Rodriguez, Ivan Emanuel

Reingeniería y migración de aplicación de Business Intelligence (BI) a plataforma en la nube

2022

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Reingeniería y migración de aplicación de Business Intelligence (BI) a plataforma en la nube

Ivan Emanuel Rodriguez

Florencio Varela, abril de 2022

Estudiante
Ivan Emanuel Rodriguez
ivancio87@gmail.com

Organización donde se realiza la Práctica Profesional Supervisada
Datalytics S.R.L.
Basavilbaso 1350, C1001 CABA
+54 11 5263 3777
Sector: Ingeniería de datos

Tutor organizacional
Lic. Buffa Claudio.
claudio.buffa@datalytics.com

Docente supervisor
Dr. Ing. Morales, Martín
martin.morales@unaj.edu.ar

Docente tutor del Taller de Apoyo para la Producción de Textos Académicos
Prof. Lavigna Lia.
llavigna@unaj.edu.ar

Coordinador de la carrera de Ingeniería en Informática
Dr. Ing. Morales, Martín
martin.morales@unaj.edu.ar

Resumen

Este proyecto se realizó en la empresa de consultoría de datos Datalytics para uno de sus clientes y consistió en la reingeniería y migración de un modelo de datos actualmente desplegado en la plataforma de inteligencia de negocios Cognos BI a la nube de Azure. Esta necesidad surgió, debido a que Cognos BI y el Data Warehouse Oracle que utilizaba, presentaban muchos problemas de performance y velocidad y no contaban con personal técnico que brinde soporte, lo que significaba un riesgo alto para el cliente.

El objetivo del proyecto fue replicar los datos desde los orígenes de la información, como así también, las reglas de negocio que se aplican dentro de la misma base de datos con la intención de que el Data Lake sea la única fuente de información para todas las herramientas de consulta/visualización.

Las tareas del proyecto abarcaron: la ingesta de los múltiples orígenes de información, la aplicación y generación de las reglas de negocio utilizando lenguaje SQL y Python, el desarrollo de los distintos modelos de datos y la réplica de los modelos para su consumo con la herramienta de visualización Power BI.

Con esta reingeniería y migración se logró centralizar toda la información y conocer todos los procesos involucrados en la generación de los reportes, lo que permite brindar el soporte técnico adecuado. Así también, se mejoró considerablemente la performance y velocidad de consulta por parte de los usuarios.

Reingeniería – Migración – Data Warehouse – Data Lake - Reportes

Abstract

This project was carried out in the data consulting company Datalytics for one of its clients and consisted in the reengineering and migration of a data model currently deployed in the business intelligence platform Cognos BI to the Azure cloud. This need came up because Cognos BI and the Oracle Data Warehouse it used presented many performance and speed problems and did not have technical staff to provide support, which meant a high risk for the client.

The objective of the project was to replicate the data from the sources of the information, as well as the business rules that are applied within the same database; with the intention that the Data Lake is the only source of information for all query/visualization tools.

The project tasks covered: the ingestion of the multiple information sources, the application and generation of the business rules using SQL and Python, the development of the different data models and the replication of the models for their consumption with the Power BI visualization tool.

With this reengineering and migration, we were able to centralize all the information and know all the processes involved in the generation of reports, which allows us to provide the appropriate technical support. This also considerably improved the performance and speed of consultation by users.

Reengineering – Migration – Data Warehouse – Data Lake - Reports

Dedicatorias y agradecimientos

Agradezco a mi tutor organizacional Claudio Buffa por el acompañamiento durante la realización de la PPS, a la profesora Lia Lavigna por el apoyo y enseñanza brindado durante el desarrollo del presente informe. También, a Datalytics por permitirme utilizar este proyecto a fines de egresarme.

Todo este trabajo se lo dedico a mi familia, pareja, amigos y compañeros de la universidad. Gracias por estar presentes y darme el aliento necesario para llegar a este momento.

ÍNDICE

ÍNDICE DE FIGURAS

1. Introducción

1.1 Descripción del proyecto

El presente trabajo de Práctica Profesional Supervisada (PPS) surge de la vinculación entre Datalytics S.R.L. (consultora dedicada a la inteligencia de negocios que ayuda a empresas a transformar sus datos en información para la toma de decisiones) y la compañía cliente (empresa cuya actividad principal es producir y comercializar bebidas a nivel mundial). De esta vinculación se crean distintos equipos de trabajo, surgen y se generan nuevos proyectos y cada equipo debe estar compuesto por múltiples perfiles de trabajo de acuerdo con las necesidades de cada proyecto. Estos generalmente se componen de ingenieros y visualizadores de datos, consultores de Inteligencia Empresarial (BI), analistas funcionales, especialistas en la gestión de negocios (Product Manager), entre otros.

Todos estos equipos deben trabajar en forma coordinada, siguiendo buenas prácticas de desarrollo con el fin de lograr sus objetivos personales y los objetivos de las organizaciones involucradas. Debido a ello, se establecen lineamientos a seguir en los procesos de desarrollo, se proponen mejoras tanto en la modalidad de trabajo como en la utilización de las tecnologías, se investiga el uso de nuevas herramientas, entre otras tareas.

Una de las necesidades, que surgió de esta vinculación y que se utilizará como propósito para realizar esta PPS, fue la de migrar y decomisar la información almacenada en la plataforma de BI “IBM Cognos” implementada sobre un Data Warehouse (repositorio de datos empresarial centralizado) en Oracle, debido a que esta tecnología no ha sido actualizada dentro de la compañía.

Para llevar a cabo dicha migración, los líderes y responsables de las organizaciones mencionadas han decidido llevar toda su plataforma analítica a la nube de Microsoft Azure, ya que es una tecnología más moderna con múltiples beneficios. Entre estos se destacan la reducción de costos, escalabilidad,

rendimiento acorde a las necesidades del proyecto, aumento de la seguridad de la información, etc.

Un factor principal de esta migración es repensar el modelo actualmente existente con el fin de adaptarlo y personalizarlo de acuerdo con las necesidades que requieren los visualizadores para realizar los reportes de análisis de los indicadores del negocio, los cuales ayudan a los líderes de la empresa a tomar sus decisiones.

Este trabajo consistirá fundamentalmente en el análisis, desarrollo, migración y adaptación del modelo dimensional COGNOS OLAP de Gestión Comercial con el fin de mejorar la rapidez, eficiencia y respuesta de los distintos reportes de análisis de negocio que realizan los múltiples equipos que componen la empresa.

1.2 Objetivos

Los objetivos específicos de la PPS se encuentran detallados a continuación:

1.2.1 Migrar la información relevante a una plataforma en la nube

Lograr que toda la información que se utiliza actualmente, para los reportes de negocio de múltiples proyectos, se migre correctamente desde las distintas fuentes origen a la plataforma en la nube de Microsoft Azure, con el fin de tener todos los datos necesarios en un repositorio centralizado, permitiendo el acceso a estos por parte de los usuarios y su utilización en diferentes proyectos.

1.2.2 Desarrollar un nuevo modelo dimensional OLAP de gestión comercial

Generar una reingeniería y reestructuración del modelo actual que detecte cuáles son las características más relevantes del modelo y cuáles son las modificaciones necesarias de acuerdo con las necesidades del negocio,

descartando todo aquello que haya quedado desactualizado o en desuso para lograr un modelo con mayor calidad, respuesta, eficiencia y performance.

1.2.3 Alimentar reportes en forma eficiente

Lograr disponer de los datos necesarios para obtener las diferentes métricas utilizadas en los reportes de negocio mediante la construcción y ejecución de consultas eficientes al nuevo modelo desarrollado.

1.3 Tareas a ejecutar

Para llevar a cabo el desarrollo y culminación del presente proyecto se han dividido las tareas en olas a ejecutar en un tiempo y forma predefinida la cual se detalla a continuación:

1.3.1 Ola 1: Relevamiento e ingesta de fuentes de datos

En esta primera ola se llevará a cabo un análisis de todas las fuentes de origen de los datos que actualmente alimentan los reportes que realiza IBM Cognos, se analizará cada una de las bases de datos, sistemas de gestión y archivos planos utilizados.

Luego de este análisis se podrá definir cuáles son cada una de las tablas o documentos que contienen información relevante para luego ingestarlo a la plataforma en la nube, descartando todo dato que haya quedado en desuso o desactualizado. También se debe considerar que existen tablas o información que ya ha sido replicada en la nube por otros equipos de la empresa, la cual no se debe volver a migrar para evitar la inconsistencia y duplicidad de información.

1.3.2 Ola 2: Análisis y documentación del modelo dimensional

En la segunda ola se procederá a analizar y documentar cada una de las dimensiones y tablas de hechos existentes en el modelo multidimensional actual

junto con los ajustes necesarios a llevar a cabo para lograr convergencia con el modelo a utilizar en la nube y los modelos implementados por otros equipos de la empresa. Este análisis y documentación son el punto de entrada para la reingeniería del código que se realizará en la ola 3.

1.3.3 Ola 3: Reingeniería del modelo de gestión comercial

En esta tercera ola se procederá al desarrollo del nuevo modelo dimensional aplicando una reingeniería a los procesos ETL (extracción, transformación y carga) implementados, así como al modelo dimensional actual. Esta etapa consiste en analizar detenidamente cada una de las dimensiones y tablas de hecho y relevar cuáles fueron los procedimientos y códigos desarrollados para poder extraer los datos desde las fuentes origen y combinarlos con el fin de alimentar el modelo dimensional. Se considera la etapa más larga de este proyecto ya que es necesario repensar los desarrollos implementados en la base de datos Oracle donde se aloja el DW actual para poder adaptarlos al código e implementarlo en la nube de Microsoft Azure.

1.3.4 Ola 4: Pruebas y ajustes del modelo

La cuarta ola consiste en realizar todas las pruebas y adaptaciones necesarias al nuevo modelo de gestión comercial para que logre alcanzar la calidad y eficiencia esperadas por el proyecto. Se deben realizar pruebas de obtención de datos mediante consultas en lenguaje SQL e implementar reportes de prueba sobre el nuevo modelo para comprobar que su respuesta y performance son las indicadas por la empresa.

1.3.5 Ola 5: Implementación y desarrollo de consultas en ambiente productivo

Esta ola 5 es la etapa final del proyecto y en ella se lleva el modelo dimensional desarrollado y puesto a prueba en las olas anteriores a un ambiente de

producción para que pueda comenzar a ser utilizado por los usuarios finales quienes, en este caso, son visualizadores y altos mandos de la empresa.

En esta etapa también se llevarán a cabo consultas específicas de manipulación de datos con el objetivo de ayudar a los visualizadores a obtener la información necesaria para sus reportes en forma eficiente y confiable.

1.4 Cronograma de trabajo

Se define un tiempo estimado de duración del proyecto a partir del siguiente diagrama.

Etapa / Mes	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6
Ola 1						
Ola 2						
Ola 3						
Ola 4						
Ola 5						

Aclaraciones

- Durante el primer mes en la ola 1 se procederá a la ingesta todas las tablas que ya se conocen y que utiliza el sistema actual, por lo cual ciertas tareas

de ingesta podrían desarrollarse durante los meses posteriores en caso de surgir nuevas tablas del análisis dispuesto en la ola 2.

- La ola 3 comienza en el segundo mes junto a la etapa de análisis, debido a que existen ciertas consideraciones del modelo de las cuales ya se tiene conocimiento y no hay impedimentos en realizar. A medida que se avanza con el relevamiento, el desarrollo continuará de acuerdo con las necesidades que se identifiquen en este.

2. Desarrollo

2.1 Definiciones y conceptos

Con el fin de dar un mejor contexto y entendimiento del presente proyecto de PPS en este apartado se procederá a introducir definiciones y conceptos relacionados al análisis de datos.

2.1.1 Inteligencia de negocios

La inteligencia de negocios (business Intelligence) se refiere al conjunto de procesos, que se realiza sobre los datos utilizando diferentes tecnologías con el fin de proveer a la organización o empresa de información relevante que la ayude a tomar decisiones.

Por otra parte, se la considera como el proceso que engloba todas las acciones necesarias para transformar los datos en conocimiento que ayudan a llevar acciones y tomar decisiones para crear una ventana competitiva respecto a otras organizaciones. Entre estas acciones se pueden mencionar el ETL (extracción, transformación y carga de datos), big data, Data Warehouse, reporting, visualización, entre otros.

2.1.2 ETL (Extract, Transform, Load)

Tal como sus siglas en inglés lo indican el ETL hace referencia a la extracción, transformación y carga de datos.

Extracción: Proceso mediante el cual se obtienen los datos de las fuentes de información y se los ubica en algún lugar para su posterior transformación.

- Transformación: Aplica reglas de negocio sobre los datos extraídos en el proceso anterior con el fin de responder a las necesidades del negocio o proyecto.

- Carga: Almacena el resultado de las transformaciones en la base de datos final la que, en la mayoría de los casos, es un Data Warehouse.

En las arquitecturas modernas el proceso ETL se está migrando a un proceso ELT (extract, load y transform) donde, en primer lugar, se extraen y cargan todos los datos en el sistema destino y directamente allí se le aplican las transformaciones necesarias. Este tipo de procesos se realizan en entornos distribuidos en plataformas en la nube como Azure, AWS, Google Cloud Platform, entre otros.

2.1.3 Big Data

Big Data es el nombre que se le da al conjunto de información que crece de una manera tan exponencial que resulta prohibitivo almacenarlos y/o procesarlos con métodos o técnicas tradicionales del mundo de las bases de datos relacionales.

Sus principales características son las siguientes:

- Velocidad: los datos se generan a un ritmo exponencial muy elevado.

- Volumen: en Big Data, debido al gran volumen de datos, se dejó de hablar de terabytes (billones) para comenzar a almacenar de petabytes (miles de billones) y zettabytes (miles de trillones).

- Variedad: remite a los datos estructurados y no estructurados provenientes de múltiples orígenes (web, sensores, logs, archivos de texto, hojas de cálculo, entre otros).

- Complejidad: se refiere a un volumen de datos tan grande y variado que no es posible de procesar y almacenar con técnicas tradicionales.

2.1.4 Data Warehouse

Data Warehouse es un sistema que extrae, limpia, normaliza y entrega datos de diversas fuentes a un modelo de almacenamiento dimensional para implementar y soportar consultas y análisis de datos, que permite a los ejecutivos de negocios organizar, comprender y utilizar estos datos para tomar decisiones estratégicas. En síntesis, es una colección de datos orientada al negocio, integrada, variante en el tiempo y no volátil para el soporte del proceso de toma de decisiones de la gerencia (William Harvey Inmon).

Características:

Orientado al negocio: el DW (Data Warehouse) excluye toda la información que no resulta relevante para la toma de decisiones de la organización, debido a que la información se clasifica en base a los aspectos que son de interés para la empresa.

Integrado: porque todos los datos de diversas fuentes que son producidos por distintos departamentos, secciones y aplicaciones, ya sea internas o externas a la organización (pero de vital importancia para la toma de decisiones gerenciales) son procesados y luego almacenados en el DW.

Variante en el tiempo: en el DW se almacenan todos los datos históricos y la información no puede ser eliminada ni modificada, por lo cual se pueden encontrar los mismos datos para distintos periodos de tiempo. Otra característica importante es que en el DW todos los datos tienen un campo de tiempo, que sirven para identificar a qué periodo de la información se está accediendo.

No volátil: la información en el DW no cambia y no se pueden borrar o eliminar datos. Lo único que se permite en el DW es la carga de nuevos datos y el acceso a los mismos mediante consultas para acceder a la información almacenada.

2.1.5 Data Lake

Data Lake es un repositorio centralizado, que permite almacenar una gran cantidad de datos brutos estructurados y no estructurados a cualquier escala, es decir, que los datos dentro de un Data Lake se mantienen tal cual provienen de los múltiples orígenes, sin ningún tipo de procesamiento.

Los Data Lakes se suelen configurar en un clúster de consumo económico y escalable, por lo cual no existe preocupación por la capacidad de almacenamiento. Estos clústeres, generalmente, se encuentran en la nube debido a su bajo costo, alta escalabilidad y excelente rendimiento.

Una vez que el contenido está en el Data Lake, puede normalizarse y enriquecerse. Cuando se tiene un problema de negocio a resolver se puede solicitar al Data Lake los datos relacionados a esta cuestión. Inmediatamente obtenidos se analiza ese conjunto de datos más pequeño para ayudar a obtener una respuesta al problema.

2.1.6 Data Mart

Data Mart es un repositorio de datos orientado a un área específica de la empresa como puede ser recursos humanos, ventas, contabilidad, inventario,

etc. Está orientado a la consulta analítica de los datos de esa área, la distribución interna de los datos es clara y estos se encuentran estructurados en modelos dimensionales.

Generalmente, los datos con los que se alimenta provienen del Data Lake o el DW de la empresa, ya que estos contienen información de todos los departamentos de la organización. Entonces el Data Mart extraerá de ellos los datos relevantes para un área específica, con el fin de utilizar esta información en herramientas de visualización y reportes que ayuden a tomar decisiones respecto al área analizada.

2.1.7 Tipos de sistemas

2.1.7.1 OLTP (On-line Transaction Processing)

Como sus siglas lo indican OLTP hace referencia a los sistemas de procesamiento de transacciones en línea, es decir, son los sistemas que ayudan a las empresas con sus operaciones y transacciones cotidianas.

Abarca todas las operaciones que lleva a cabo la empresa donde se almacenan, modifican o eliminan datos. Generalmente, estos datos se procesan en bases de datos transaccionales que se usan en el día a día en la empresa para sistemas de inventario, contabilidad, fabricación, entre otros.

Uno de los principales factores a tener en cuenta en estos sistemas es que los datos presentan diferentes características en formato, procedencia, función, etc. Por lo que, generalmente, deben ser transformados para integrarse al DW.

Entre los OLTP más habituales de cualquier organización se pueden mencionar:

- Archivos de textos.
- Hipertextos.
- Hojas de cálculos.
- Informes diarios, semanales, mensuales, etc.

- Bases de datos transaccionales.

2.1.7.2 OLAP (OnLine Analytical Processing)

Son sistemas orientados al procesamiento analítico. Este análisis, usualmente, implica el procesamiento de grandes volúmenes de datos, de diferentes fuentes (sistemas OLTP), con el fin de extraer algún tipo de información relevante para la empresa, que luego se almacena en la mayoría de los casos en bases de datos multidimensionales como un Data Mart.

Estas bases de datos almacenan información en lo que se conoce como tablas de hechos y tablas de dimensiones las cuales proveen una estructura que permite, a través de la consulta a una estructura de datos determinada, tener acceso flexible a los datos. En consecuencia, se pueden explorar y analizar sus relaciones y obtener resultados que ayuden en la toma de decisiones respecto a un problema específico de la organización.

Los sistemas OLAP tienen diferentes implementaciones de arquitectura y entre las más utilizadas se encuentran:

- ROLAP: El almacén de datos se construye sobre un sistema de gestión de base de datos relacional.
- MOLAP: El almacén de datos no es relacional y utiliza una estructura de datos multidimensional.
- HOLAP: Combina las arquitecturas ROLAP Y MOLAP para brindar una solución que combine las mejores características de ambas.

2.1.8 Modelo Dimensional

El modelo dimensional es una técnica de modelado para bases de datos en las que se debe aplicar un enfoque analítico (comúnmente DW o Data Marts),

por lo que debe ser intuitiva para el usuario y estar optimizada para lograr buenas prestaciones en consultas.

