

[2] ¿Qué es una puerta de enlace de datos local? `https://docs.microsoft.com/es-es/data-integration/gateway/service-gateway-onprem`, Microsoft Docs. Última visita: junio de 2020.

8. Conclusiones y trabajo futuro

En este último capítulo se analizarán los resultados del trabajo realizado, discutiendo las ventajas y limitaciones de la herramienta desarrollada y las conclusiones finales del proyecto. Adicionalmente, se expondrán posibles vías de trabajo futuro.

8.1. Discusión de la herramienta

8.1.1. Ventajas

Las principales ventajas de incorporar el motor en un sistema BI son:

- La implementación automática de modelos de datos dentro del servidor de bases de datos que puedan servir como origen de la información explotada por las aplicaciones BI. El administrador del servidor puede crear o modificar estos modelos especificando las entidades los que componen y los datos que almacena cada entidad.
- Personalización de la latencia de los datos contenidos en los modelos implementados a nivel de entidad por medio de las planificaciones que establecen el refresco de las entidades.

El motor supone la centralización y automatización de todos los procesos de generación y carga de modelos de datos de explotación. Además, permite prescindir de otros software de ETL, ahorrando costes en licencias, tiempo para el desarrollo de estos procesos y aprovechando al máximo las capacidades del motor de consultas.

El motor se ha diseñado de forma suficientemente flexible para poder ser utilizado en diferentes situaciones. Ejemplos de utilización del motor podrían ser:

- Generar, dentro de un DW que sólo contenga entidades que almacenan datos atómicos, entidades que contengan agregados y resúmenes de estos datos.
- Implementar uno o varios modelos relacionales tomando como base el almacén corporativo, cuyos datos sean consultados directamente por las aplicaciones BI.
- Implementar uno o varios modelos relacionales que sirvan como origen de datos para generar los modelos multidimensionales o tabulares de los servidores OLAP, los cuales utilizarán las aplicaciones BI.

Por ejemplo, el motor podría crear el modelo en estrella a partir del DW corporativo específicamente diseñado para generar cubos multidimensionales en un servidor OLAP, como Analysis Services, y disponibilizarlos para su explotación en las aplicaciones BI. También, para generar modelos tabulares en Qlik o en PowerBI, como se ha hecho en este proyecto, implementados mediante consulta directa o por copia en memoria.

Hay ocasiones en las que no es posible implementar un modelo multidimensional o tabular. El motor en estos casos puede ser muy útil para generar un modelo relacional específico que pueda ser consultado directamente por las aplicaciones BI. Por ejemplo, con la tecnología CDE (*Community Dashboard Editor*) de Pentaho, se puede consultar dinámicamente el modelo relacional implementado para obtener los datos que se muestran en las visualizaciones. Otra aplicación del motor podría ser la de generar tablas cuya información sirva como conjuntos de datos para entrenar modelos predictivos y de minería de datos.

El motor también podría ser útil para generar modelos de datos que soporten aplicaciones BI *circunstanciales* (Löser y cols., 2008), es decir, cuando puntualmente sea necesario analizar datos en otro enfoque distinto al de los aplicaciones BI que usualmente explotan los datos de la compañía. Además, aunque la propuesta del motor ha estado enfocada en su implantación en un sistema BI, podría servir de ayuda en otros sistemas informacionales que no sigan estrictamente un esquema dimensional.

8.1.2. Limitaciones

El motor obtiene los datos del nuevo refresco de una entidad ejecutando la consulta sobre el sistema o modelo base. Por tanto, la responsabilidad de que los datos obtenidos son los correctos recae sobre el Administrador, que es el encargado de hacer esas consultas, por lo que debe tener conocimientos avanzados del lenguaje de consultas utilizado y de la información del negocio, además de disponer de tiempo para construir correctamente las consultas.

Esto va en contra del llamado *self-service BI*, es decir, permitir que los usuarios de las aplicaciones BI tengan mayor independencia del equipo IT de la compañía (Imhoff y White, 2011). Aunque el motor es flexible para adaptar cambios en el modelo, sería ventajoso realizar anteriormente un análisis del modelo que se quiere generar identificando la información relevante a incorporar y realizar cambios más puntuales en función de las necesidades de análisis.

8.2. Conclusiones

Una vez finalizado el proyecto, se puede concluir que se han cumplido los objetivos marcados al inicio del mismo. Se ha desarrollado una herramienta que permite generar modelos de datos

relacionales en sistemas BI, flexible para poder adaptarse a los cambios en las necesidades de análisis del negocio.

En principio, la herramienta podría ser fácilmente exportable a otros sistemas informacionales implantados con la misma tecnología. Gracias a la arquitectura escogida, se pueden ampliar o modificar sus funcionalidades de forma sencilla, implementando los cambios en el código en la capa correspondiente. Además, mediante la utilización de los artefactos generados (PIM y PSM) se puede acortar el tiempo de desarrollo de la herramienta en otras tecnologías.

En este trabajo se han abordado con éxito metodologías y tecnologías inicialmente desconocidas para el alumno. El proyecto ha servido, además, para ampliar los conocimientos sobre las técnicas y arquitecturas de sistemas relacionadas con el BI, un área de trabajo en la que actualmente se está destinando una importante cantidad de recursos y que origina muchas oportunidades en el entorno laboral.

8.3. Trabajo futuro

La línea de trabajo futuro más relevante es incorporar el motor en un sistema de producción real, en la que se pueda comprobar verdaderamente las ventajas y limitaciones de la herramienta. Posiblemente se tendría que modificar alguna funcionalidad del motor para adaptarse a los requisitos concretos del sistema de información.

Otra posible vía de mejora es proporcionar alguna facilidad a la hora de construir las consultas de las entidades, de forma que el Administrador no tenga que especificarlas directamente en una cadena de texto. Para ello habría que limitar de alguna forma el tipo de consultas que se pueden definir. Desarrollando esta línea de trabajo, la herramienta podría hacerse más accesible para otro tipo de usuarios menos expertos.

Centrándonos en la tecnología de Microsoft BI, se podría crear una interfaz propia del motor, que facilitase la utilización del mismo. Por otra parte, habría que estudiar la posibilidad de integrar la interfaz en SSMS, para no descentralizar la administración del servidor en dos interfaces independientes.

También sería interesante poder generar el modelo con estructuras de datos diferentes a tablas básicas, según sea lo más apropiado para el tipo de entidad concreto. Por ejemplo, vistas indexadas para dimensiones desnormalizadas, o tablas particionadas para entidades con un gran número de registros por refresco.

Referencias

Adam, F., y Pomerol, J.-C. (2008). Developing Practical Decision Support Tools Using Dashboards of Information. En F. Burstein y C. W. Holsapple (Eds.), *Handbook on decision support systems 2: Variations* (p. 175-194). Springer. doi: 10.1007/978-3-540-48716-6

Adrian, M., Feinberg, D., y Cook, H. (2019). *Magic Quadrant for Operational Database Management Systems* (Inf. Téc.). Gartner.

Albright, S. C., y Winston, W. L. (2018). *Business Analytics: Data Analysis & Decision Making* (sexta ed.). Cengage Learning.

Alhir, S. S. (2013). Understanding the Model Driven Architecture (MDA). *Methods & Tools*, *11*(3), 17-24.

Alvi, I. A. (2019). *Transactional vs. Analytical Databases: How Does OLTP Differ from OLAP* (Inf. Téc.). Data Warehouse Information Center.

Badia, A. (2006). Data Warehouses. En L. C. Rivero, J. H. Doorn, y V. E. Ferraggine (Eds.), *Encyclopedia of database technologies and applications.* Idea Group Reference.

Ben-Gan, I. (2012). *Microsoft SQL Server 2012 T-SQL Fundamentals* (K. Borg, Ed.). Microsoft Corporation.

Brown, A., y Conallen, J. (2005). *An introduction to model-driven architecture. Part III: How MDA affects the iterative development process* (Inf. Téc.). IBM.

Davenport, T. H. (2010). Business Intelligence and Organizational Decisions. *International Journal of Business Intelligence Research*, *1*(1), 1-12. doi: 10.4018/jbir.2010071701

Deshpande, P. M., y Ramasamy, K. (2006). Data Warehousing, Multi-Dimensional Data Models and OLAP. En L. C. Rivero, J. H. Doorn, y V. E. Ferraggine (Eds.), *Encyclopedia of database technologies and applications.* Idea Group Reference.