Su función principal es estructurar y dimensionar los datos para que, de esta forma lograr, se pueda lograr una captura que muestre de qué manera se mide un proceso de negocio de la organización. Todos los datos que se pueden medir se denominan “hechos” y aquellos que no admiten medición otorgan un contexto a estas mediciones que se definen como “dimensiones”.

2.1.8.1 Tablas de hechos

Las tablas de hechos almacenan las medidas generadas por las actividades de negocio o eventos de la empresa, aunque no se deben diseñar alrededor de las preguntas a responder o reportes. Normalmente son numéricas y aditivas, pero también pueden existir semiaditivas (no pueden acumularse a través del tiempo) y no aditivas (no pueden sumarse).

Estas tablas concentran el 90% de los datos del modelo. Sus claves primarias (claves únicas y no nulas) son el conjunto de claves foráneas de las dimensiones.

Las tablas deben tener la mayor granularidad posible y ser uniformes según el proceso de negocio analizado. La granularidad hace referencia a la definición que brinda el negocio de la medida con que se representan los registros de las tablas.

Existen tres tipos de tablas de hechos más populares:

- Transaccionales: son el tipo más común. La granularidad es un registro por transacción y una vez insertado un nuevo registro este no sufre modificaciones.

- Captura periódica: cada cierto intervalo de tiempo predefinido se capturan fotos de los datos y se cargan en la tabla. Esta frecuencia suele ser diaria o semanal.

- Captura acumulativa: es el tipo menos utilizado y son usados para capturar medidas sobre un intervalo de tiempo indeterminado de procesos que tienen bien definido un inicio y fin. Se ingresan los registros al producirse el evento inicial y los eventos posteriores actualizan sobre el registro original incorporando datos adicionales.

2.1.8.2 Tablas de dimensiones

A diferencia de las tablas de hechos, que son claves y medidas numéricas, las dimensiones son campos descriptivos que le dan un contexto a lo que se quiere medir.

Sus atributos, habitualmente, se utilizan para realizar filtros en las consultas al modelo y para obtener descripciones o información adicional de los resultados obtenidos. Se dice que el poder de un DW es directamente proporcional a la calidad y profundidad que poseen los atributos dimensionales.

Los registros de las tablas de dimensiones se identifican por un campo de clave único (conocido como clave subrogada) el cual no se deriva del valor de clave de las fuentes origen, sino que se construye dentro del modelo a partir de una secuencia autogenerada. Los valores nulos generalmente se remplazan por textos (Desconocido, N/A).

Las dimensiones, en la mayoría de los casos, son SCD (lentamente cambiantes – Slowly Changing Dimension) y esto significa que sus atributos suelen ser más estáticos y estables con relación a una tabla de hechos. El modelo dimensional necesita capturar los cambios en el tiempo de los atributos dependiendo de las necesidades del negocio.

Existen tres tipos de SCD más utilizadas:

- Tipo 1: su comportamiento es de sobreescritura, los cambios de los valores de atributos son actualizados con los últimos cambios, se pierden los datos históricos y las dimensiones reflejan el último estado de los registros.

- Tipo 2: es la técnica más popular y poderosa para capturar los cambios en las dimensiones. Consiste en generar un nuevo registro con una propia clave subrogada cada vez que un valor de un atributo dimensional cambia. El registro con valor anterior y el actual siguen conservando la misma clave origen.

- Tipo 3: cuando se realiza un cambio de valor de un atributo, el valor anterior se desplaza a otra columna. Este tipo se utiliza con dimensiones de cambios frecuentes.

2.1.8.3 Tipos de modelamiento

Existen dos modelos que son los más utilizados: estrella y copo de nieve.

- Modelo estrella: es el más sencillo en cuanto a estructura, consta de una tabla de hechos central y varias dimensiones a su alrededor. La principal característica de este modelo es que solo existe una tabla de dimensiones para cada dimensión. Esto significa que no existen relaciones entre las tablas dimensionales, si no que desde la tabla de hechos se puede acceder directamente a los atributos de todas las tablas de dimensiones.

  Además, es el esquema más sencillo de interpretar y optimiza los tiempos de respuesta ante las consultas de los usuarios. Es soportado por casi todas las herramientas de consulta y análisis de datos.

- Modelo copo de nieve: es una variación o derivación del modelo estrella. En este esquema la tabla de hechos deja de ser la única relacionada con otras tablas, debido a que existen dimensiones que se relacionan con otras tablas dimensionales y no tienen una relación directa con la de hechos.

  Este modelo se construyó con la premisa de facilitar el mantenimiento de las dimensiones, sin embargo, al realizar consultas se

deben generar más relaciones de tablas y esto provoca que la extracción de datos sea más difícil y se vuelve aún más complejo mantener el modelo.

2.2 Tecnologías utilizadas

A continuación, se describirán en forma detallada cada una de las tecnologías implicadas en la realización del presente proyecto.

2.2.1 Apache Spark

Apache Spark es un motor de código abierto y análisis unificado para el procesamiento de datos a gran escala. Su función es la de proveer un entorno de trabajo y programación multilenguaje que utiliza clústeres con múltiples nodos, que permiten trabajar con grandes volúmenes de datos en forma distribuida, a gran velocidad y con un óptimo rendimiento.

Spark no almacena los datos, si no que se centra en el procesamiento de estos en memoria, lo que es mucho más rápido que las alternativas basadas en disco.

Proporciona APIs de múltiples lenguajes de programación como Java, Scala, Python y R, y un motor optimizado que admite gráficos de ejecución general. También es compatible con un amplio conjunto de herramientas de alto nivel como Spark SQL, que permite procesar datos estructurados y semi estructurados, MLlib para aprendizaje automático, GraphX para procesamiento de gráficos y Structured Streaming para procesamiento incremental y secuencias.

Arquitectura

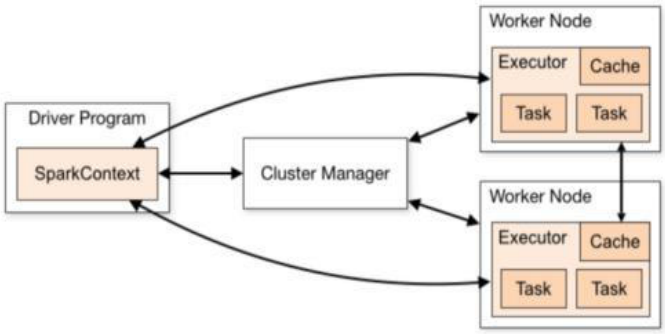


Figura 1 Arquitectura Apache Spark.
Recuperado de: https://www.instintoprogramador.com.mx/ 2019/06/una-introduccion-apache-spark-con-java.html

Posee una arquitectura maestro-esclavo con un administrador de clúster, tiene un solo nodo maestro y cualquier número de nodos esclavos o workes. Se identifica al maestro como la instancia que aloja el Driver Program y el Worker es la instancia que aloja a los ejecutores. En síntesis, su arquitectura se basa en tres componentes principales como los siguientes: el controlador, los ejecutores y el administrador de clúster.

Controlador (Driver): es el encargado de la ejecución de una aplicación Spark y mantiene todos los estados del clúster (el estado y las tareas de los ejecutores). En el programa de controlador de Spark reside en el nodo maestro y es quien crea y posee el contexto de Spark, por lo cual debe interactuar con el administrador del clúster para obtener recursos físicos e iniciar los ejecutores.

Ejecutores (Executors): son los procesos que realizan las tareas asignadas por el controlador, es decir, que poseen el código de aplicación que se desea implementar. Su principal responsabilidad es recibir las tareas asignadas por el controlador, ejecutarlas e informar sobre su estado (éxito o fracaso) y resultados. Cada aplicación Spark tiene sus propios procesos ejecutores independientes.

Administrador de Clúster (Clúster Manager): es el responsable de asignar los recursos necesarios para la ejecución de la aplicación. Cada vez que se ejecuta Spark, se solicitan recursos al administrador y él se encarga de distribuirlos y disponerlos en forma equitativa a cada uno de los nodos ejecutores para lograr el resultado esperado con la mayor velocidad y rendimiento posibles.

2.2.2 Azure Databricks

Azure Databricks es una plataforma de análisis de datos optimizada e integrada con los servicios en la nube de Azure, que nace de la colaboración de Apache, Databricks y Microsoft. Es una plataforma creada por el fundador de Spark, que integra este servicio mediante el cual se pueden lanzar potentes algoritmos analíticos sobre grandes cantidades de datos y en tiempo real en la nube.

Respecto a su arquitectura propone dos entornos para trabajar con datos: Azure SQL Analytics y Azure Workspace. En ambos se permite gestionar los entornos de Apache Spark y hacerlos evolucionar según las necesidades, aumentar la capacidad de cómputo de los clústeres, automatiza su inicio y apagado, lo cual simplifica el despliegue y acelera la instalación y configuración de los entornos de desarrollo. También cuenta con una opción sin servidor donde se puede ignorar la complejidad de la infraestructura y acceder directamente al uso del servicio.

Los entornos de Databricks permiten un desarrollo colaborativo mediante espacios de trabajo interactivos llamados notebooks. En estos entornos los desarrolladores cuentan con un perfil propio dentro de la aplicación y pueden acceder a distintas notebooks en forma colaborativa junto con sus compañeros. Cada notebook tiene un uso restringido y deben brindarse los permisos necesarios a cada perfil (solo lectura, modificación y administrador).

El clúster Databricks tiene dos modos de funcionamiento: estándar y alta concurrencia. El clúster de alta concurrencia es compatible con los idiomas de programación Python, R y SQL, mientras que el clúster estándar es compatible con Scala, Java, Python, R y SQL. Además, se integra directamente con GitHub para ayudar al trabajo colaborativo y con otras aplicaciones de análisis de datos y reporting como Power Bi y Tableau.

2.2.3 ADF (Azure Data Factory)

ADF es el servicio ETL en la nube de Azure para la integración y transformación de datos sin servidor de escalabilidad horizontal. Además, permite crear flujos de trabajo a fin de coordinar el movimiento y extracciones de los datos de múltiples orígenes.

Con Azure Data Factory se pueden crear y programar flujos de trabajo basados en datos (llamados canalizaciones), que pueden ingerir estos desde distintas bases y almacenes de datos. También permite crear procesos ETL complejos, que transformen datos visualmente mediante servicios de proceso, como Azure HDInsight Hadoop, Azure Databricks y Azure SQL Database.

Básicamente provee de las herramientas necesarias para ejecutar todo el proceso ETL de los datos. Abarca desde la conexión con los distintos sistemas orígenes (bases de datos, archivos de texto, hojas de cálculo, APIs, etc.), la extracción de dichos datos, la copia a otro almacén de datos en la nube (generalmente Azure Blob Storage), la transformación de dichos datos según las necesidades de negocio (Azure Databricks) y la publicación de estas transformaciones y nuevos modelos en diferentes almacenes de datos (DW, Data Lake, entre otros).

2.2.4 SQL (Structured Query Language)

SQL es un lenguaje de consulta destinado al dominio y manejo de la información almacenada en bases de datos relacionales. La programación o

armado de consultas en SQL sirve para almacenar, manipular, modificar y recuperar datos.

Una de las características principales es el manejo del álgebra relacional, que permite hacer cálculos avanzados y otorga un alcance que abarca la inserción de datos, consultas, actualizaciones y borrado. Como también la creación y modificación de esquemas y el control de acceso a los datos.

Oracle PL/SQL: en el caso de la plataforma analítica a migrar esta se encuentra montada sobre una base de datos Oracle. Por este motivo, se utiliza el lenguaje PL/SQL, que es un lenguaje de procedimiento que permite utilizar todas las sentencias SQL variando en ciertos aspectos de sintaxis y agregando nuevas especificaciones y funcionalidades propias de esta base de datos.

2.2.5 Python

Python es un lenguaje de programación de alto nivel, interpretado (no necesita compilarse para su ejecución), multiplataforma y de código abierto que se utiliza tanto para desarrollo web, creación de software, procesamiento y análisis de datos, entre muchos otros propósitos. Se basa en los lenguajes C y C++ y tiene sus raíces en el sistema operativo UNIX.

También es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Python se ha vuelto muy popular en los últimos años debido a su sencillez y amplias posibilidades de desarrollo, ya que facilita trabajar con inteligencia artificial, Big Data, Machine Learning y Data Science, que son campos dentro de la informática que están siendo muy explotados en los últimos tiempos.

Como se mencionó este lenguaje se utiliza mucho dentro de las áreas de BI debido a su simplicidad y su gran número de bibliotecas de procesamiento de información, lo que lo transforma en el lenguaje ideal a la hora de analizar y gestionar una gran cantidad de datos.

2.2.6 GeneXus

GeneXus es una plataforma de desarrollo de software que simplifica y automatiza las tareas de crear y mantener aplicaciones del tipo empresarial. Permite crear aplicaciones para Web, Windows, dispositivos móviles y plataformas legadas, automáticamente generando y conectando todas las funcionalidades, servicios y bases de datos que son requeridos.

A partir del modelado del sistema deseado, GeneXus crea automáticamente la base de datos, el código de las aplicaciones, la interfaz de usuario para el cliente y los servicios necesarios del lado del servidor. Estos modelos que se crean están basados en el conocimiento que los usuarios y clientes tienen del negocio. Además, les permite concentrarse en sus necesidades empresariales para que GeneXus se encargue de generar todo el código para resolverlo en forma técnica y tecnológica.

La plataforma permite crear aplicaciones en diferentes lenguajes de programación, entre los que se incluyen C#, COBOL, Java, Objective-C, RPG, Ruby, Visual Basic y Visual FoxPro.2. También soporta múltiples sistemas gestores de bases de datos, entre los más populares se pueden mencionar SQL Server, Oracle, DB2, Informix, PostgreSQL y MySQL.

Una de las principales ventajas de esta tecnología es su capacidad de manejar datos (incluso datos en producción), organizarlos y moverlos a través de plataformas y llevarlos a cualquier dispositivo.

2.2.7 Power Bi

Power Bi es un software de inteligencia de negocios que se utiliza principalmente para crear tableros de mando, que facilitan la toma de decisiones. Esta información se puede actualizar de manera automatizada o manual y permite la compartición de los informes mediante la propia herramienta.

Power BI se puede conectar a una gran variedad de fuentes de datos (desde hojas de cálculo hasta bases de datos y aplicaciones tanto en la nube como en los servidores de las empresas). Una vez conectada recopila los datos y los procesa, convirtiéndolos en información relevante para la toma de decisiones, generalmente utilizando gráficos y tablas visualmente convincentes y fáciles de procesar. Esto permite a los usuarios generar y compartir instantáneas claras y útiles de lo que está sucediendo en su negocio.

Power BI tiene varios componentes entre los que se destacan:

- Power BI Desktop: aplicación de escritorio para diseñar informes y cuadros de mando que luego se pueden publicar y subir a la nube.
- Power BI Service: servicio en la nube basado en SaaS (software como servicio).
- Power Bi Mobile: aplicaciones móviles de Power BI para dispositivos Android y iOS.
- Power Bi Gateway: utilizado para enlazar y sincronizar datos externos dentro y fuera de Power BI.
- Power BI Embedded: es la API REST de Power BI que se puede utilizar para crear informes dentro de otras aplicaciones o portales para distribuirlos a usuarios que no tienen cuenta de Power BI.
- Power BI Report Server: lo utilizan organizaciones que no quieren usar la nube y en su lugar recurren a un servidor propio dentro de la empresa.

2.2.8 Cognos BI

Cognos Bi es una plataforma desarrollada por IBM para la inteligencia de negocios, que comprende una amplia gama de funcionalidades que ayudan a las organizaciones a interpretar los datos de su empresa. Todo el personal de su organización puede utilizar IBM Cognos BI para ver o crear informes empresariales, analizar datos, o supervisar eventos y métricas, a fin de contribuir con una mayor eficacia a la toma de decisiones empresariales.

Integra diferentes componentes y actividades de inteligencia empresarial en una solución basada en Web, que le permiten visualizar, explorar, analizar y transformar sus datos en conocimiento mediante herramientas como:

- Reportes gráficos navegables.
- Tableros de mando para monitorear los indicadores claves del negocio en tiempo real.
- Simulación de escenarios para intentar predecir posibles sucesos que afecten al negocio.
- Movilidad que permite acceder a la información desde cualquier dispositivo móvil.

2.2.9 GitHub

GitHub es una plataforma de alojamiento, propiedad de Microsoft, que ofrece la posibilidad de crear repositorios de código y guardarlos en la nube de forma segura, usando un sistema de control de versiones, llamado Git.

Un sistema de control de versiones es ese sistema con el cual los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las versiones que sacan de sus aplicaciones. Así, al tener copias de cada

versión de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

GitHub permite comparar el código de un archivo para ver las diferencias entre las versiones, restaurar versiones antiguas si surge algún error en la actual y fusionar los cambios de distintas versiones. También permite crear diferentes ramas de los proyectos lo que facilita el trabajo sin afectar lo ya realizado o el trabajo de otros colaboradores.

Por otra parte, es una de las plataformas más populares para la creación de trabajos colaborativos. Una de las razones es que no censura ni discrimina los lenguajes de programación existentes, los acepta a todos sin inconveniente, por lo que le facilita el trabajo a la gran mayoría de los desarrolladores.

2.2.10 Azure Devops

Azure Devops es un conjunto de herramientas y servicios que ayudan en la administración del ciclo de vida de los proyectos de desarrollo de software. Cuenta con diferentes herramientas que permiten la gestión de proyectos utilizando una metodología de desarrollo ágil. En el caso de este proyecto se utiliza la herramienta Azure Boards para gestionar los proyectos de datos de la organización.

Azure Boards permite realizar un seguimiento de cada una de las tareas, que los miembros del equipo de desarrollo tienen a su cargo mediante la utilización de tableros Kanban, en ella se puede registrar el trabajo pendiente, el que se encuentra en proceso y el finalizado. También provee herramientas para realizar gráficas en tiempo real que ayudan a entender de forma más rápida lo que está sucediendo con las tareas durante el desarrollo del sprint (ciclo de ejecución de tareas de un proyecto que va de dos a cuatro semanas) en curso.

También está optimizado para trabajar con metodologías ágiles como Scrum mediante el uso de herramientas de planeamiento y paneles integrados

para ayudar a los equipos a ejecutar sprints y celebrar reuniones breves o de planeamiento.

2.3 Análisis y requerimientos

Para llevar a cabo el desarrollo de este proyecto en primer lugar fue necesario realizar un análisis exhaustivo y detallado del sistema actual, con el fin de definir los requerimientos funcionales y no funcionales, las tecnologías a utilizar y el plan de ejecución necesario para su implementación.

Actualmente el sistema se compone de un Data Warehouse implementado en base de datos Oracle llamada BIPROD, que tiene diferentes esquemas y procedimientos almacenados, en los cuales se encuentran desarrollados diferentes modelos dimensionales. En este proyecto se realizará la migración de los modelos correspondientes al sistema OLAP llamado gestión comercial, desde este modelo Cognos BI funciona como una plataforma que centraliza y transforma los datos de BIPROD, para armar los reportes y sus visualizaciones.

Cognos es utilizado por diversas áreas y departamentos para construir sus reportes diarios resultando ser una herramienta central para la gestión de la información.

Cabe destacar que, actualmente, la herramienta tiene una performance bastante baja en cuanto a eficiencia y productividad y no cuenta con soporte técnico lo que representa un riesgo alto para la compañía. Esto se debe principalmente a que tanto la base de datos BIPROD como el desarrollo de reportes en Cognos se realizaron hace muchos años y no han sido actualizados ni mantenidos por la empresa.

2.3.1 Arquitectura inicial

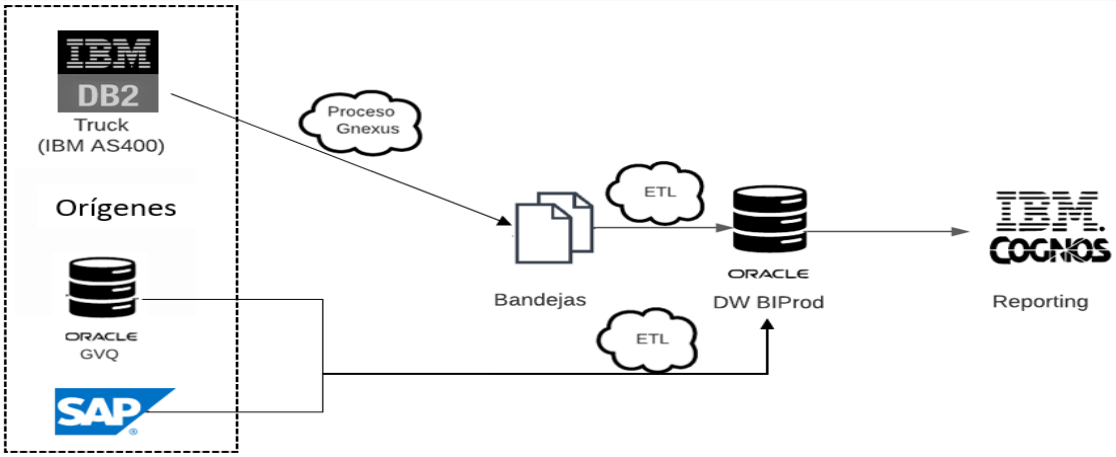


Figura 2 Arquitectura inicial de la plataforma BI.
Fuente: Elaboración propia (basada en la práctica).

Como se puede observar en la Figura 2, la arquitectura actual se compone de tres sistemas orígenes diferentes, que se describen a continuación:

Truck: sistema transaccional que registra la operatoria de ventas de la empresa, logística, RR.HH., etc. Su información es alojada en una base de datos DB2 (AS400). La historia data de agosto de 1999 a la fecha y se encuentra particionada en distintos esquemas y servidores.

Los esquemas de desarrollo contienen los datos históricos productivos.

Ambiente	Esquema	País
Productivo	TRK35NULL	Argentina
Productivo	TRKNULLCNB	Bolivia
Productivo	TRKNULLCPA	Paraguay
Productivo	TRKNULLFNC	Uruguay

Tabla 1 Descripción de ambientes y esquemas Truck. Fuente: Elaboración propia (basada en la práctica).

SAP: sistema ERP (Enterprise Resource Planning), que contiene la operatoria del canal Distribuidores. En este ambiente se consumen las siguientes tablas MABIVT (Ventas) y MABID2 (Apertura por tipificación de descuento).

Los esquemas y ambientes a consultar son:

Ambiente	Esquema	País
Productivo	INTERTRK	Argentina
Productivo	INTERCPA	Paraguay
Productivo	INTERFNC	Uruguay

Tabla 2 Descripción de ambientes y esquemas SAP. Fuente: Elaboración propia (basada en la práctica).

Bolivia no contiene datos que provengan de este sistema.

GVQ: base de datos desarrollada en Oracle llamada POWA, que contiene información sobre los objetivos de ventas mensuales que desea cumplir la compañía. Esta información se encuentra dentro el esquema GVQ y las tablas DLK_OBJ_MES y DLK_OBJ_PESOS_DIAS.

Se compone de los siguientes esquemas:

Ambiente	Esquema	País
Productivo	AR_PROD_GVQ	Argentina
Productivo	BO_PROD_GVQ	Bolivia
Productivo	PY_PROD_GVQ	Paraguay
Productivo	UY_PROD_GVQ	Uruguay

Tabla 3 Descripción de ambientes y esquemas GVQ. Fuente: Elaboración propia (basada en la práctica).

De estas tres fuentes se obtiene toda la información relevante para el sistema OLAP de gestión comercial.

De los tres sistemas, SAP y GVQ son consumidos en forma directa desde la base de datos Oracle BIPROD (nuestro data warehouse). Mientras que el sistema Truck es procesado por GeneXus para aplicarle un proceso ETL, con el fin de extraer, transformar y disponibilizar los datos origen en un archivo que también es consumido por el DW antes mencionado.

En este punto la información de los tres sistemas fuentes se encuentra disponible para ser consumida por nuestra base de datos BIPROD. Sin embargo, antes de consumir dichos datos se les aplican una serie de transformaciones mediante procedimientos almacenados en el DW, con el objetivo de generar el modelo dimensional necesario para los reportes.

Se crean múltiples tablas combinando datos de los diferentes sistemas origen. Estas tablas son las que se mencionaron anteriormente en la explicación del modelo dimensional, se crean las tablas de hechos que contienen las métricas de negocio y las tablas dimensionales que le van a dar el contexto necesario a estas métricas.

Finalmente se tiene el modelo OLAP de gestión comercial implementado en el Data Warehouse de BIPROD y esta información puede ser consumida por IBM Cognos Report Studio. Por último, para generar los reportes utilizando la herramienta se aplican los filtros y ajustes correspondientes, necesarios para obtener la información que ayude a los grupos gerenciales en la toma de decisiones de la compañía.

2.3.2 Arquitectura propuesta

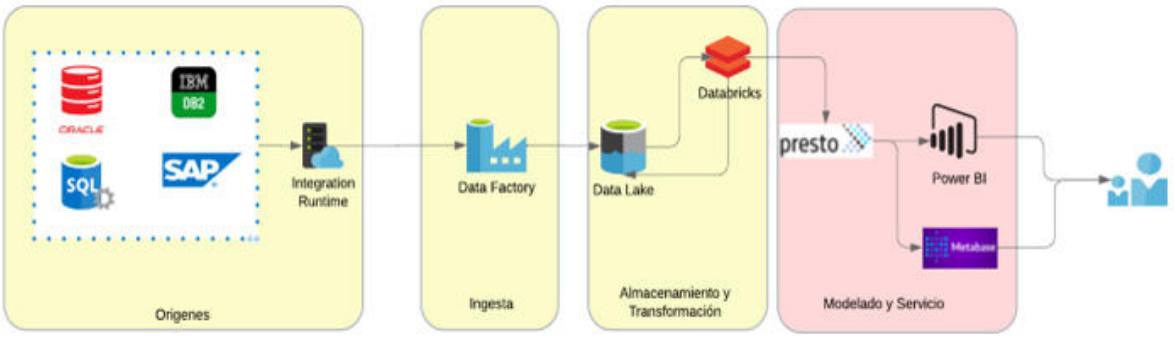


Figura 3 Arquitectura propuesta luego de la migración de la plataforma BI.
Fuente: Elaboración propia (basada en la práctica).

Como se ilustra en la Figura 3 la arquitectura de solución cuenta con los mismos tres sistemas orígenes descriptos anteriormente. Esto se debe a que el proyecto de PPS se trata de una migración de plataforma, por lo cual los sistemas orígenes desde donde se extraen los datos del modelo dimensional son los mismos. Sin embargo, en esta arquitectura los datos de los sistemas orígenes no sufren de ningún tipo de transformación antes de realizar el procedimiento de ingesta y se replican directamente al Datalake montado en la

nube de Azure (se realiza un procedimiento ELT). Esta réplica se realiza utilizando Azure Data Factory.

El servicio de ADF se conecta directamente a las bases de datos orígenes y genera la réplica e ingesta de las tablas al DL. Este proceso se explicará detalladamente más adelante.

Una vez generada la ingesta de todas las tablas desde los sistemas orígenes se tiene toda la información necesaria en el datalake. Entonces no hay impedimentos para comenzar a migrar el modelo de gestión comercial y para ello se usa Databricks con el fin de transformar todos los datos provenientes de los distintos sistemas orígenes y generar las tablas de hechos y dimensiones correspondientes.

Finalmente, una vez desarrollado el modelo dimensional con las distintas tablas, se genera una réplica de estas utilizando Presto (motor de consulta SQL distribuido de alto rendimiento para grandes datos) al cual se accede por la implementación de una librería instalada en Databricks. Mediante Presto se replican los datos a Metabase (herramienta de software libre que permite crear cuadros de mando a partir de múltiples fuentes de datos).

Por último, los visualizadores se conectan a Metabase con el fin de cargar las tablas modeladas a Power BI, donde aplican los últimos filtros y ajustes para generar los reportes a utilizar por el negocio.

2.3.3 Organización del datalake

Figura 4 Organización del datalake.
Fuente: Elaboración propia (basada en la práctica).

En primer lugar, se hace la extracción de los datos del sistema fuente y estos se almacenan en una zona a la que llamamos **"Pre Landing Zone" (PZ)** sin ningún tipo de transformación, es decir, en esta zona se almacenan los datos crudos tal como se extrajeron del origen.

Luego de esto los datos pasan a una segunda instancia o zona conocida como **History Zone (HZ),** la que conserva el historial de versiones de cada uno de los registros de las tablas del sistema fuente.

Por último, los datos pasan a la **Consume Zone (CZ)** donde los datos aparecen en su última versión, es decir, los datos están actualizados y es la zona en la cual generalmente trabajan los ingenieros de datos.

Como se observa en la Figura 4 en el Datalake y las zonas se trabaja con dos formatos de archivos: Avro y Parquet.

2.3.3.1 Formato Avro

Avro es un formato utilizado para la serialización de datos, que proporciona estructuras de datos complejas, con un formato binario, compacto y rápido.

La base fundamental del formato son los esquemas. Siempre que se lee un formato Avro está presente el esquema con el que han sido escritos, esto permite aumentar el rendimiento al escribir los datos, haciendo la serialización rápida y viable en espacio.

Se utiliza en la PZ ya que es la zona donde se escriben los datos desde el origen y este tipo de archivos se caracteriza por su gran velocidad de escritura.

2.3.3.2 Formato Parquet

Parquet es un formato de almacenamiento en columnas para Hadoop. Fue creado para poder disponer de un formato de compresión y codificación eficiente.

El formato de Parquet está compuesto por tres piezas:

- Row group: es un conjunto de filas en formato columnar.
- Column chunk: son los datos de una columna en un grupo.
- Page: es donde finalmente se almacenan los datos y debe ser lo suficientemente grande para que la compresión sea eficiente.

Este formato permite procesar consultas sobre datos de manera más rápida al estar en un formato columnar. Por este motivo, se utiliza en la CZ ya que es allí donde se aplican las transformaciones necesarias a los datos y se generan los modelos que se adaptan a las necesidades del negocio.

2.4 Ola 1: Relevamiento e ingesta de fuentes de datos

En el punto anterior, por una parte, se mencionaron las distintas bases de datos y sistemas fuentes a utilizar en este proyecto. Por otra parte, en esta

Ola se describirá en forma más detallada el procedimiento de relevamiento e ingesta de la información proveniente de los diferentes sistemas OLTP.

Todo el proceso de ingesta se realiza utilizando ADF, el cual está integrado con GitHub para que cada ingeniero de datos pueda trabajar libremente sobre una su propia rama de desarrollo sin afectar las tareas de los demás ingenieros.

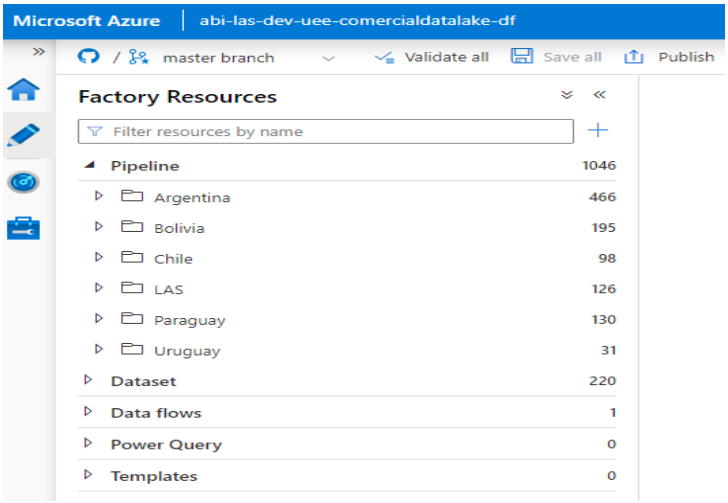


Figura 5 Organización de Azure Data Factory.
Fuente: Elaboración propia (basada en la práctica).

Como se observa en la Figura 3 en la parte superior se encuentra la integración con Git (en este ejemplo se utiliza la rama master) desde la cual se puede elegir en qué rama trabajar, así como también crear nuevas ramas de desarrollo.

Debajo de esto se encuentran las diferentes herramientas a utilizar en ADF, en primer lugar, está la herramienta de canalización o pipeline mediante la cual realiza todo el proceso de extracción, transformación y carga de los datos a los sistemas orígenes, es decir, se realiza todo el proceso ETL o ELT dependiendo el caso.

Los procesos de ingesta se encuentran divididos por país (Argentina, Bolivia, Chile, Paraguay y Uruguay), por lo cual se tienen distintas carpetas para

cada uno de ellos y una carpeta general (LAS) donde se encuentran los pipelines, que contemplan a todos los países.

A su vez, dentro de cada país, hay subcarpetas por cada sistema origen a replicar, para esta migración se utilizan los sistemas orígenes Truck, SAP y GVQ. Entonces, estas son las carpetas que interesan para el proyecto y en las cuales se entrará más a detalle a continuación.

En este informe se realizará la explicación del proceso de ingesta para la fuente de origen de datos Truck y el país Argentina. Para los otros sistemas fuente y los demás países el proceso de ingesta es similar, excepto por algunas pequeñas consideraciones que se detallarán cuando corresponda.

La explicación se centra en Truck, debido a que es nuestra principal fuente de información para el proyecto; es decir, la mayoría de las tablas que se van a replicar son de este sistema y se complementa con algunas tablas de los otros dos (SAP y GVQ).

2.4.1 Proceso de ingesta para Truck Argentina

El proceso de ingesta de tablas del sistema Truck se divide principalmente en dos tipos:

- Ingesta de tablas maestro
- Ingesta de tablas incrementales

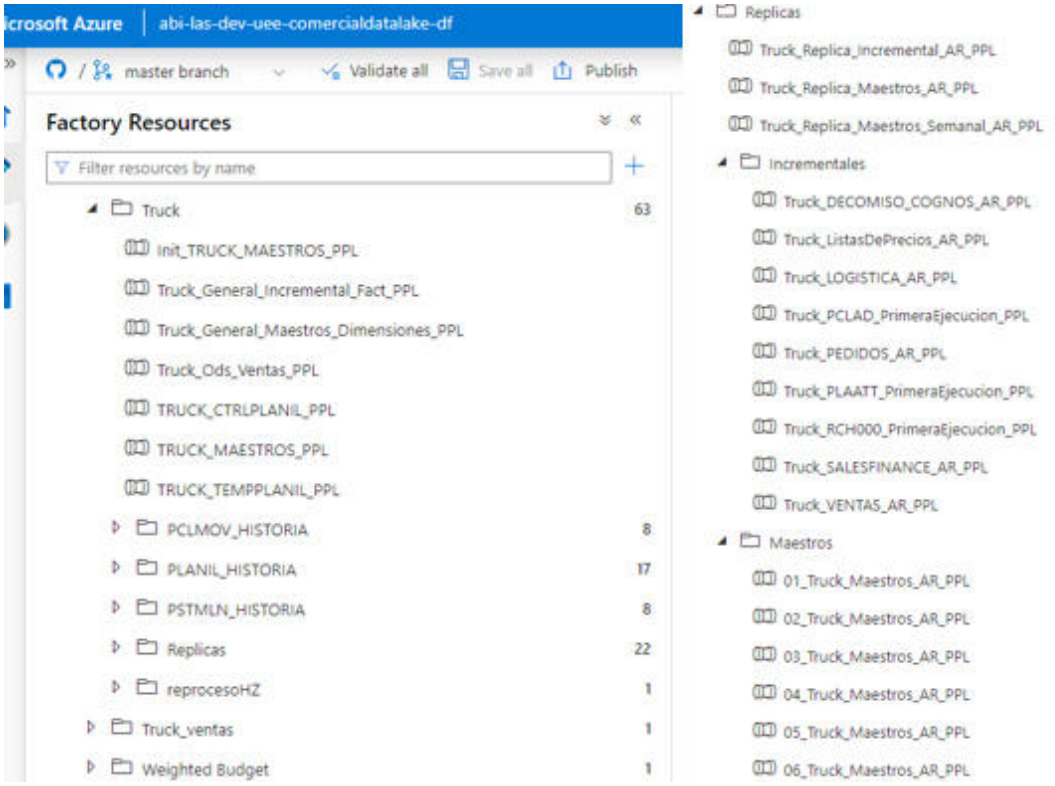


Figura 6 División del proceso de ingesta en ADF.
Fuente: Elaboración propia (basada en la práctica).

Como se puede observar en la Figura 6 hay distintos directorios y pipelines en ADF para las ingestas incrementales y las ingestas de tablas maestro. A su vez, hay un pipeline general para las réplicas incrementales (Truck_Replica_Incremental_AR_PPL) y un pipeline general para las réplicas maestros (Truck_Replica_Maestros_AR_PPL), este pipeline general se encarga de ejecutar cada uno de los pipes que se encuentran en los directorios mencionados.

Tablas maestro: son aquellas tablas donde se lee y reprocesa toda la información que contienen (en general esto se realiza todos los días, en el mismo horario) para replicar dichos datos en el sistema de destino. Es decir, todos los días se lee absolutamente toda la información de la tabla y se sobrescriben los datos en el sistema destino.

Tablas incrementales: se refiere a aquellas tablas donde se lee y replica toda la información solo la primera vez y luego se replican solo las novedades basándose en un campo de control que contiene la tabla. Si la tabla no contiene este campo se debe implementar una solución lógica que simule el mismo comportamiento y solo obtenga las novedades de dicha tabla. En síntesis, la primera vez se lee toda la información de las tablas y los días subsiguientes solo se replican los datos nuevos o modificados.

Esta diferencia de réplicas se basa en el volumen de datos de las tablas, si la tabla tiene un volumen de datos excesivo, se debe aplicar una ingesta incremental, ya que es muy costoso leer y replicar toda la información todos los días. En cambio, si la tabla tiene un volumen de datos aceptable para nuestro sistema se realiza una ingesta maestro. En este proyecto se realizan ingestas incrementales cuando el volumen de la tabla supera el millón de registros y una ingesta maestro cuando el volumen de datos es inferior a ese número. Esta lógica aplica tanto al sistema Truck como a los demás sistemas orígenes implicados en este desarrollo.