Díaz, J. C. (2010). *Introducción al Business Intelligence* (primera ed.). UOC.

Hernandez-Orallo, J. (2006). Data Warehousing and OLAP. En L. C. Rivero, J. H. Doorn, y V. E. Ferraggine (Eds.), *Encyclopedia of database technologies and applications.* Idea Group Reference.

Imhoff, C., y White, C. (2011). *Self-Service Business Intelligence. Empowering Users to Generate Insights* (Inf. Téc.). Data Warehousing Institute (TDWI).

Kimball, R., y Merz, R. (1993). *The Data Warehouse Toolkit* (primera ed.). Wiley.

Kimball, R., y Ross, M. (2013). *The Data Warehouse Toolkit* (tercera ed.). Wiley.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., y Becker, B. (2007). *The Data Warehouse Lifecycle Toolkit* (segunda ed.). Wiley.

Kleppe, A., Warmer, J., y Bast, W. (2003). *MDA Explained. The Model Driven Architecture: Practice and Promise* (primera ed.). Addison-Wesley Professional.

Knabke, T., y Olbrich, S. (2013). Understanding Information System Agility: The Example of Business Intelligence. En *Proceedings of the 46th annual hawaii international conference on system sciences.* doi: 10.1109/HICSS.2013.581

Krawatzeck, R., y Dinter, B. (2015). Agile Business Intelligence: Collection and Classification of Agile Business Intelligence Actions by Means of a Catalog and a Selection Guide. *Information Systems Management*, *32*(3), 177-191. doi: 10.1080/10580530.2015.1044336

Krneta, D., Jovanović, V., y Marjanović, Z. (2014). A Direct Approach to Physical Data Vault Design. *Computer Science and Information Systems*, *11*(2), 569-599. doi: 10.2298/CSIS130523034K

Kruchten, P. (2003). *The Rational Unified Process: An Introduction* (tercera ed.). Addison-Wesley Professional.

Larson, B. (2017). *Delivering Business Intelligence with Microsoft SQL Server 2016* (cuarta ed.). Mc Graw Hill Education.

Löser, A., Hueske, F., y Markl, V. (2008). Situational Business Intelligence. En C. M., D. U., y S. T. (Eds.), *Business Intelligence for the Real-Time Enterprise* (Vol. 27). Springer. doi: 10.1007/978-3-642-03422-0_1

Mellor, S. J., y Watson, A. (2005). *Roles in the MDA Process. MDA will make developers more productive, not redundant* (Inf. Téc.). OMG.

Muñoz, L., Mazón, J.-N., y Trujillo, J. (2009). Automatic generation of ETL processes from conceptual models. En *Acm press proceeding of the acm twelfth international workshop.* doi: 10.1145/1651291.1651298

Negash, S., y Gray, P. (2008). Business Intelligence. En F. Burstein y C. W. Holsapple (Eds.), *Handbook on decision support systems 2: Variations* (p. 175-194). Springer. doi: 10.1007/978-3-540-48716-6

Plattner, H. (2009). *A common database approach for OLTP and OLAP using an in-memory column database.* ACM SIGMOD International Conference on Management of data. doi: 10.1145/1559845.1559846

Ponniah, P. (2001). *Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals* (primera ed.). Wiley-Interscience.

Post, G. V. (2004). *Database Management Systems. Designing and building business applications* (tercera ed.). Mcgraw-Hill.

Rahman, N. (2012). Incremental Load in a Data Warehousing Environment. En V. Sugumaran (Ed.), *Insights into advancements in intelligent information technologies: Discoveries* (p. 161-177). Intel Corporation. doi: 10.4018/978-1-4666-0158-1.ch009

Richardson, J., Sallam, R., Schlegel, K., Kronz, A., y Sun, J. (2020). *Magic Quadrant for Analytics and Business Intelligence Platforms* (Inf. Téc.). Gartner.

Sanders, G., y Shin, S. (2001). Denormalization effects on performance of RDBMS. En *Proceedings of the 34th annual hawaii international conference on system sciences.* doi: 10.1109/HICSS.2001.926306

Siegel, J. M. (2014). *Model Driven Architecture (MDA) MDA Guide rev. 2.0* (Inf. Téc.). Object Management Group (OMG).