2.4.1.1 Ingesta maestro

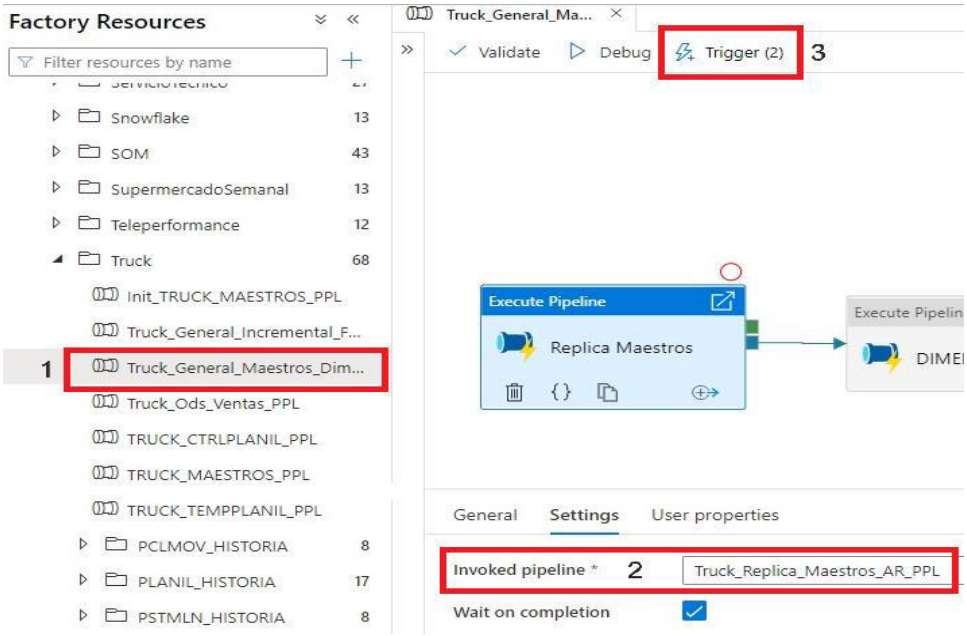


Figura 7 Descripción del proceso de ingesta de tablas maestro.
Fuente: Elaboración propia (basada en la práctica).

Se pueden observar tres puntos importantes en la ejecución de este proceso, los cuales se encuentra numerados en la Figura 7 y se describen a continuación:

1. Este es el pipeline que se observa en la Figura 7, contiene la ejecución de dos pipelines conectados en forma secuencial. El primero (Replica Maestros) se encarga de ejecutar la réplica de tablas maestro para Argentina y el siguiente las dimensiones. Esta explicación se centra en el primer pipeline debido a que este apartado no hace referencias a modelado dimensional.
2. Se puede notar que el pipeline Replica Maestros tiene como función invocar a otro pipeline llamado “Truck_Replica_Maestros_AR_PPL”.
3. En esta sección de ADF se programan los triggers (disparadores que ejecutan ciertas acciones en forma automática) que se utilizan para ejecutar los pipelines de Azure Data Factory.

El pipeline Truck_Replica_Maestros_AR_PPL, que se mencionó en el punto 2, tiene como única función invocar a otros PPLS (Pipelines), que contienen la información de las diferentes tablas maestros a replicar. Para el caso de Argentina existen seis pipelines maestros (Figura 6**).**

Dentro de cada maestro existe un número de tablas de Truck a replicar. Se decidió dividirlo en distintos pipelines para acelerar el proceso de ejecución y en caso de error poder volver a ejecutar un número menor de réplicas, en menor tiempo.

Estos maestros también están conectados en forma secuencial, por lo cual si falla el maestro 1 es necesario volver a ejecutar todo el pipeline desde el inicio. Pero si el error ocurre en el maestro 5, solo es necesario volver a ejecutar los maestros cinco y seis.

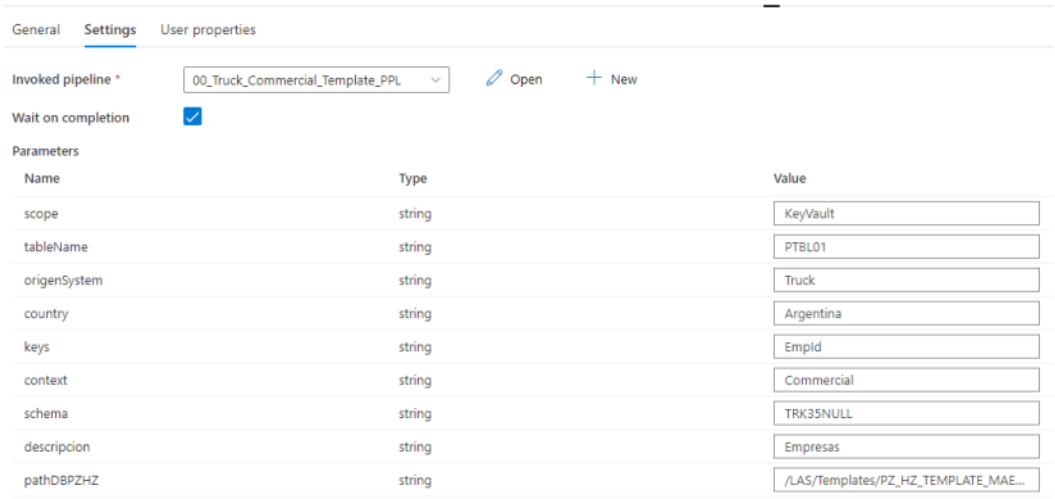


Figura 8 Composición de pipeline maestro.
Fuente: Elaboración propia (basada en la práctica).

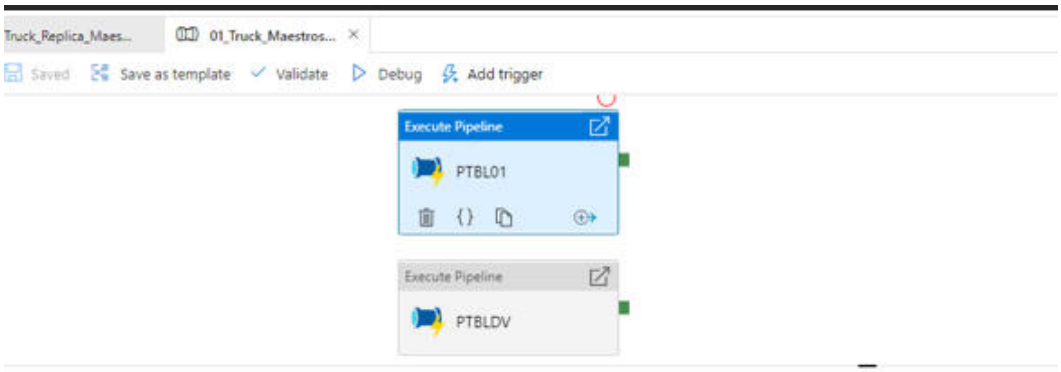


Figura 9 Parámetros necesarios en tablas maestro.
Fuente: Elaboración propia (basada en la práctica).

En la figura 8 se puede ver cómo está compuesto el Maestros 1 y como ya se mencionó tiene múltiples pipelines en su interior (uno por cada tabla Truck a replicar). Si se hace clic en uno de estos pipelines (en este caso el correspondiente a la tabla PTBL01), se puede observar que este invoca otro pipeline (“00_Truck_Commercial_Template_PPL”) del cual se hablará más

adelante. También muestra los parámetros definidos para la ejecución del PPL (Figura 9) y, por lo tanto, la réplica de la tabla, los cuales se detallan a continuación:

- Scope: se utiliza para cifrar las claves o credenciales de los distintos sistemas a los que desea acceder el PPL (en este caso Truck). Para este cifrado se usa Azure Key Vault en notebooks de Azure Databricks.
- tableName: es el nombre de la tabla del sistema Truck que se desea replicar, en este ejemplo el nombre es PTBL01, pero esto varía según cada tabla.
- origenSystem: nombre del sistema origen desde el cual se lleva a cabo las réplicas de tablas.
- country: nombre del país al cual corresponde la ingesta de tablas.
- keys: son las claves primarias de la tabla a replicar.
- context: se refiere al ámbito donde se van a replicar y existir las tablas.
- schema: es el esquema donde se encuentra la tabla en la base de datos origen.
- description: descripción de la tabla a replicar

- pathDBPZHZ: es la ruta donde se encuentra el template de la notebook de Databricks, que se utiliza para la réplica de las tablas de Truck.

Luego de esto, como se mencionó anteriormente, este pipeline invoca a otro llamado “00_Truck_Commercial_Template_PPL”, el cual en su interior contiene la lógica que se utiliza para la réplica de tablas del sistema origen al DL, la cual se aprecia en la siguiente imagen.

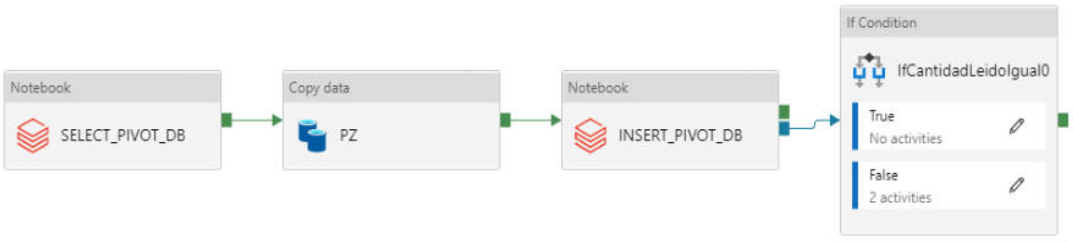


Figura 10 Proceso de réplica de tablas maestro.
Fuente: Elaboración propia (basada en la práctica).

Se puede ver que dicho PPL, en primer lugar, ejecuta una notebook de Databricks (SELECT_PIVOT_DB), luego un elemento de ADF llamado Copy Data (el más importante del proceso por lo que se detallará su funcionamiento más adelante), otra notebook (INSERT_PIVOT_DB) y otra herramienta de ADF, que básicamente es un condicional que indica si se leyeron datos de la tabla origen o no, y en base a ello aplica cierta lógica que se explicará a la brevedad.

Para entender el funcionamiento de la notebook SELECT_PIVOT_DB es necesario aclarar el concepto de campo pivot.

Campo Pivot

Para cada tabla y archivo a mover al datalake se definirá si aplica un campo pivot. Este campo es el que determina qué registros son los que cargarán en cada ejecución del pipeline. Es decir, es el campo que indica cuáles son los nuevos registros desde la última vez que se replicó. Se admiten pivots del tipo datetime e integer y en general este campo va a corresponder a un identificador incremental o campo de fecha de auditoría en la tabla origen.

Select Pivot

Una vez aclarado este concepto se puede continuar con la explicación de la función de la actividad “SELECT_PIVOT_DB”. Su función es llamar a una notebook de Databricks, que se encarga de devolver un archivo JSON que contiene la consulta a ser usada dentro del copy data y el último valor del pivot.

Esta notebook a su vez consulta una tabla que almacena la metadata de las réplicas. Dicha tabla se configura dentro de un sitio web creado mediante Microsoft SharePoint y es accedida por la notebook de Databricks según los siguientes parámetros.

TABLE_CONFIG_PIVOT	
NOMBRE COLUMNA	**DESCRIPCION**
ID_CONF_TABLE	Identificador auto incremental de la tabla
NEGOCIO	Nombre del negocio

TABLA_NOMBRE	Nombre de la tabla
SCHEMA	Esquema de la tabla
TIPO_CARGA	Tipo de carga a realizar
QUERY	Consulta a utilizar para el tipo de carga 0
FILE_DIR_CZ	Directorio donde se encuentra dentro de la consume zone
FILE_DIR_HZ	Directorio donde se encuentra dentro de la history zone
FILE_DIR_PZ	Directorio donde se encuentra dentro de la prelanding zone
ESTADO	Si la tabla esta activa para replicar o no
ORIGEN	Nombre del sistema origen
PAIS	Nombre del país
FECHA_INSERT_UPDATE	Fecha de creación o modificación del registro
TIPO_CAMPO_PIVOT	Tipo del campo pivot
DESCRIPCION	Texto descriptivo de la tabla
QUERY_REPROCESO	Consulta a utilizar para el tipo de carga 1
TABLA_PIVOT	Nombre de la tabla que contiene al campo pivot
CAMPO_PIVOT	Nombre del campo para usar de pivot

Tabla 4 Configuración de Select Pivot. Fuente: Elaboración propia (basada en la práctica).

Debe existir una entrada por cada tabla o archivo a ser replicado en el datalake. La inserción o modificación de los registros en las tablas es manual utilizando el sitio web de SharePoint y es condición necesaria que antes de replicar una tabla o archivo este se encuentre registrado en dicho sitio.

Consideraciones importantes

TIPO_CARGA: los valores a tomar serán 0 si es una carga normal, 1 si es un reproceso y 3 si es una carga histórica.

ESTADO: los valores a tomar serán 1 que indica que la tabla se puede replicar o 0 que indica lo contrario.

QUERY: este campo contiene la consulta a ser utilizada literalmente con una salvedad, todos los valores de los pivot se tienen que escribir como valorPivotn donde n es un entero mayor o igual a 0.

Por ejemplo:

"SELECT * FROM SCHEMA.TABLA WHERE (CAMPOA = valorPivot0 AND CAMPOB = valorPivot1) OR CAMPOC = valorPivot2 "

TIPO_CAMPO_PIVOT: contiene el tipo de campo del pivot y este debe ser DATETIME o INT. Si hay más de un valor de pivot se deben separar por un pipe siempre y cuando corresponda a otra tabla si no por una coma. Siguiendo el ejemplo anterior:

"DATETIME, INT, DATETIME"

Lo que indica esto es que el primer valor del pivot es una fecha, el segundo es un entero y el tercero es otra fecha. Si tuviera más pivots correspondiente a otras tablas, el ejemplo sería:

"DATETIME, INT, DATETIME | DATETIME | INT"

Esto muestra que los primeros tres corresponden a la primera tabla, el siguiente a otra tabla y el siguiente a una última diferente.

TABLA_PIVOT: tiene el nombre de las tablas que contienen los pivot. Como mínimo tiene lo que se escribió en el campo TABLA_NOMBRE y en caso de existir más tablas se van a separar por un pipe.

Por ejemplo:

"TABLA1 | TABLA2 | TABLA3"

TABLA1 es el valor que tiene el campo TABLA_NOMBRE, TABLA2 y TABLA3 son las tablas de donde se saca el valor del pivot. Estas podrían ser

tablas que se joinean (se refiere a unir dos tablas diferentes buscando un campo de coincidencia en ambas) con la tabla a replicar.

Por ejemplo:

"SELECT * FROM SCHEMA.TABLA1 INNER JOIN SCHEMA.TABLA2 ON TABLA1.CAMPO = TABLA2.CAMPO LEFT JOIN SCHEMA.TABLA3. CAMPO = TABLA1.CAMPO"

CAMPO_PIVOT: Tiene el nombre de los campos a usar de pivot. Si hay más de un campo, estos se van a separar por coma en caso de corresponder a la misma tabla o por un pipe si corresponden a otra.

Por ejemplo:

"CAMPOA, CAMPOB | CAMPOC | CAMPOD"

Esto muestra que CAMPOA y CAMPOB corresponden a la TABLA1, CAMPOC a la TABLA2 y CAMPOD a la TABLA3. Es importante que estén ordenados según lo indica el campo QUERY, es decir, valorPivot0 corresponde al CAMPOA, valorPivot1 corresponde al CAMPOB y así sucesivamente. El orden de TABLA_PIVOT y CAMPO_PIVOT también es importante y deben coincidir al igual que el TIPO_CAMPO_PIVOT.

Copy Data

Dentro de esta actividad se hará hincapié en los dos procesos más importantes: Source y Sink.

Source: en el cual se configura la conexión al sistema origen (en este caso Truck) y los parámetros necesarios para la lectura de las tablas. En síntesis, es desde donde se desean copiar los datos.

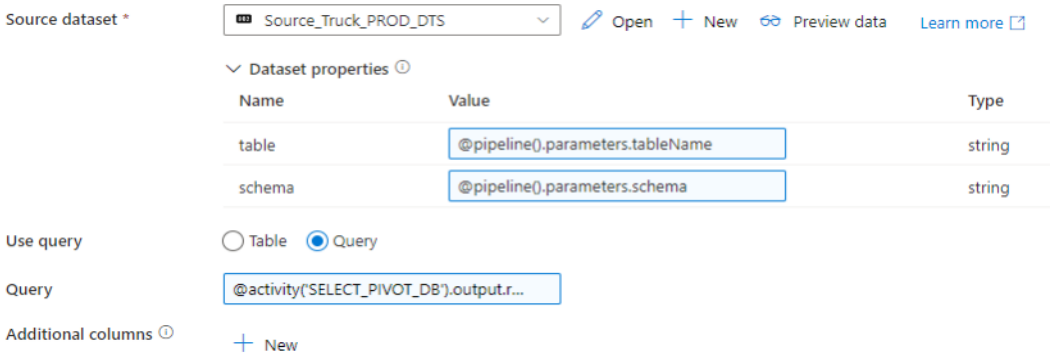


Figura 11 Configuración del source.
Fuente: Elaboración propia (basada en la práctica).

Dentro del source se observa que existe un dataset (colección de datos) que en su interior tiene la conexión al sistema Truck. Para este dataset se necesitan los parámetros “table” y “schema” los cuales fueron configurados con anterioridad dentro del pipeline PTBL01 (Figura 11.

Luego se puede ver que para obtener los datos se tilda la opción “Query”, esto significa que se obtienen los datos de la tabla mediante una consulta SQL, la cual se encuentra configurada dentro de la notebook “SELECT_PIVOT_DB”. Para el caso de esta tabla la sentencia SQL a utilizar es la siguiente:

SELECT * FROM TRK35NULL.PTBL01 FOR FETCH ONLY

Esta consulta trae todos los datos de la tabla PTBL01 del esquema TRK35NULL, que se encuentra en un ambiente productivo.

Configuración del dataset (source)

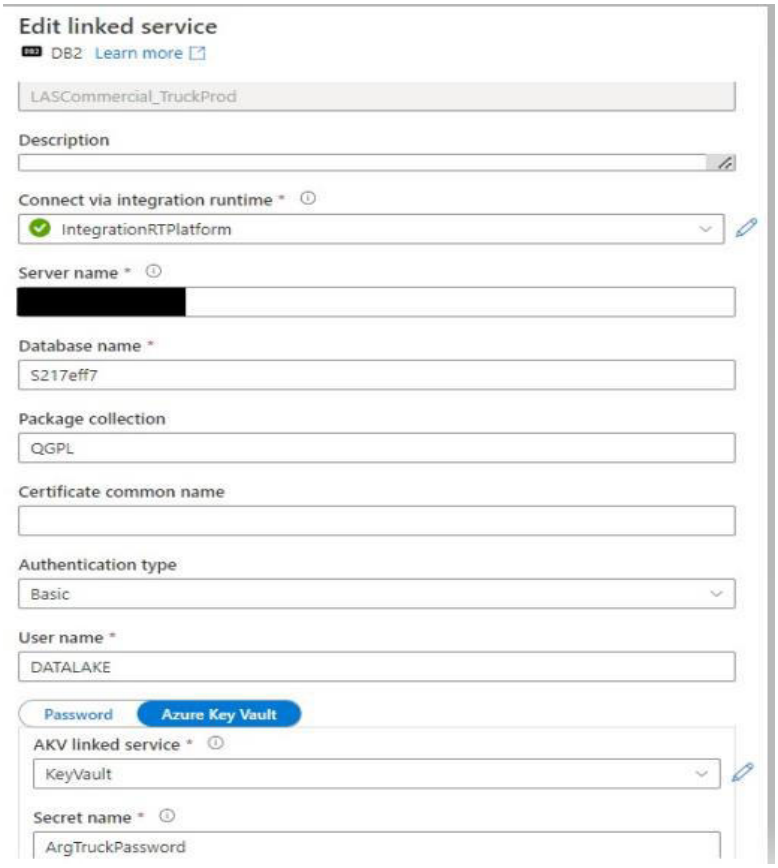


Figura 12 Configuración del dataset source.
Fuente: Elaboración propia (basada en la práctica).