Silberschatz, A., Korth, H. F., y Sudarsham, S. (2011). *Database System Concepts* (sexta ed.). Mcgraw-Hill.

Tamayo, M., y Moreno, F. J. (2006). Análisis del modelo de almacenamiento MOLAP frente al modelo de almacenamiento ROLAP. *Ingeniería e Investigación*, *26*(3), 135-142.

Thomsen, E. (2002). *OLAP Solutions. Building Multidimensional Information Systems* (segunda ed.). Wiley.

Vassiliadis, P., y Sellis, T. (1999). A survey of logical models for OLAP databases. *ACM SIGMOD*, *28*(4), 65-69. doi: 10.1145/344816.344869

Wixom, B., y Watson, H. (2010). The BI-Based Organization. *International Journal of Business Intelligence Research*, *1*(1), 13-28. doi: 10.4018/jbir.2010071702

Zhang, X., Sun, W., Wang, W., Feng, Y., y Shi, B. (2006). Generating Incremental ETL Processes Automatically. En *First international multi-symposiums on computer and computational sciences.* doi: 10.1109/imsccs.2006.229

Zimmer, M., Baars, H., y Kemper, H.-G. (2012). The Impact of Agility Requirements on Business Intelligence Architectures. En *Proceedings of the 45th annual hawaii international conference on system sciences.* doi: 10.1109/hicss.2012.567