El dataset se encuentra conectado vía integration runtime, se necesita el nombre del servidor, de la base de datos, del package y un tipo de autenticación. En este caso se utiliza autenticación básica y Azure Key Vault para el cifrado de claves.

Con todo esto configurado se pueden leer y obtener los datos del sistema origen y proceder a configurar el Sink.

Sink: es donde se configura la conexión al sistema destino, en nuestro caso el datalake. Básicamente, es el lugar a donde se quieren llevar los datos que se copiaron del origen.

Al igual que en el source, el sink tiene un dataset que provee la conexión al sistema destino (Datalake) y se reciben los mismos dos parámetros “table” y “esquema” los cuales como se mencionó ya fueron configurados.

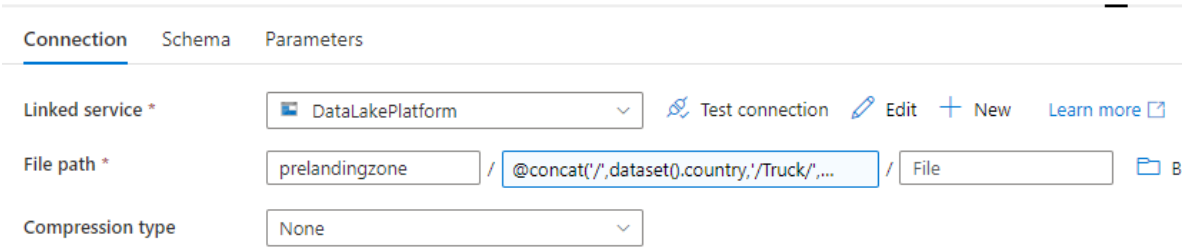


Figura 13 Configuración del sink.
Fuente: Elaboración propia (basada en la práctica).

En la conexión del dataset se observa que ya está configurado el link al "DataLakePlatform" mediante el cual se accede a las diferentes zonas del datalake, y en "File Path" se configura el directorio exacto donde se van a copiar los datos dentro del DL. Todo esto está parametrizado, por lo que el directorio dentro se observa de la siguiente forma:

@concat('/',dataset().country,'/Truck/',dataset().table,'/',string(formatDateTime(utcnow(),'yyyy')),'/',string(formatDateTime(utcnow(),MM')),'/',formatDateTime(utcnow(), 'dd'),'/',dataset().table)

Esto se traduce en prelandingzone/Argentina/Truck/PTBL01/Año/Mes/Dia/PTBL01, donde año, mes y día corresponden a la fecha en que se ejecutó el pipeline de réplica de la tabla.

Si todo se ejecutó correctamente y la consulta al sistema origen obtuvo registros a copiar ya se puede afirmar que los datos se encuentran en la prelanding zone y solo restaría copiarlos a la history zone y consume zone.

Insert Pivot

La actividad siguiente al copy data es la "INSERT_PIVOT_DB", que se encarga de almacenar toda la información del copy data ejecutado anteriormente. Por cada ejecución del copy data se generan registros en los parámetros explicados en el inciso anterior y la insert pivot se encarga de mantener un historial de estas ejecuciones para poder llevar un control de cuál

fue el último valor pivot de la tabla replicada, cuáles fueron los parámetros involucrados y si todo se ejecutó de acuerdo con lo esperado.

If Condition

La última actividad para culminar con la réplica de tablas maestros es un condicional que como se puede apreciar en la Figura 10 se encarga de verificar si los datos leídos en el copy data son iguales a cero. En caso de ser cero el proceso es True y no genera ninguna actividad ya que significa que hubo un problema con la tabla origen o que simplemente dicha tabla no tiene datos a copiar por lo cual el PPL falla. En el otro caso (distinto de cero), el proceso es False lo que significa que se obtuvieron datos desde la tabla origen y fueron copiados exitosamente a la prelanding zone, por lo que el condicional invoca a otro pipeline que contiene dos notebooks de Databricks donde se realiza la copia de datos desde PZ a HZ y finalmente desde HZ a CZ.

2.4.1.2 Réplicas entre las distintas zonas

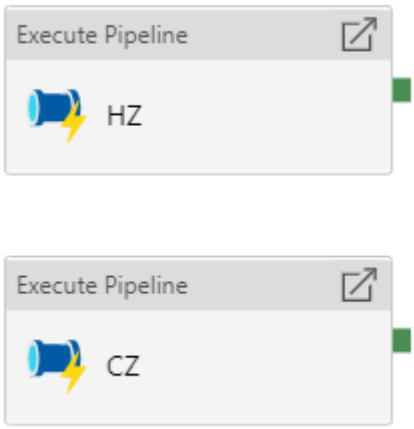


Figura 14 Pipelines de réplicas entre zonas.
Fuente: Elaboración propia (basada en la práctica).

El pipeline HZ invoca una notebook que se encarga de copiar los datos desde la prelanging zone a la history zone. La diferencia entre estas dos zonas

es que en la HZ se almacena toda la historia de los datos, con todas las versiones y se eliminan los registros duplicados. Además, el formato utilizado para esta zona es el parquet debido a su velocidad para consultas.

Figura 15 Ejemplo réplica PZ a HZ.
Fuente: Elaboración propia (basada en la práctica).

Como se aprecia en la figura 15 al pasar de la PZ a HZ mediante transformaciones utilizando SQL y Python en Databricks se eliminan los registros duplicados. En este caso el ID 3 estaba duplicado en la prelanding zone y al pasar a la history zone solo se conserva un registro. Se podría pensar que el ID 1 se encuentra duplicado, pero en realidad no es así, se tienen dos registros con este ID debido a que la persona cambió de dirección. En la history zone se deben conservar ambos registros ya que se necesita la historia de todos los datos en sus diferentes versiones.

El pipeline CZ por su parte copia los registros desde la history zone a la consume zone, pero en este caso solo se conservan los datos listos para ser consumidos, solo se necesita la última versión de los registros.

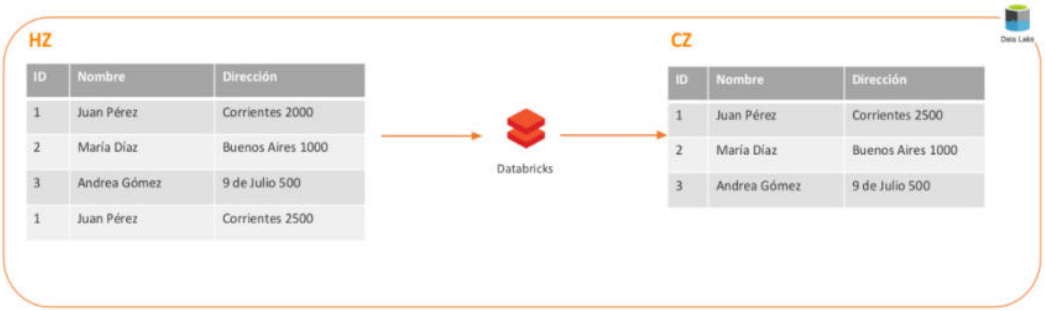


Figura 16 Ejemplo réplica de HZ a CZ.
Fuente: Elaboración propia (basada en la práctica).

Siguiendo el ejemplo de la Figura 16 se puede notar que al pasar de la HZ a la CZ solo se conserva la última versión del ID 1, es decir, se elimina la versión anterior del registro y se mantiene el dato actual que contiene la última dirección de Juan Pérez.

Con todo este proceso ejecutado en forma correcta se tienen los datos listos para ser consumidos en nuestro datalake y se puede comenzar a trabajar en los modelos necesarios para el proyecto que nos compete.

Este proceso de ingesta se aplica también para los sistemas SAP y GVQ. La única diferencia respecto al proceso detallado anteriormente es que para este se necesita crear un nuevo source en la herramienta copy data, para cada sistema origen. En otras palabras, se necesita un source que contenga la conexión a Truck como se explicó y otros dos sources, uno que contenga la conexión al sistema SAP y otro con conexión al sistema GVQ. Una vez configurado el source a cada sistema origen el proceso de ingesta maestro es igual para los tres sistemas (Truck, SAP y GVQ)

2.4.1.3 Ingesta incremental

Como se explicó el proceso de ingesta incremental se realiza cuando el volumen de datos de la tabla origen es considerado grande como para replicarlo en forma completa todos los días. En este caso, se considera que una tabla tiene un volumen de datos elevado cuando supera el millón de registros y es allí cuando se procede a realizar una ingesta de tipo incremental.

Para las ingestas incrementales las tablas origen (en la mayoría de los casos) tienen un campo de control o auditoría, que ayuda en este proceso y este campo indica cuál es la última fecha de actualización de los datos y la fecha en que se ingresaron nuevos registros en el sistema. A partir de estos datos (luego de realizar una primera replica total de la tabla) se puede saber cuáles son los nuevos registros de la tabla y cuáles fueron actualizados en una fecha posterior a la réplica total realizada la primera vez. De esta forma, se pueden traer al

sistema destino (datalake) estos registros, tanto los nuevos como los que fueron actualizados.

Para los sistemas origen SAP y GVQ las tablas contienen un campo de control llamado “ULT_ACTUALIZACION”, el cual indica cuáles son registros nuevos y cuáles fueron actualizados recientemente. En consecuencia, para estos sistemas solo fue necesario construir una consulta SQL que la primera vez leyera todos los registros y a partir del siguiente día solo las actualizaciones. Sin embargo, para el sistema Truck no existe ningún campo de auditoría en las tablas origen, por lo que es necesario aplicar una lógica de negocio incremental diferente en aquellas que superaran el millón de registros.

Proceso de ingesta incremental para el sistema Truck.

Dentro de Truck una de las tablas principales a replicar es la “PLANIL” que contiene información acerca de las planillas de pedidos y ventas que realiza la compañía. En esta tabla existe un campo (PLASTS), que contiene el estado de las planillas; además, entre estos estados existe el “cerrado” y la clave para lograr implementar este proceso fue saber que una vez que una planilla tiene un estado “cerrado” los datos referentes a ella y todos sus relacionados ya no pueden sufrir ningún tipo de cambio o actualización.

Basándose en esta información se solicitó a los referentes del sistema Truck la creación de dos tablas que se utilizaran como si fueran campos de control y auditoría. Dichas tablas fueron llamadas “CTRLPLANIL”, alias control planil, la cual contiene información de todas las planillas que se encuentran en estado cerrado y la “TMPPLANIL”, alias tabla temporal planil, que contiene la información de las planillas que aún no se han cerrado.

```
INSERT INTO TRK35NULL.TMPPLANIL
SELECT P.*
FROM TRK35NULL.PLANIL P
WHERE NOT EXISTS
(SELECT * FROM TRK35NULL.CTRLPLANIL T
  WHERE P.EMPID = T.EMPID AND P.PLANRO = T.CLMNROPLA)
    AND PLAFCH >= Cast(VarChar_Format((CURRENT_DATE -1 YEAR),'yyyyMMdd') as int
)
```

Figura 17 Lógica de carga de la tabla TMPPLANIL.
Fuente: Elaboración propia (basada en la práctica).

Como se observa en la Figura 17 para la carga de la tabla “TMPPLANIL” se utiliza una consulta SQL, que realiza una extracción de datos de la “PLANIL”, pero para todos aquellos números de planilla que no existan en la tabla “CTRLPLANIL”. En efecto, trae todas las planillas con estado distinto de cerrado (estas planillas son las que se encuentran dentro de la “CTRLPLANIL”) para el último año.

Para terminar de comprender este proceso se debe entender que, en primer lugar, la tabla “CTRLPLANIL” está “vacía” por lo que todos los datos de la “PLANIL” pasarán a la “TMPPLANIL” y se replicarán a nuestro datalake.

```
INSERT INTO TRK35NULL.CTRLPLANIL
(EMPID, CLMNROPLA, PLAFCHCIE, FCHPRO)
SELECT DISTINCT EMPID
              , PLANRO
              , PLAFCHCIE
              , Cast(VarChar_Format((CURRENT_DATE),'yyyyMMdd') as int) AS FCHPRO
FROM  TRK35NULL.TMPPLANIL P
WHERE PLASTS = 'CER'
AND  NOT EXISTS  (SELECT * FROM TRK35NULL.CTRLPLANIL T
                  WHERE P.EMPID = T.EMPID
                    AND P.PLANRO = T.CLMNROPLA)')
```

Figura 18 Lógica de carga de la tabla CTRLPLANIL.
Fuente: Elaboración propia (basada en la práctica).

Una vez que todos los datos de la “PLANIL” están dentro de la “TMPPLANIL” se procede a realizar la consulta SQL que carga la tabla “CTRLPLANIL” (Figura 18). Esta consulta inserta dentro de la control planil todos los datos de las planillas que están dentro de la tabla temporal planil cuyo estado es cerrado.

Por lo tanto, aplicando esta lógica se tienen registros dentro de la tabla “CTRLPLANIL” y los mismos ya no volverán a ser consultados en la próxima carga de la “TMPPLANIL”. En consecuencia, se reducen la cantidad de registros que se replican diariamente al datalake debido a que los datos que están dentro de la tabla “CTRLPLANIL” no vuelven a ser leídos para realizar una réplica y solo se conservan aquellos registros que contienen datos de planillas pendientes de cerrar.

Como se puede observar en la figura 17 esta lógica de traer datos solo de planillas pendientes de cierre se aplica para planillas del último año. Entonces, si se tienen planillas que se cerraron en un periodo anterior en nuestro datalake seguirán apareciendo en estado pendiente y para solucionar esto se decidió que los días domingo se realizará una consulta SQL igual a la de la figura 17, pero sin la condición temporal. Es decir, se traerá toda la historia de las planillas y, de esta forma, si existen planillas con fecha anterior a un año durante la ejecución de los domingos y se encuentran en estado cerrado se actualizará la información en el DL.

En conclusión, la única diferencia entre la ingesta incremental y la maestro son estas consideraciones que se mencionaron. Si recurrimos a la lógica más elemental se puede decir que en las tablas maestro se utiliza una consulta SQL, que trae toda la información de la tabla a leer y la réplica al DL todos los días, lo que se efectúa es un “SELECT * FROM TABLA” y se replica la información que contiene dicha tabla. En cambio, en la ingesta incremental esta consulta SQL cambia y como se mencionó verifica el estado de las planillas y solo se conservan aquellas que aún están pendientes de cierre por lo que aún pueden sufrir modificaciones.

Para que se pueda realizar una ingesta incremental de una tabla de Truck esta debe tener algún campo que permita relacionarla con la tabla “PLANIL” para, de esta forma, acceder al estado de las planillas y poder diferenciar entre las cerradas y las no cerradas.

2.4.2 Consideraciones finales de la OLA 1

Una vez detallados los procesos de réplica de las tablas de los distintos sistemas origen es necesario aclarar cuántas tablas nuevas se replicaron para cada país y cada sistema en relación a nuestro datalake.

- Para Argentina se replicaron 28 tablas del sistema Truck, 2 tablas de SAP y 1 tabla de GVQ.
- Para Bolivia se replicaron 19 tablas del sistema Truck. 2 tablas de SAP y 2 tablas de GVQ.
- Para Paraguay se replicaron 24 tablas del sistema Truck, 2 tablas de SAP y 2 tablas de GVQ.
- Para Uruguay se replicaron 30 tablas del sistema Truck, 2 tablas de SAP y 2 tablas de GVQ.

2.5 Ola 2: Análisis y documentación del modelo dimensional

Como se ha mencionado en oportunidades anteriores el modelo dimensional, actualmente, es un sistema OLAP que se encuentra montado dentro de un DW llamada BIPROD, que utiliza una base de datos Oracle y se nutre de los datos provenientes de Truck, GVQ y SAP.

Dentro de la base de datos Oracle se encuentra el Data Warehouse y se puede observar que este contiene muchos esquemas en su interior. De estos esquemas para este proyecto son relevantes solamente dos, los cuales se detallan:

- DWSADATA: es un paso anterior a la creación del DW final y contiene algunos procesos ETL realizados para lograr refinar el modelo dimensional que interesa replicar.

- DWDATA: es el DW que se ve replicar o para ser más exactos que contiene los modelos OLAP de gestión comercial, que son los que se necesitan replicar para este desarrollo.

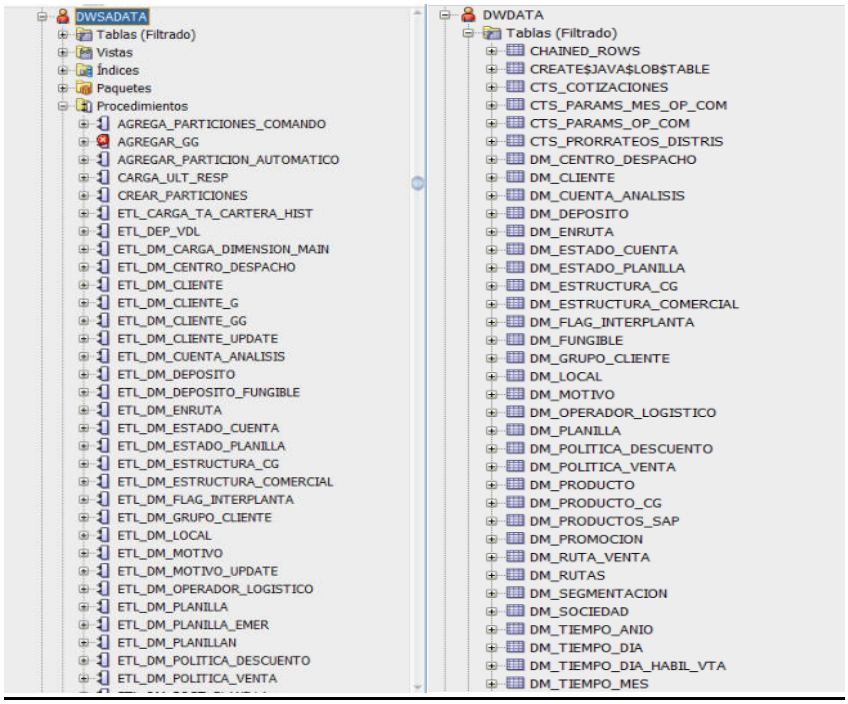


Figura 19 Esquemas DWSADATA y DWDATA.
Fuente: Elaboración propia (basada en la práctica).

En la figura 19 se puede observar con más detalle que dentro de DWSADATA se encuentran los procedimientos ETL que fueron realizados para construir las diferentes dimensiones del modelo dimensional. Por su parte, se aprecia que en DWDATA ya se encuentran las tablas dimensionales y tablas de hechos construidas y listas para utilizar por las herramientas de reporting.