A. Anexo I: PSM

```
CREATE DATABASE MOTOR
GO

USE MOTOR
GO

/** DBO **/

CREATE TABLE dbo.fecha
    (
     id_fecha integer NOT NULL ,
     fecha_estandar date NOT NULL ,
     des_fecha nvarchar (50) NOT NULL ,
     dia_mes integer NOT NULL ,
     dia_semana integer NOT NULL ,
     nombre_dia_semana nvarchar (15) NOT NULL ,
     semana_mes integer NOT NULL ,
     semana_anio integer NOT NULL ,
     id_mes integer NOT NULL ,
     mes_anio integer NOT NULL ,
     nombre_mes nvarchar (15) NOT NULL ,
     id_trimestre integer NOT NULL ,
     trimestre_anio integer NOT NULL ,
     nombre_trimestre nvarchar (2) NOT NULL ,
     anio integer NOT NULL
    )
GO
ALTER TABLE dbo.fecha ADD CONSTRAINT fecha_pk PRIMARY KEY CLUSTERED (id_fecha)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
```

```
/** MAIN **/

CREATE SCHEMA main
GO

CREATE PROCEDURE main.CrearEntidad
        @nombre nvarchar(128)
        ,@tipo nvarchar(50)
        ,@esquema nvarchar(128)
        ,@bd nvarchar(128)
        ,@consulta nvarchar(max)
        ,@activo nvarchar(1)
        ,@primer_refresco nvarchar(10)
        ,@descripcion nvarchar(255)
        ,@planificacion_0 nvarchar(20)
        ,@planificacion_1 nvarchar(20)
        ,@planificacion_2 nvarchar(20)
        ,@planificacion_3 nvarchar(20)
        ,@planificacion_4 nvarchar(20)
        ,@planificacion_5 nvarchar(20)
        ,@planificacion_6 nvarchar(20)
        ,@planificacion_7 nvarchar(20)
        ,@planificacion_8 nvarchar(20)
        ,@planificacion_9 nvarchar(20)
AS
GO
CREATE PROCEDURE main.CrearPlanificacion
        @nombre nvarchar(20)
        ,@periodicidad nchar(1)
        ,@dia nvarchar(10)
        ,@frecuencia nvarchar(10)
        ,@maximo nvarchar(10)
        ,@fecha_ocasional nvarchar(10)
        ,@descripcion nvarchar(255)
AS
GO
CREATE PROCEDURE main.RealizarRefrescos
        @fecha_carga nvarchar(10)
AS
GO
CREATE PROCEDURE main.BorrarRefrescos
```

```
        @nombre nvarchar(128),
        @bd nvarchar(128),
        @esquema nvarchar(128),
        @fecha_inicio nvarchar(10),
        @fecha_fin nvarchar(10)
AS
GO
CREATE PROCEDURE main.CambiarRefrescoActivo
        @nombre nvarchar(128),
        @bd nvarchar(128),
        @esquema nvarchar(128),
        @activo nvarchar(1)
AS
GO
CREATE PROCEDURE main.BorrarEntidad
        @nombre nvarchar(128),
        @bd nvarchar(128),
        @esquema nvarchar(128)
AS
GO
CREATE PROCEDURE main.BorrarPlanificacion
        @nombre nvarchar(128)
AS
GO

/** CONTROLADOR **/

CREATE SCHEMA controlador
GO

CREATE TABLE controlador.mensaje
    (
     id_mensaje integer NOT NULL IDENTITY NOT FOR REPLICATION,
     mensaje nvarchar (500) NOT NULL ,
         tiempo datetime NOT NULL ,
     id_ejecucion integer NOT NULL
    )
GO
ALTER TABLE controlador.mensaje ADD CONSTRAINT mensaje_pk
PRIMARY KEY CLUSTERED (id_mensaje)
     WITH (
```

```
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE controlador.ejecucion
    (
     id_ejecucion integer NOT NULL IDENTITY NOT FOR REPLICATION ,
     procedimiento nvarchar (128) NOT NULL ,
     estado_ejecucion nvarchar (5) NOT NULL ,
     tiempo_inicio DATETIME NOT NULL ,
     tiempo_fin DATETIME NOT NULL ,
     duracion BIGINT NOT NULL ,
     fecha_ejecucion integer NOT NULL
    )
GO
ALTER TABLE controlador.ejecucion ADD CONSTRAINT ejecucion_pk
PRIMARY KEY CLUSTERED (id_ejecucion)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
ALTER TABLE controlador.mensaje
    ADD CONSTRAINT mensaje_ejecucion_fk FOREIGN KEY (id_ejecucion)
    REFERENCES controlador.ejecucion (id_ejecucion)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE controlador.ejecucion
    ADD CONSTRAINT ejecucion_fecha_fk FOREIGN KEY (fecha_ejecucion)
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO

CREATE PROCEDURE controlador.crear_entidad
AS
GO
CREATE PROCEDURE controlador.crear_planificacion
AS
GO
CREATE PROCEDURE controlador.realizar_refrescos
AS
```

```
GO
CREATE PROCEDURE controlador.borrar_refrescos
AS
GO
CREATE PROCEDURE controlador.cambiar_refresco_activo
AS
GO
CREATE PROCEDURE controlador.borrar_entidad
AS
GO
CREATE PROCEDURE controlador.borrar_planificacion
AS
GO
CREATE PROCEDURE controlador.nueva_ejecucion
AS
GO
CREATE PROCEDURE controlador.actualizar_ejecucion
AS
GO
CREATE PROCEDURE controlador.nuevo_mensaje
AS
GO
CREATE PROCEDURE controlador.aplicar_refresco
AS
GO

/** MODELO **/

CREATE SCHEMA modelo
GO

CREATE TABLE modelo.entidad
    (
     id_entidad integer NOT NULL IDENTITY NOT FOR REPLICATION ,
     nombre nvarchar (128) NOT NULL ,
     descripcion nvarchar (255) ,
     tipo nvarchar (50) NOT NULL ,
     consulta nvarchar (max) NOT NULL ,
     activo bit NOT NULL ,
     primer_refresco integer NOT NULL ,
     id_localizacion integer NOT NULL ,
```

```
     existe bit NOT NULL
    )
GO
ALTER TABLE modelo.entidad ADD CONSTRAINT entidad_pk
PRIMARY KEY CLUSTERED (id_entidad)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE modelo.entidad_planificacion
    (
     id_entidad integer NOT NULL ,
     id_planificacion integer NOT NULL
    )
GO
ALTER TABLE modelo.entidad_planificacion ADD CONSTRAINT entidad_planificacion_pk
PRIMARY KEY CLUSTERED (id_entidad, id_planificacion)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE modelo.localizacion
    (
     id_localizacion integer NOT NULL IDENTITY NOT FOR REPLICATION,