Considerando estos dos esquemas y viendo cuáles son las tablas dimensionales que se crean y utilizan en ellos es necesario analizar qué tablas ya fueron desarrolladas por otros equipos y se encuentra en nuestro datalake y cuáles se deben crear desde cero. También se debe verificar que las tablas que ya fueron desarrolladas contengan todos los campos que utiliza el modelo OLAP y en caso de que falten campos se deberá modificar la dimensión para incorporarlos.

2.5.1 Tablas dimensionales a desarrollar por país

Argentina

Dimensiones que fueron desarrolladas por otros equipos, pero se necesitan incorporar nuevos campos para el proyecto:

- Dimensión Productos: se deben incorporar los campos MARCA_UNID_DSC y FAMILIA_DSC.
- Dimensión Clientes Histórica: se deben incorporar los campos NOMBRE_FANTASIA y FLAG_PERF.
- Dimensiones Estructura Comercial Histórica y Zona Histórica: se deben incorporar los campos COORDINADOR_ID y COORDINADOR_DSC.
- Dimensión Planillas: en BIPROD es una dimensión, pero otro equipo de la compañía cliente la migró y la convirtió en una tabla de hechos. Sin embargo, en este proyecto continúa funcionando como una tabla dimensional porque solo se utilizan campos que no son indicadores. Por el contrario, estos campos dan un contexto para los KPIS (Key Performance Indicator) de las tablas de hechos desarrolladas en el proyecto. En esta tabla se deben incorporar dos nuevos campos, los cuales se detallan a continuación: ESTADO_VENTA_DSC y ESTADO_PLANILLA_DSC

Dimensiones que fueron desarrolladas por otros equipos y no necesitan ningún tipo de modificación para la migración del proyecto de esta PPS:

- Dimensión Segmentos.
- Dimensión Sociedades.
- Dimensión Canales.
- Dimensión Tiempo Día.

- Dimensión Depósitos.
- Dimensión Tipo de Movimiento.
- Dimensión Territorio Histórico.
- Dimensión Región Histórica.
- Dimensión Área Histórica.
- Dimensión Dirección Histórica.

Bolivia

Dimensiones que fueron desarrolladas por otros equipos, pero se necesitan incorporar nuevos campos para el proyecto:

- Dimensión Productos: se deben incorporar los campos MARCA_UNID_DSC y FAMILIA_DSC.
- Dimensión Clientes Histórica: se deben incorporar los campos NOMBRE_FANTASIA y FLAG_PERF.

Dimensiones que fueron desarrolladas por otros equipos y no necesitan ningún tipo de modificación para la migración del proyecto de esta PPS:

- Dimensión Segmentos.
- Dimensión Sociedades.
- Dimensión Canales.
- Dimensión Tiempo Día.
- Dimensión Depósitos.
- Dimensión Tipo de Movimiento.

Dimensiones que se desarrollaron desde cero según las necesidades de este proyecto, debido a que no existían en Bolivia y ningún otro equipo iba a generarlas:

- Dimensión Estructura Comercial Histórica.
- Dimensión Territorio Histórico.
- Dimensión Zona Histórica.
- Dimensión Región Histórica.
- Dimensión Subregión Histórica.
- Dimensión Área Histórica.
- Dimension Dirección Histórica.
- Dimensión Planillas: en realidad se la desarrollará como si fuera una tabla de hechos para mantener concordancia con lo desarrollado por otro equipo en Argentina. Sin embargo, como ya se explicó, para este proyecto funciona como una dimensión.

Cabe destacar que para mantener concordancia en los desarrollos se respetará la estructura, nomenclatura y campos de las tabla dimensionales que ya fueron desarrolladas para Argentina y, también, se necesiten crear desde cero en los otros países.

Paraguay

Dimensiones que fueron desarrolladas por otros equipos, pero se necesitan incorporar nuevos campos para el proyecto:

- Dimensión Productos: se deben incorporar los campos MARCA_UNID_DSC y FAMILIA_DSC.
- Dimensión Clientes Histórica: se deben incorporar los campos NOMBRE_FANTASIA y FLAG_PERF.
- Dimensiones Estructura Comercial Histórica y Zona Histórica: se deben incorporar los campos COORDINADOR_ID y COORDINADOR_DSC.

Dimensiones que fueron desarrolladas por otros equipos y no necesitan ningún tipo de modificación para la migración del proyecto de esta PPS:

- Dimensión Segmentos.
- Dimensión Sociedades.
- Dimensión Canales.
- Dimensión Tiempo Día.
- Dimensión Depósitos.
- Dimensión Tipo de Movimiento.

Dimensiones que se desarrollaron desde cero según las necesidades de este proyecto debido a que no existían en Paraguay y ningún otro equipo iba a generarlas:

- Dimensión Territorio Histórico.
- Dimensión Región Histórica.
- Dimensión Subregión Histórica.
- Dimensión Área Histórica.
- Dimensión Dirección Histórica.
- Dimensión Planillas: como ya se explicó en realidad es una tabla de hechos.

Uruguay

Dimensiones que fueron desarrolladas por otros equipos, pero se necesitan incorporar nuevos campos para el proyecto:

- Dimensión Productos: se deben incorporar los campos MARCA_UNID_DSC y FAMILIA_DSC.

- Dimensión Clientes Histórica: se deben incorporar los campos NOMBRE_FANTASIA y FLAG_PERF.

Dimensiones que fueron desarrolladas por otros equipos y no necesitan ningún tipo de modificación para la migración del proyecto de esta PPS:

- Dimensión Segmentos.
- Dimensión Sociedades.
- Dimensión Canales.
- Dimensión Tiempo Día.

Dimensiones que se desarrollaron desde cero según las necesidades de este proyecto debido a que no existían en Uruguay y ningún otro equipo las generaría:

- Dimensión Territorio Histórico.
- Dimensión Región Histórica.
- Dimensión Subregión Histórica.
- Dimensión Área Histórica.
- Dimensión Dirección Histórica.
- Dimensión Estructura Comercial Histórica.
- Dimension Depósitos.
- Dimensión Tipo de Movimiento.
- Dimensión Planillas: como ya se explicó en realidad es una tabla de hechos.

2.5.2 Tablas de hechos a desarrollar en el datalake

Luego de las tablas dimensionales fue necesario realizar un análisis de las tablas de hechos, que se utilizan para realizar los reportes de gestión comercial. Es necesario mencionar que estas tablas no fueron desarrolladas por ningún otro equipo ya que son el objetivo principal de este proyecto. Debido a ello, no existen en ninguno de los cuatro países y la lógica que se utilizara para desarrollarlas es la misma en todos, salvo algunas pequeñas excepciones que se mencionarán en la OLA 3 “Reingeniería del modelo de gestión comercial”.

Del análisis realizado se desprende que las tablas de hechos se encuentran dentro del esquema DWDATA y son las dos que se mencionan a continuación:

- TS_VENTA_DESCUENTO: contiene información sobre las ventas que realiza la cervecería, con un detalle y apertura sobre la venta bruta, venta neta, venta con descuentos, ventas sin cargo, etc.

- TS_OBJETIVO_VENTA: provee información acerca de cuál es el objetivo de ventas de la empresa a nivel mensual con una apertura sobre ventas por región, cliente, producto, etc.

Figura 20 Tablas de hechos del modelo OLAP.
Fuente: Elaboración propia (basada en la práctica).

Como se observa en la Figura 20 las dos tablas de hechos tienen muchos campos, pero no todos se utilizan en los reportes de gestión comercial. A raíz de ello fue necesario realizar un análisis de qué campos se utilizan en los reportes para incorporarlos a las nuevas tablas de hechos, que se generarán en el datalake. Algunos campos surgen de las dimensiones que se analizaron en el punto anterior y otros corresponden a los indicadores que se obtienen de Truck a través de un desarrollo ETL generado en GeneXus.

A partir de este análisis se llegó a la conclusión de que para la tabla “ventas_diarias” (réplica de la tabla “ts_venta_descuento”) se van a necesitar los siguientes campos que se mencionan a continuación:

- fk_tiempo_emision_venta
- fk_tiempo_pedido_venta
- fk_tiempo_salida_venta
- fk_tiempo_cierre_venta

- fk_deposito
- fk_sociedad
- fk_cliente_hist
- fk_canal
- fk_producto
- fk_segmento
- fk_ec_territorio
- fk_movimiento
- fk_planilla
- hl_venta_bruta
- hl_venta_neta
- hl_total_scargo
- importe_facturado
- importe_neto
- importe_dtos_total
- importe_sc_total
- und_venta
- precio_unitario
- q_und_por_empaque
- und_scargo_aut
- und_scargo_man
- und_scargo_neg
- importe_dto
- importe_mfinanciero
- importe_mfin_canje
- last_update
- periodo

En el caso de la réplica de la tabla “ts_objetivo_venta”, que en el Data Lake se llamará “objetivos_ventas”, se necesitarán los campos que se detallan a continuación:

- fk_tiempo_dia
- fk_sociedad
- fk_cliente_hist
- fk_canal
- fk_segmento
- fk_producto
- fk_ec_territorio
- hl_objetivo
- last_update
- periodo

2.6 Ola 3: Reingeniería del modelo de gestión comercial

A partir del análisis e ingesta de tablas descripto en la OLA 1 y el análisis de los modelos OLAP del DW BIPROD mencionados en la OLA 2 ya se está en condiciones de comenzar a replicar y ajustar el modelo dimensional en el Data Lake, para ello se tienen replicar los procesos ETL realizados en GeneXus y también dentro de la base de datos Oracle.

En primer lugar, se desarrollarán las tablas dimensionales necesarias para el proyecto, debido a que son necesarias para luego crear las tablas de hechos de ventas_diarias y objetivos_ventas.

2.6.1 Proceso de desarrollo de tablas dimensionales

Para este proyecto se desarrollarán dos tipos de tablas de dimensiones: dimensiones de tipo 1 y dimensiones de tipo 2, las diferencias entre ambas ya fueron explicadas en la introducción a conceptos. Sin embargo, es necesario

aclarar que para este proyecto todas aquellas dimensiones que son históricas serán tablas dimensionales de tipo 2 y el resto de tipo 1.

Teniendo esto presente, en primer lugar, se describirá el proceso de desarrollo desde el inicio de una tabla dimensional de tipo 1 y, en segundo lugar, el proceso de desarrollo para una tabla dimensional de tipo 2.

2.6.1.1 Desarrollo de tabla dimensional de tipo 1

En este caso se describirá el proceso de desarrollo de la tabla dimensional Depósitos para Uruguay la cual fue desarrollada desde cero siguiendo los lineamientos de estructura y nomenclatura utilizados en la tabla existente en Argentina.

Lo primero que se debe hacer es analizar cuáles son las tablas orígenes que intervendrán en el desarrollo de la dimensión, es decir, se necesita saber de qué tablas orígenes se extraen los datos que conforman los campos de cada una de las columnas de la tabla dimensional.

Para la dimensión Depósitos solo se necesita una tabla origen proveniente de Truck, la tabla “PTBL73”.

Todo el proceso ETL para crear la dimensión se realiza utilizando una notebook en Azure Databricks, por lo cual es necesario explicar cómo están divididos los procesos dentro del espacio de trabajo de la herramienta.

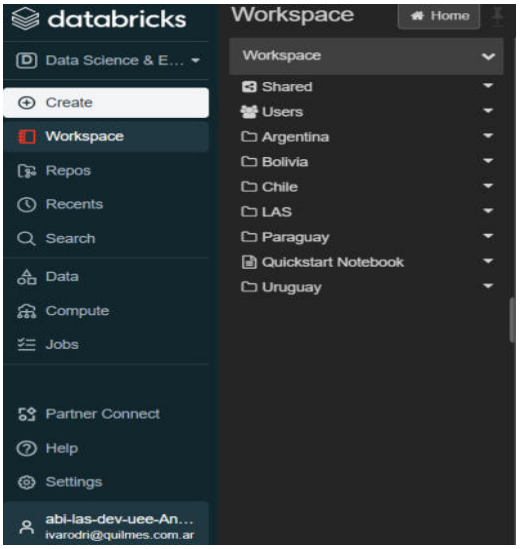


Figura 21 División del espacio de trabajo en Databricks.
Fuente: Elaboración propia (basada en la práctica).

Como se observa en la Figura 21 los procesos ETL que se realizan en Databricks están divididos en carpetas según cada País y aquellos que son generales, es decir, se realizan en todos los países se encuentran en la carpeta "LAS". Por otro lado, en la esquina inferior izquierda se puede observar qué usuario es el que inició sesión en la herramienta y según esto un usuario administrador otorga permisos a las distintas notebooks de los espacios de trabajo. De esta manera, para poder hacer cambios en cualquier notebooks de Databricks primero se deben solicitar los permisos correspondientes y cada vez que un usuario realiza una ejecución de una notebook esta queda registrada, esto facilita la transparencia, seguimiento y confiabilidad de los procesos ETL desarrollados en la plataforma.

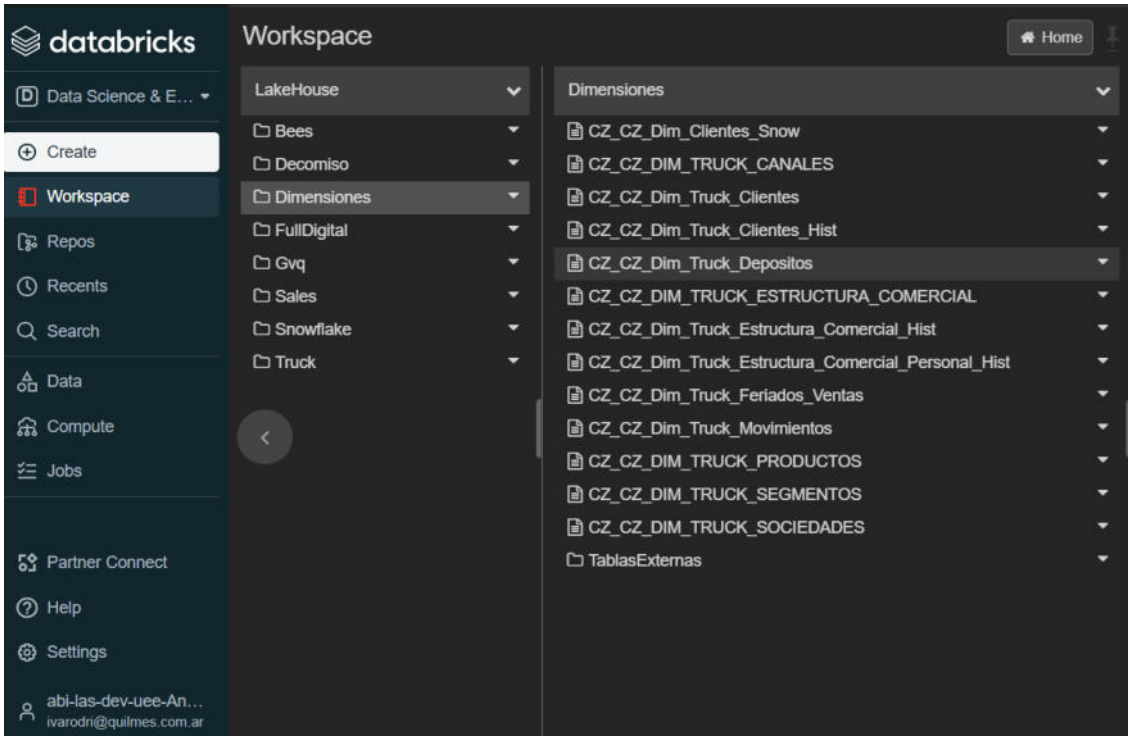


Figura 22 Subdivisión del espacio de trabajo en Databricks.
Fuente: Elaboración propia (basada en la práctica).

Una vez que se hace un clic dentro de una de las carpetas correspondientes a un país se puede observar una subdivisión de carpetas. Esa subdivisión corresponde al nombre de cada equipo, base de datos o modelos en los que se está trabajando. Para este ejemplo se centrará la explicación en la carpeta “Dimensiones” y en su interior en la notebook llamada “CZ_CZ_Dim_Truck_Depositos”.

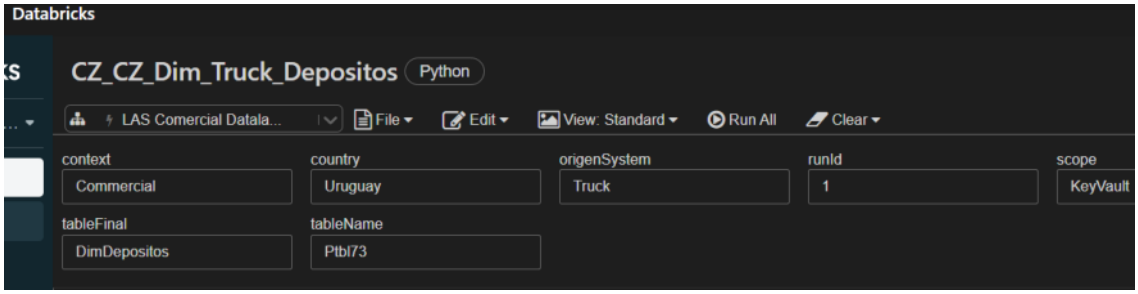


Figura 23 Notebook inicio de dimensión depósitos en Databricks.
Fuente: Elaboración propia (basada en la práctica).

Como se observa en la Figura 23 en la esquina superior izquierda se detalla el nombre que se le da a la notebook y a su costado derecho el lenguaje por defecto que se utiliza, que en este caso es Python. Por convención del cliente todas las notebooks deben estar definidas con Python como su lenguaje de desarrollo, pero dentro de las celdas se puede utilizar lenguaje SQL simplemente escribiendo el código “%sql” al inicio.

Debajo del nombre de la notebook se puede observar el clúster que se está utilizando y a su derecha un menú de opciones desde el cual se pueden realizar múltiples tareas sobre la notebook como, por ejemplo: copiarla, editarla, ajustar las vistas, ejecutar todas las celdas o limpiar todo el contenido, etc.

Por último, en la parte inferior de la Figura 23 se observan múltiples parámetros, a este sector de la notebook se lo conoce como widgets y se utiliza para cargar los parámetros provenientes de Azure Data Factory en la notebook. Es por ello, que se pueden observar algunos de los parámetros que fueron descriptos en la OLA 1 de ingesta de fuentes de datos.

```
%sql

DROP VIEW IF EXISTS VIEW_TMP_DEP_UY;

CREATE OR REPLACE TEMP VIEW VIEW_TMP_DEP_UY as (
select
 cast(deposito.EMPID as int) AS EMPRESA_ID,
 cast(deposito.DPSID as int) AS DEPOSITO_ID,
upper(trim(deposito.DPSTXT)) AS DEPOSITO_DESC,
upper(trim(deposito.DPSABV)) AS  DEPOSITO_ABREV,
cast(deposito.LAST_UPDATE as date) AS LAST_UPDATE
FROM TRUCK.UY_PTBL73 AS deposito

);

OK
```

Figura 24 Vista temporal para comenzar a crear la dimensión depósitos en Databricks. Fuente: Elaboración propia (basada en la práctica).

En la Figura 24 se observa la consulta SQL que crea una vista temporal con los datos necesarios para la dimensión depósitos. Estos datos se extraen de la tabla PTBL73 proveniente de Truck.