     bd nvarchar (128) NOT NULL ,
     esquema nvarchar (128) NOT NULL
    )
GO
ALTER TABLE modelo.localizacion ADD CONSTRAINT localizacion_pk
PRIMARY KEY CLUSTERED (id_localizacion)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE modelo.periodicidad
    (
     id_periodicidad integer NOT NULL IDENTITY NOT FOR REPLICATION,
     nombre nvarchar (50) NOT NULL ,
     descripcion nvarchar (255) ,
     codigo nvarchar (5) NOT NULL ,
     min_dia integer NOT NULL ,
```

```
     max_dia integer NOT NULL ,
     min_frecuencia integer NOT NULL ,
     max_frecuencia integer NOT NULL ,
     min_maximo integer NOT NULL ,
     max_maximo integer NOT NULL ,
     min_fecha integer NOT NULL ,
     max_fecha integer NOT NULL
    )
GO
ALTER TABLE modelo.periodicidad ADD CONSTRAINT periodicidad_pk
PRIMARY KEY CLUSTERED (id_periodicidad)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE modelo.planificacion
    (
     id_planificacion integer NOT NULL IDENTITY NOT FOR REPLICATION,
     nombre nvarchar (10) NOT NULL ,
     descripcion nvarchar (255) ,
     dia integer NOT NULL ,
     frecuencia integer NOT NULL ,
     maximo integer NOT NULL ,
     fecha_ocasional integer NOT NULL ,
     id_periodicidad integer NOT NULL ,
         existe bit NOT NULL
    )
GO
ALTER TABLE modelo.planificacion ADD CONSTRAINT planificacion_pk
PRIMARY KEY CLUSTERED (id_planificacion)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
CREATE TABLE modelo.refresco
    (
     id_refresco integer NOT NULL IDENTITY NOT FOR REPLICATION ,
     fecha_refresco integer NOT NULL ,
     id_entidad integer NOT NULL ,
     id_planificacion integer NOT NULL ,
     existe bit NOT NULL
```

```
    )
GO
ALTER TABLE modelo.refresco ADD CONSTRAINT refresco_pk
PRIMARY KEY CLUSTERED (id_refresco)
     WITH (
     ALLOW_PAGE_LOCKS = ON ,
     ALLOW_ROW_LOCKS = ON )
GO
ALTER TABLE modelo.entidad
    ADD CONSTRAINT entidad_fecha_fk FOREIGN KEY (primer_refresco)
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.entidad
    ADD CONSTRAINT entidad_localizacion_fk FOREIGN KEY(id_localizacion)
    REFERENCES modelo.localizacion(id_localizacion)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.entidad_planificacion
    ADD CONSTRAINT entidad_planificacion_entidad_fk FOREIGN KEY (id_entidad)
    REFERENCES modelo.entidad (id_entidad)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.entidad_planificacion
    ADD CONSTRAINT entidad_planificacion_planificacion_fk FOREIGN KEY
    (id_planificacion)
    REFERENCES modelo.planificacion (id_planificacion)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.planificacion
    ADD CONSTRAINT planificacion_fecha_fk FOREIGN KEY (fecha_ocasional)
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.periodicidad
    ADD CONSTRAINT periodicidad_fecha_fk FOREIGN KEY (min_fecha)
```

```
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.periodicidad
    ADD CONSTRAINT periodicidad_fecha_fkv1 FOREIGN KEY (max_fecha)
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.planificacion
    ADD CONSTRAINT planificacion_periodicidad_fk FOREIGN KEY (id_periodicidad)
    REFERENCES modelo.periodicidad (id_periodicidad)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.refresco
    ADD CONSTRAINT refresco_entidad_fk FOREIGN KEY (id_entidad)
    REFERENCES modelo.entidad (id_entidad)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.refresco
    ADD CONSTRAINT refresco_fecha_fk FOREIGN KEY (fecha_refresco)
    REFERENCES dbo.fecha (id_fecha)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO
ALTER TABLE modelo.refresco
    ADD CONSTRAINT refresco_planificacion_fk FOREIGN KEY (id_planificacion)
    REFERENCES modelo.planificacion (id_planificacion)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
GO

/** entidades **/

CREATE PROCEDURE modelo.validar_valores_entidad
AS
GO
CREATE PROCEDURE modelo.comprobar_datos_entidad
```

```
AS
GO
CREATE PROCEDURE modelo.nueva_localizacion
AS
GO
CREATE PROCEDURE modelo.nueva_entidad
AS
GO
CREATE PROCEDURE modelo.nueva_entidad_planificacion
AS
GO
CREATE PROCEDURE modelo.obtener_datos_entidad
AS
GO
CREATE PROCEDURE modelo.actualizar_entidad
AS
GO
CREATE PROCEDURE modelo.seleccionar_entidades_refresco
AS
GO

/** planificaciones **/

CREATE PROCEDURE modelo.validar_planificaciones
AS
GO
CREATE PROCEDURE modelo.validar_valores_planificacion
AS
GO
CREATE PROCEDURE modelo.nueva_planificacion
AS
GO
CREATE PROCEDURE modelo.comprobar_datos_planificacion
AS
GO
CREATE PROCEDURE modelo.actualizar_planificacion
AS
GO

/** refrescos **/
```

```
CREATE PROCEDURE modelo.comprobar_maximo_refrescos
AS
GO
CREATE PROCEDURE modelo.nuevo_refresco
AS
GO
CREATE PROCEDURE modelo.actualizar_refrescos
AS
GO

/** FACHADA **/

CREATE SCHEMA fachada
GO

CREATE PROCEDURE fachada.existe_tabla
AS
GO
CREATE PROCEDURE fachada.existe_localizacion
AS
GO
CREATE PROCEDURE fachada.preparar_consulta
AS
GO
CREATE PROCEDURE fachada.ejecutar_subconsulta
AS
GO
CREATE PROCEDURE fachada.crear_tabla
AS
GO
CREATE PROCEDURE fachada.agregar_columna
AS
GO
CREATE PROCEDURE fachada.eliminar_tabla
AS
GO
CREATE PROCEDURE fachada.eliminar_datos
AS
GO
CREATE PROCEDURE fachada.insertar_datos
AS
```

```
GO

/** CONSULTAS **/

CREATE SCHEMA consultas
GO

CREATE PROCEDURE consultas.resultado_int
AS
GO
CREATE PROCEDURE consultas.sin_resultado
AS
GO
```