Se puede observar que en, primer lugar, se selecciona el campo “EMPID” y se lo transforma al tipo de dato entero y finalmente se le asigna el nombre de “EMPRESA_ID” y se hace lo mismo el campo DPSID (DEPOSITO_ID) proveniente de la tabla PTBL73. Para el caso de los campos “DPSTXT” (DEPOSITO_DESC) y “DEPSABV” (DEPOSITO_ABREV) se utilizan las funciones upper y trim (funciones de SQL para tipos de datos String, es decir, textos). La primera tiene como función poner el texto del campo en mayúsculas y, la segunda, eliminar posibles espacios en blanco al principio o final del texto. Por último, al campo LAST_UPDATE se lo transforma a tipo fecha manteniendo el nombre original.

```
%sql

DROP TABLE IF EXISTS VIEW_FULLJOIN_DEP_UY;
CREATE OR REPLACE TEMP VIEW VIEW_FULLJOIN_DEP_UY as (
SELECT
  coalesce(stg.EMPRESA_ID, TF.EMPRESA_ID) as EMPRESA_ID,
  coalesce(stg.DEPOSITO_ID, TF.DEPOSITO_ID) as DEPOSITO_ID,
  coalesce(stg.DEPOSITO_DESC, TF.DEPOSITO_DESC) as DEPOSITO_DESC,
  coalesce(stg.DEPOSITO_ABREV, TF.DEPOSITO_ABREV) as DEPOSITO_ABREV,
  coalesce(stg.LAST_UPDATE, TF.LAST_UPDATE) as LAST_UPDATE
FROM VIEW_TMP_DEP_UY stg --tabla origen
FULL JOIN LK.DIM_DEPOSITOS_UY TF --tabla dimensional
  on stg.EMPRESA_ID = TF.EMPRESA_ID
  and stg.DEPOSITO_ID = TF.DEPOSITO_ID
) ;

OK
```

Figura 25 Vista temporal con full join entre tabla origen y dimensional depósitos
Fuente: Elaboración propia (basada en la práctica).

En la Figura 25 se crea una vista temporal que hace un full join entre la tabla origen y la tabla dimensional (Figura 28) con el fin de tener todos los registros de ambas tablas, para luego elegir cual dato debe conservarse.

Para lograr esto se deben tener en cuenta tres casos que se describen a continuación:

- Caso 1: registro existe en tabla origen, pero no en tabla dimensional, se conserva el registro de la tabla origen.
- Caso 2: registro existe en tabla dimensional pero no en tabla origen, se conserva el registro de la tabla dimensional.
- Caso 3: registro existe en ambas tablas (origen y dimensional), se prioriza y conserva el registro de la tabla origen ya que es el dato más actualizado.

El resultado del full join se guarda en la vista temporal "VIEW_FULLJOIN_DEP_UY" la cual será utilizada en la siguiente celda.

```
%sql
DROP TABLE IF EXISTS VIEW_TMP_FINAL_DEP_UY;
CREATE OR REPLACE TEMP VIEW VIEW_TMP_FINAL_DEP_UY as (
SELECT
   ---PK autogenerada
  CAST(
    coalesce(TF.PK_DEPOSITO, SUM(NVL2(TF.PK_DEPOSITO, 0, 1))
    OVER(ORDER BY stg.EMPRESA_ID, stg.DEPOSITO_ID ROWS UNBOUNDED PRECEDING) + CONTEO.MaxId)
    as int) as PK_DEPOSITO,
  stg.EMPRESA_ID     AS EMPRESA_ID,
  stg.DEPOSITO_ID    AS DEPOSITO_ID,
  stg.DEPOSITO_DESC  AS DEPOSITO_DESC,
  stg.DEPOSITO_ABREV AS DEPOSITO_ABREV,
  stg.LAST_UPDATE    AS LAST_UPDATE
FROM VIEW_FULLJOIN_DEP_UY stg --vista temporal
--calcula el maximo
CROSS JOIN ( SELECT coalesce(max(PK_DEPOSITO),0)  as MaxId FROM LK.DIM_DEPOSITOS_UY) CONTEO
LEFT JOIN LK.DIM_DEPOSITOS_UY TF  --tabla dimensional
  on stg.EMPRESA_ID = TF.EMPRESA_ID
  and stg.DEPOSITO_ID = TF.DEPOSITO_ID
WHERE stg.DEPOSITO_ID <> -1 --se excluye el registro con -1 ya que se agrega en el union
UNION
--dato por defecto en caso de que la tabla dimensional se encuentre vacia
SELECT
  -1 AS PK_DEPOSITO,
  -1 AS EMPRESA_ID,
  -1 AS DEPOSITO_ID,
  'DESCONOCIDO' AS DEPOSITO_DESC,
  'DESCONOCIDO' AS DEPOSITO_ABREV,
  cast('2199-12-31' as date) as LAST_UPDATE
);

OK
```

Figura 26 Vista final dimensión depósitos.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 26 se cruza la vista temporal creada en la celda anterior (Figura 25) con la tabla dimensional (Figura 28) con el fin de crear la PK (Primary Key) de la tabla dimensional. Esta PK es una clave subrogada, es decir, es una clave propia dentro del Data Lake que se puede utilizar únicamente en este.

Para crear las PKS de la tabla se deben validar dos casos, los cuales se detallan a continuación:

- Caso 1: si el registro ya existe en la tabla dimensional, se mantiene la PK que ya tenía.
- Caso 2: si el registro no existe en la tabla dimensional, se calcula la máxima Primary Key existente y se le asigna el número consecutivo.

Si la tabla dimensional se encuentra vacía se asignan valores de control por defecto a los campos. En el caso de los datos numéricos se les asigna el valor “-1” y en el caso de los textos el valor “DESCONOCIDO”. Por último, para el valor de control de última actualización se les pone una fecha muy futura, en este caso “2199-12-31” en formato yiya-MM-dd, es decir, año-mes-día.

El resultado de esta consulta se guarda en una vista temporal llamada “VIEW_TMP_FINAL_DEP_UY”.

Hay que tener en cuenta dos condiciones a la hora de realizar esta vista temporal, las cuales se detallan:

- Los campos por los que se hace join deben ser del mismo tipo de dato. Ejemplo: tbl1 join tbl2 on campo1 = campo2. Entonces campo1 y campo2 deben ser ambos numéricos o string.
- Colocar los campos que hacen parte de la clave primaria en el cross join y en el cálculo de la PK autogenerada.

```
try:
    print("Guardar en DL")

    finalTableQuery = spark.sql('''SELECT
                                    PK_DEPOSITO,
                                    EMPRESA_ID,
                                    DEPOSITO_ID,
                                    DEPOSITO_DESC,
                                    DEPOSITO_ABREV,
                                    LAST_UPDATE
                                   FROM VIEW_TMP_FINAL_DEP_UY'''
                                )
    consumeZonePath = '/mnt/consumezone/'+country+'/'+Context+'/Dimensiones/'+tableFinal
    finalTableQuery.write.mode('overwrite').format('parquet').save(consumeZonePath)

    print('Proceso termina exitosamente!')

except Exception as e:
  print('Proceso termina con Error!')
  raise e
```

Figura 27 Guardado de la dimensión depósitos en el DL.
Fuente: Elaboración propia (basada en la práctica).

Finalmente, en la Figura 27 se realiza el guardado de la dimensión depósitos en el Data Lake dentro de la consume zone.

Se puede observar que se utiliza lenguaje Python y dentro de este se realiza una llamada a Spark para poder utilizar la sentencia SQL que selecciona los datos se guardarán en la consume zone, específicamente en la ruta escrita en la variable “consumeZonePath”. Esta variable se compone de textos y se llama a los parámetros de los widgets de la Figura 23 para completarla. Entonces la ruta escrita en la Figura 27 ‘mnt/consumezone/’+country’+’/’+ Context+’/Dimensiones/+tableFinal’ se traduce en ‘mnt/consumezone/Uruguay/ Commercial/Dimensiones/ DimDepositos’.

Dentro de esta ruta es donde se almacenará la tabla dimensional de depósitos de Uruguay en el Data Lake y para acceder a estas mismas tablas, pero de otros países solo es necesario escribir la ruta modificando el nombre del país.

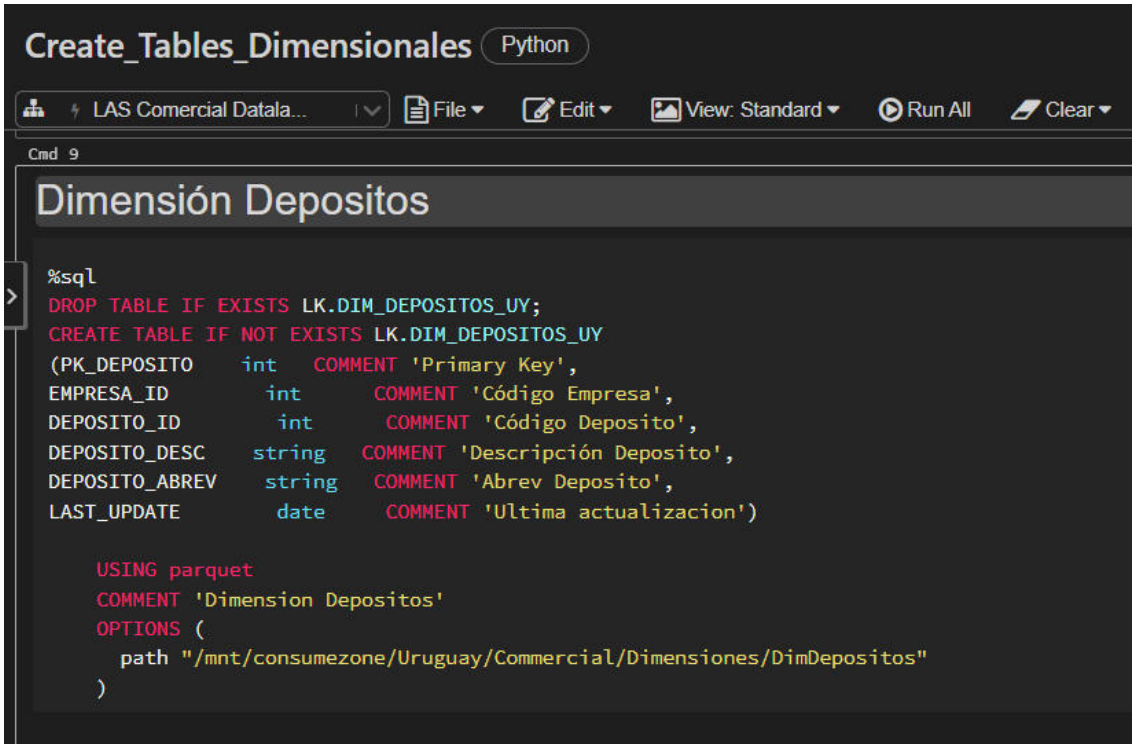


Figura 28 Tabla externa de la dimensión depósitos.
Fuente: Elaboración propia (basada en la práctica).

Esta tabla externa se crea con el fin de poder acceder a los datos de la dimensión desde cualquier espacio de trabajo de Databricks y, también, para poder mantener las mismas PKS en los datos tal como se explicó con anterioridad.

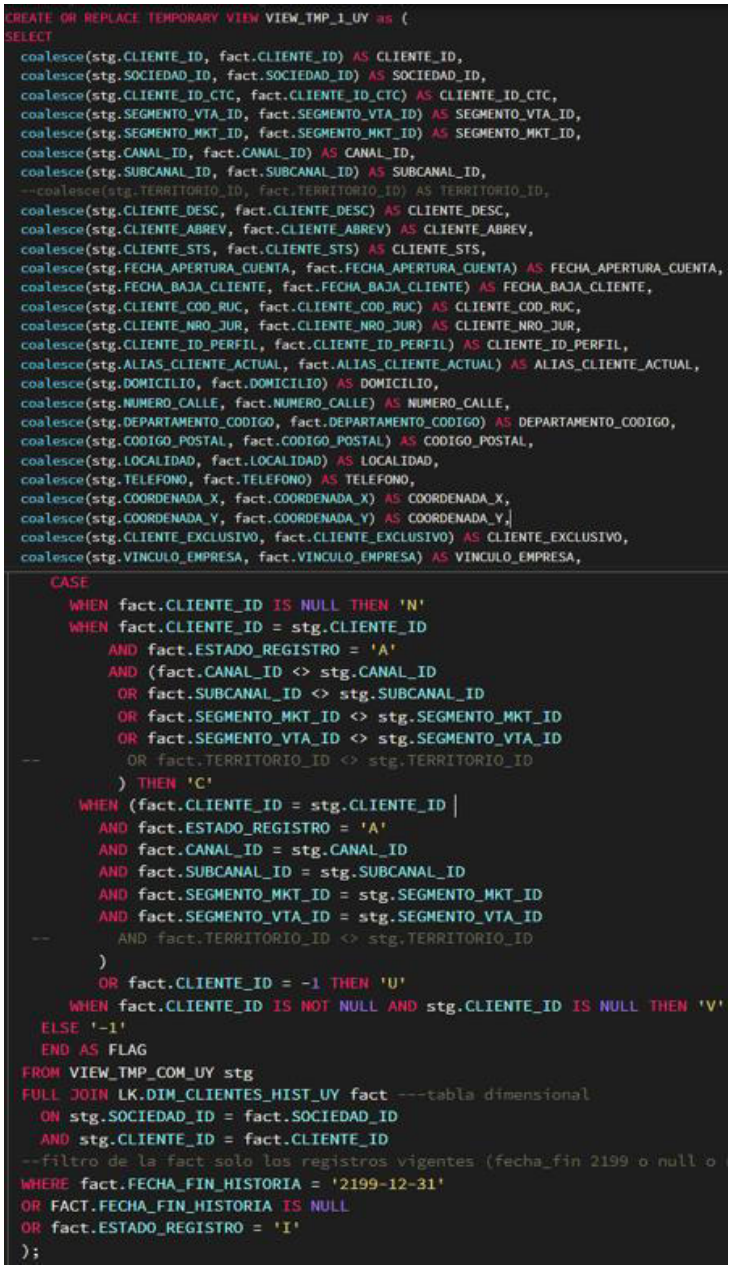
La tabla externa de dimensión depósitos (Figura 28) por obvias razones contiene los mismos campos que se mencionaron y utiliza la misma ruta donde se guardaron los datos en el Data Lake, pero también agrega una descripción a cada uno de los campos para conocer con más detalle a qué hace referencia cada uno de ellos.

Básicamente la tabla externa sirve para acceder a los datos de la dimensión depósitos desde cualquier notebook de Databricks, sin la necesidad de crear una vista o una nueva tabla en esa notebook.

2.6.1.2 Desarrollo de tabla dimensional de tipo 2

El desarrollo de esta tabla dimensional es similar al de la dimensión de tipo 1 que fue explicado anteriormente. La principal diferencia es que se sabe cuándo cambia un registro dimensional en la base de datos origen y de dicho dato se quiere conservar la historia, entonces es necesario crear una nueva PK para este.

Para exponer el desarrollo realizado se utilizará la creación de la Dimensión clientes histórica de Uruguay.



```
CREATE OR REPLACE TEMPORARY VIEW VIEW_TMP_1_UY as (
SELECT
  coalesce(stg.CLIENTE_ID, fact.CLIENTE_ID) AS CLIENTE_ID,
  coalesce(stg.SOCIEDAD_ID, fact.SOCIEDAD_ID) AS SOCIEDAD_ID,
  coalesce(stg.CLIENTE_ID_CTC, fact.CLIENTE_ID_CTC) AS CLIENTE_ID_CTC,
  coalesce(stg.SEGMENTO_VTA_ID, fact.SEGMENTO_VTA_ID) AS SEGMENTO_VTA_ID,
  coalesce(stg.SEGMENTO_MKT_ID, fact.SEGMENTO_MKT_ID) AS SEGMENTO_MKT_ID,
  coalesce(stg.CANAL_ID, fact.CANAL_ID) AS CANAL_ID,
  coalesce(stg.SUBCANAL_ID, fact.SUBCANAL_ID) AS SUBCANAL_ID,
  --coalesce(stg.TERRITORIO_ID, fact.TERRITORIO_ID) AS TERRITORIO_ID,
  coalesce(stg.CLIENTE_DESC, fact.CLIENTE_DESC) AS CLIENTE_DESC,
  coalesce(stg.CLIENTE_ABREV, fact.CLIENTE_ABREV) AS CLIENTE_ABREV,
  coalesce(stg.CLIENTE_STS, fact.CLIENTE_STS) AS CLIENTE_STS,
  coalesce(stg.FECHA_APERTURA_CUENTA, fact.FECHA_APERTURA_CUENTA) AS FECHA_APERTURA_CUENTA,
  coalesce(stg.FECHA_BAJA_CLIENTE, fact.FECHA_BAJA_CLIENTE) AS FECHA_BAJA_CLIENTE,
  coalesce(stg.CLIENTE_COD_RUC, fact.CLIENTE_COD_RUC) AS CLIENTE_COD_RUC,
  coalesce(stg.CLIENTE_NRO_JUR, fact.CLIENTE_NRO_JUR) AS CLIENTE_NRO_JUR,
  coalesce(stg.CLIENTE_ID_PERFIL, fact.CLIENTE_ID_PERFIL) AS CLIENTE_ID_PERFIL,
  coalesce(stg.ALIAS_CLIENTE_ACTUAL, fact.ALIAS_CLIENTE_ACTUAL) AS ALIAS_CLIENTE_ACTUAL,
  coalesce(stg.DOMICILIO, fact.DOMICILIO) AS DOMICILIO,
  coalesce(stg.NUMERO_CALLE, fact.NUMERO_CALLE) AS NUMERO_CALLE,
  coalesce(stg.DEPARTAMENTO_CODIGO, fact.DEPARTAMENTO_CODIGO) AS DEPARTAMENTO_CODIGO,
  coalesce(stg.CODIGO_POSTAL, fact.CODIGO_POSTAL) AS CODIGO_POSTAL,
  coalesce(stg.LOCALIDAD, fact.LOCALIDAD) AS LOCALIDAD,
  coalesce(stg.TELEFONO, fact.TELEFONO) AS TELEFONO,
  coalesce(stg.COORDENADA_X, fact.COORDENADA_X) AS COORDENADA_X,
  coalesce(stg.COORDENADA_Y, fact.COORDENADA_Y) AS COORDENADA_Y,
  coalesce(stg.CLIENTE_EXCLUSIVO, fact.CLIENTE_EXCLUSIVO) AS CLIENTE_EXCLUSIVO,
  coalesce(stg.VINCULO_EMPRESA, fact.VINCULO_EMPRESA) AS VINCULO_EMPRESA,
    CASE
      WHEN fact.CLIENTE_ID IS NULL THEN 'N'
      WHEN fact.CLIENTE_ID = stg.CLIENTE_ID
          AND fact.ESTADO_REGISTRO = 'A'
          AND (fact.CANAL_ID <> stg.CANAL_ID
           OR fact.SUBCANAL_ID <> stg.SUBCANAL_ID
           OR fact.SEGMENTO_MKT_ID <> stg.SEGMENTO_MKT_ID
           OR fact.SEGMENTO_VTA_ID <> stg.SEGMENTO_VTA_ID
--          OR fact.TERRITORIO_ID <> stg.TERRITORIO_ID
          ) THEN 'C'
       WHEN (fact.CLIENTE_ID = stg.CLIENTE_ID
          AND fact.ESTADO_REGISTRO = 'A'
          AND fact.CANAL_ID = stg.CANAL_ID
          AND fact.SUBCANAL_ID = stg.SUBCANAL_ID
          AND fact.SEGMENTO_MKT_ID = stg.SEGMENTO_MKT_ID
          AND fact.SEGMENTO_VTA_ID = stg.SEGMENTO_VTA_ID
--          AND fact.TERRITORIO_ID <> stg.TERRITORIO_ID
        )
        OR fact.CLIENTE_ID = -1 THEN 'U'
      WHEN fact.CLIENTE_ID IS NOT NULL AND stg.CLIENTE_ID IS NULL THEN 'V'
    ELSE '-1'
    END AS FLAG
FROM VIEW_TMP_COM_UY stg
FULL JOIN LK.DIM_CLIENTES_HIST_UY fact ---tabla dimensional
  ON stg.SOCIEDAD_ID = fact.SOCIEDAD_ID
  AND stg.CLIENTE_ID = fact.CLIENTE_ID
--filtro de la fact solo los registros vigentes (fecha_fin 2199 o null o
WHERE fact.FECHA_FIN_HISTORIA = '2199-12-31'
OR FACT.FECHA_FIN_HISTORIA IS NULL
OR fact.ESTADO_REGISTRO = 'I'
);
```

Figura 29 Vista temporal con full join entre tabla origen y dimensional clientes histórica.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 29 se crea una consulta que hace un join entre los registros de la tabla origen y la tabla dimensional de clientes, pero también se agregaron condiciones para poder tener una historia de los clientes de la dimensión.