B. Anexo II: Diseño de BASE

Partiendo del modelo lógico del almacén base mostrado en la Figura 2.1, se ha derivado un modelo físico específico para SQL Server. El diagrama de este modelo se puede ver en la Figura B.1. Las entidades incluidas se dividen en dos esquemas que diferencian el tipo de información almacenada: **produccion**, que contiene las tablas **categoria**, **marca**, **producto**, y **stock** y por otro lado **ventas**, con las tablas **cliente**, **empleado**, **tienda**, **pedido**, **item_pedido**. Además, se ha agregado la tabla **fecha** en el esquema por defecto **dbo**.

La inserción de las dimensiones se lleva a cabo mediante instrucciones INSERT incluidas junto con las CREATE en el script *BASE.sql*. La inserción inicial del stock para cada producto y tienda se lleva a cabo por medio de un TRIGGER que se activa después de una inserción en la tabla **producto**, en la fecha de salida del mismo.

Con respecto a los hechos, sin embargo, no poseemos ninguna fuente de la que obtenerlos, por lo que no hay más remedio que generarlos. De ello se encarga la procedimiento **carga_hechos**. Este procedimiento realiza la generación e inserción de nuevos registros para las tablas **pedido** e **item_pedido** para una fecha indicada y la actualización del stock según los nuevos pedidos.

En cada ejecución el procedimiento inserta entre 10 y 30 pedidos, cada uno con entre 1 y 3 items. Para simular la salida de nuevos productos, este procedimiento también realiza en algunas fechas concretas la inserción de nuevas filas en la tabla **producto**.

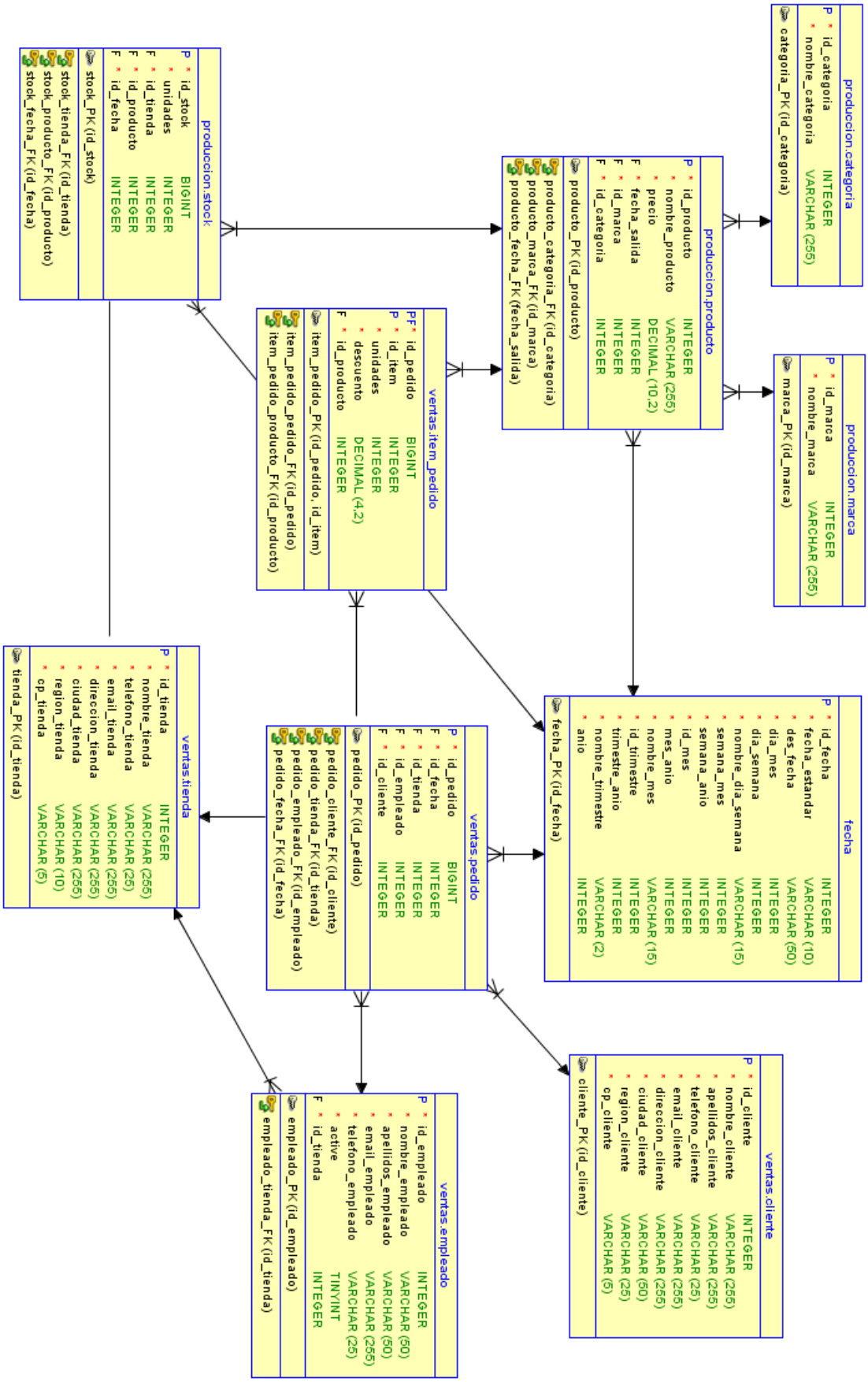

Figura B.1.: Modelo físico del almacén base

C. Anexo III: Organización de los documentos

Los documentos derivados del desarrollo se encuentran en la siguiente carpeta de OneDrive: `https://alumnosuvaes-my.sharepoint.com/:f:/g/personal/santiago_perez_alumnos_uva_es/EtXc686FyNxFsHq14bS53mMBSku6fD601HvTCZgCBxJurA?e=yOzuUk`

- *PIM*: contiene la codificación del PIM en estándar XMI y en XML.
- *PSM*: contiene la codificación del PSM en SQL.
- *Codigo*: contiene el código fuente del motor.
  - *despliegue.bat*: despliega el motor en una instancia de SQL Server.
- *Recursos*
  - *Scripts*
    - *BASE.sql*: implementa la base de datos BASE.
    - *EXPL.sql*: implementa la base de datos EXPL.
    - *jobs_BASE.sql*: crea las tareas de carga de BASE.
    - *jobs_MOTOR.sql*: crea las tareas de realización de refrescos del motor.
    - *entidades_planificaciones.sql*: crea las entidades y planificaciones.
  - *Plantillas*: contiene las plantillas del motor para realizar cada caso de uso.
  - *Visualizacion*
    - *VizVentasStock.pbit*: plantilla del informe de Power BI.