Estas condiciones se basan en los IDS presentes en la tabla dimensional de clientes. Esta tabla tiene un total de cinco campos que son ids que se

mencionan a continuación: CLIENTE_ID, CANAL_ID, SUBCANAL_ID, SEGMENTO_MKT_ID y SEGMENTO_VTA_ID.

Si alguno de estos registros se modifica se debe crear una nueva clave subrogada para el cliente que en nuestro Data Lake es la Primary Key.

Adicionalmente se añadió un campo a la dimensión llamado ESTADO_REGISTRO, el cual puede tener dos valores:

- "A" indica que es el registro actualmente activo.
- "I" indica que es un dato anterior de ese cliente, es decir, un registro inactivo.

Para poder contemplar todos estos cambios de registros se añadió un campo "Flag" que puede tomar 4 valores y según dicho valor aplicar una lógica diferente. La mencionada lógica y posibles casos se detallan a continuación:

- El cliente existe en tabla origen y en tabla dimensional, pero hubo un cambio en alguno de los IDS mencionados (canal_id, subcanal_id, segmento_mkt_id o segemento_vta_id): se debe cerrar el registro vigente (pasar a estado inactivo) y crear un nuevo registro (Flag= "C").
- El cliente existe en tabla origen y en tabla dimensional sin cambios en alguno de sus ids. Sin embargo, se modificaron sus descripciones (Flag= "U")
- El cliente existe en tabla origen, pero no existe en tabla dimensional: es un nuevo registro (Flag= "N")
- El cliente no existe en tabla origen, pero sí existe en tabla dimensional: el cliente ya no está vigente, se debe cerrar el registro y pasarlo a estado inactivo (Flag = V)
- Puede pasar que un cliente que se desactivó en alguna fecha vuelva a reactivarse. En este caso se debe crear un nuevo registro (una nueva PK) con el cliente_id vigente.

Para saber si un cliente existe o no en las tablas se utiliza su código único de identificación, es decir, el campo cliente_id.

El resultado de esta consulta SQL se guarda en una vista temporal la cual será utilizada en la siguiente celda de la notebook. En dicha celda se crea otra vista temporal final con todos los posibles casos mencionados anteriormente y se realiza la lógica necesaria para contemplar cada uno de ellos.

Por último, se realiza el proceso de guardado en forma exactamente igual a la descripta en la Figura 27, pero cambiando la ruta en la que se guarda la dimensión.

Con este proceso realizado para todas las dimensiones mencionadas en la OLA 2 de análisis se está en condiciones de comenzar a desarrollar las tablas de hechos.

2.6.2 Proceso de desarrollo de tablas de hechos

Como se mencionó con anterioridad en este proyecto se desarrollan dos tablas de hechos: la tabla de hechos ventas diarias y la tabla de hechos objetivos ventas.

En primer lugar, se detallará el desarrollo de la tabla de hechos ventas diarias y luego el de la tabla objetivos ventas.

2.6.2.1 Desarrollo de tabla de hechos ventas diarias:

Para el desarrollo de esta tabla, en primer lugar, es necesario transformar los procesos ETL generados tanto en GeneXus como en los procedimientos almacenados del Data Warehouse BIPROD en consultas SQL en nuestro Data Lake, es decir, en consultas utilizando Spark SQL. Para realizar este proceso de transformación, se realizó un análisis del código GeneXus y también del código SQL del DW.

El proceso de desarrollo de la tabla de hechos se dividirá en dos notebooks. Una primer notebook, donde se creará una tabla ODS con el fin de

obtener los indicadores provenientes de las fuentes origen (en este caso Truck) y una segunda notebook, donde se creará la tabla de hechos utilizando la tabla ODS generada en primer lugar y las tablas dimensionales mencionadas anteriormente.

Finalmente, para los países Argentina, Paraguay y Uruguay la tabla de ODS proveniente de Truck se unirá a otra tabla ODS generada con información extraída de SAP (donde los datos ya vienen procesados y solo es necesario extraerlos) para generar una tabla ODS final que contenga información de las dos fuentes orígenes mencionadas (Truck y SAP). En el caso de Bolivia, este país no tiene datos provenientes de SAP por lo cual la tabla ODS y la tabla de hechos tiene como única fuente el sistema origen Truck.

Desarrollo de tabla ODS:

Vista temporal para generar tabla ODS Ventas Diarias

```
%sql
DROP VIEW IF EXISTS DECO_VENTAS_ODS;
CREATE OR REPLACE TEMP VIEW DECO_VENTAS_ODS AS
SELECT
 LEFT(cabe.CLMFCHEMI, 8)                                          AS FECHA_EMISION_VENTA
,LEFT(ppedcb.PEDFCHPED, 8)                                        AS FECHA_PEDIDO_VENTA
,LEFT(planil.Plafchsal, 8)                                        AS FECHA_SALIDA_VENTA
,LEFT(planil.plafchcie, 8)                                        AS FECHA_CIERRE_VENTA
,CAST(coalesce(det.clmiddps, -1) as int)                          AS DEPOSITO_ID
,CAST(coalesce(soc.empid, -1) as int)                             AS SOCIEDAD_ID
,CAST(coalesce(subcan.invmertpo, -1) as int)                      AS CANAL_ID
,CAST(coalesce(subcan.clivalinvm, -1) as int)                     AS SUBCANAL_ID
,CAST(coalesce(pcto.prdid, -1) as int)                            AS PRODUCTO_ID
,CAST(coalesce(seg_mkt.sgmvtaid, -1) as int)                      AS SEGMENTO_VTA_ID
,CAST(coalesce(seg_mkt.sgmmktid, -1) as int)                      AS SEGMENTO_MKT_ID
,CAST(coalesce(terr.trrid, -1) as int)                            AS TERRITORIO_ID
,coalesce(CAST(CAST(cabe.clmnroint AS int) AS STRING), "-1")      AS NUMERO_INTERNO
,CAST(coalesce(planil.planro, -1) as int)                         AS PLANILLA_ID
,CAST(coalesce(cli.cliid, -1) as int)                             AS CLIENTE_ID
,CAST(coalesce(cabe.codmovid, -1) as int)                         AS MOVIMIENTO_ID
,frmpago.fmapgotpop                                               AS FORMA_PAGO
,pcto.prdflgnps                                                   AS PRODUCTO_ESTADISTICO
,mov.codmovflgm                                                   AS FLG_MUEVE_MERCADERIA
,pcto.prdvalcap                                                   AS N_CAPACIDAD_UNIDAD_LT
,det.linprc                                                       AS PRECIO_UNITARIO
,CAST(COALESCE(uni.CnvUniCft,0) as int)                           AS Q_UND_POR_EMPAQUE
```

Figura 30 Consulta inicial para obtener campos de Truck.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 30 de describen los campos que se obtienen de las diferentes tablas de Truck y los cuales no necesitan de un proceso ETL

sofisticado para su obtención, solo se hacen pequeñas transformaciones de tipos de datos o limpiezas de espacios en blanco o caracteres especiales.

```
,case
    when
      CodAdcId is null
    and mov.CodmovFlgM = 'S'
    and mov.MovTpoPgo <> 'CSG'
    and mov.MovHisFlg <> 'C'
    and not (mov.CodmovFlgV = 'N'
            and mov.MovTpoPgo <> 'PND'
            and mov.CodmovTpoT = 'F'
            and mov.CodmovFlgE = 'N'
            and mov.MovHisFlg = 'N')
    then (LinCutCjeU + LinCutVtaU) *
    (1 -2 * mov.codmovcrd)
    when
      CodAdcId is null S
      and mov.MovHisFlg = 'C'
      and mov.CodmovFlgV = 'S'
      and mov.CodmovFlgM = 'N'
      and mov.CodmovFlgM = 'S'
      then ((LinCntPrm + LinCutCjeU + LinCutVtaU) *
      (1 -2 * mov.codmovcrd)) * -1
    else 0
    end                                                    AS Q_UNID_VENTA
,case
    when
      (CodAdcId = 'SCARGO' or LinCntPrm <> 0)
      and f.PolId is not null
      then (LinCntPrm+(LinCutCjeU+LinCutVtaU) *
      case
        when
        CodAdcId = 'SCARGO'
        then 1
      else 0
      end ) *(1-2*codmovcrd)
    else 0
    end                                                    AS Q_UND_SCARGO_AUT
```

Figura 31 Consulta para obtener algunos indicadores desde Truck.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 31 se observa el cálculo de dos indicadores (Q_UNID_VENTA y Q_UND_SCARGO_AUT), que luego formarán parte de la tabla de hechos final. Estos indicadores están presentes en la tabla ODS debido a que su origen son directamente las tablas de Truck.

En la tabla de hechos final aparte de funcionar como indicadores también se transformarán en campos que se utilizarán para obtener otros KPIS de la tabla. Es decir, a partir de estos indicadores se aplicarán ciertas

transformaciones y lógicas para obtener KPIS que también formen parte de la tabla de hechos.

Es necesario detallar que aparte de los indicadores descriptos en la Figura 31 en la tabla ODS también se aplica la lógica para obtener los KPIS que se mencionan a continuación: Q_UND_SCARGO_MAN, Q_UND_SCARGO _NEG, N_IMPORTE_FACTURADO, N_IMPORTE_DTO, N_IMPORTE_MFI NANCIERO y N_IMPORTE_MFIN_CANJE.

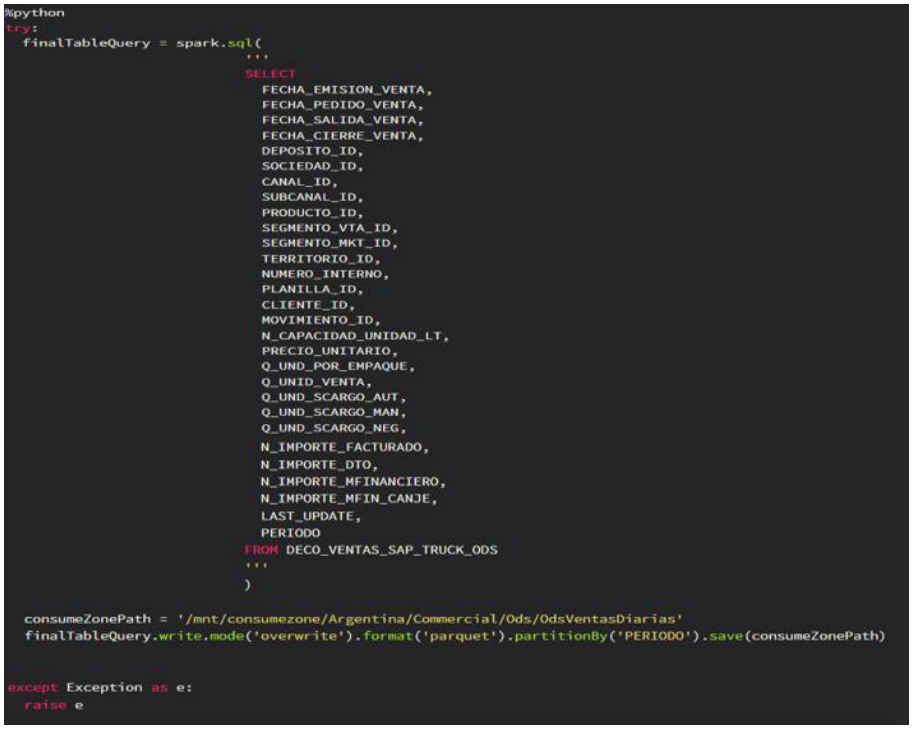
También resulta necesario mencionar cuáles son las tablas de Truck más importantes que se encuentran involucradas en la obtención de los indicadores, estas se mencionan a continuación:

- PCLMOV: contiene los movimientos operacionales de las planillas.
- PSTMLN: de ella se obtiene el detalle de las transacciones de planillas.
- PTBL03: contiene los movimientos adicionales de las planillas.
- PLANIL: provee información detallada de las planillas.
- PSTOCK: provee información de los productos vendidos.
- PTBL53: de ella se obtiene la forma de pago realizada por el cliente.
- INVER: provee información del canal utilizado.
- PCLMER: provee información del subcanal utilizado.
- PTBL01: indica de qué país son las transacciones.
- PTBL30: informa el territorio donde se realizaron las ventas.
- SGMMKT: provee información del segmento de marketing.
- PPOLDF: define las políticas de venta utilizadas.
- PPEDCB: informa los pedidos realizados por cada cliente.

Una vez generada la tabla ODS con todos los datos provenientes de Truck es necesario realizar el mismo proceso con los datos obtenidos de SAP. Para ello el proceso fue mucho más sencillo ya que los datos que se replican al DL desde SAP ya están transformados y limpios. Por este motivo, solo fue

necesario realizar transformaciones de tipos de datos para que coincidan con los campos de la tabla ODS de Truck.

Una vez que se tienen ambas tablas ODS desarrolladas solo resta generar una vista temporal final. En esta vista se realiza una consulta SQL de "UNION ALL" con el fin de justamente unir la información proveniente de cada las dos tablas. Esta vista es la que luego se guardará en una ruta dentro del Data Lake.



```
%python
try:
  finalTableQuery = spark.sql(
                            '''
                            SELECT
                              FECHA_EMISION_VENTA,
                              FECHA_PEDIDO_VENTA,
                              FECHA_SALIDA_VENTA,
                              FECHA_CIERRE_VENTA,
                              DEPOSITO_ID,
                              SOCIEDAD_ID,
                              CANAL_ID,
                              SUBCANAL_ID,
                              PRODUCTO_ID,
                              SEGMENTO_VTA_ID,
                              SEGMENTO_MKT_ID,
                              TERRITORIO_ID,
                              NUMERO_INTERNO,
                              PLANILLA_ID,
                              CLIENTE_ID,
                              MOVIMIENTO_ID,
                              N_CAPACIDAD_UNIDAD_LT,
                              PRECIO_UNITARIO,
                              Q_UND_POR_EMPAQUE,
                              Q_UNID_VENTA,
                              Q_UND_SCARGO_AUT,
                              Q_UND_SCARGO_MAN,
                              Q_UND_SCARGO_NEG,
                              N_IMPORTE_FACTURADO,
                              N_IMPORTE_DTO,
                              N_IMPORTE_MFINANCIERO,
                              N_IMPORTE_MFIN_CANJE,
                              LAST_UPDATE,
                              PERIODO
                            FROM DECO_VENTAS_SAP_TRUCK_ODS
                            '''
                            )

  consumeZonePath = '/mnt/consumezone/Argentina/Commercial/Ods/OdsVentasDiarias'
  finalTableQuery.write.mode('overwrite').format('parquet').partitionBy('PERIODO').save(consumeZonePath)


except Exception as e:
  raise e
```

Figura 32 Guardado de la tabla ODS de ventas diarias en el DL.
Fuente: Elaboración propia (basada en la práctica).

Por último, en la Figura 32 se realiza el guardado de la tabla ODS de ventas diarias en una ruta dentro del Data Lake, específicamente en la ruta 'mnt/consumezone/Argentina/Commercial/Ods/OdsVentasDiarias'.

El proceso de desarrollo de las tablas ODS fue el mismo para todos los países exceptuando a Bolivia que, como ya se mencionó, no cuenta con datos provenientes del sistema SAP y, por este motivo, en ese desarrollo no fue necesario realizar la consulta SQL de "UNION ALL" detallada anteriormente.

Ahora que ya tenemos la tabla ODS en nuestro DL, se genera una tabla externa de esta tal como se explicó en la Figura 28 para poder utilizar en otra notebook donde se desarrollará la tabla de hechos final.

Desarrollo de tabla de hechos:

Vista para generar la FT_VENTAS_DIARIAS

```
%sql
DROP VIEW IF EXISTS FT_VENTAS_DIARIAS_TOTAL;
CREATE OR REPLACE TEMP VIEW FT_VENTAS_DIARIAS_TOTAL AS
SELECT
  vtas.FECHA_EMISION_VENTA                                  AS fk_tiempo_emision_venta,
  vtas.FECHA_PEDIDO_VENTA                                   AS fk_tiempo_pedido_venta,
  vtas.FECHA_SALIDA_VENTA                                   AS fk_tiempo_salida_venta,
  vtas.FECHA_CIERRE_VENTA                                   AS fk_tiempo_cierre_venta,
  deps.PK_DEPOSITO                                          AS fk_deposito,
  soc.PK_SOCIEDAD                                           AS fk_sociedad,
  cli.PK_CLIENTE_HIST                                       AS fk_cliente_hist,
  can.PK_CANAL                                              AS fk_canal,
  pcto.PK_PRODUCTO                                          AS fk_producto,
  seg.PK_SEGMENTO                                           AS fk_segmento,
  ter.PK_EC_TERRITORIO                                      AS fk_ec_territorio,
  mov.PK_MOVIMIENTO                                         AS fk_movimiento,
  pla.PLANILLA_ID                                           AS fk_planilla,
  Round(
  SUM(
      (
        (
          (vtas.Q_UNID_VENTA + vtas.Q_UND_SCARGO_AUT) +
          (vtas.Q_UND_SCARGO_MAN + vtas.Q_UND_SCARGO_NEG)
        ) * vtas.N_CAPACIDAD_UNIDAD_LT
      ) / 100
      ),4)                                                  AS hl_venta_bruta,
```

Figura 33 Comienzo de la vista temporal para generar la tabla de hechos ventas diarias.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 33 se puede apreciar la creación de la vista temporal para generar la tabla de hechos, la misma se nutre de los campos generados en la ODS (todos aquellos campos que comienzan con el alias “vtas”) y los campos provenientes de las dimensiones analizadas con anterioridad.

También se puede apreciar el cálculo de uno de los indicadores calculados en la tabla (hl_venta_bruta) el cual como ya se había mencionado se calcula utilizando los KPIS generados en la tabla ODS. En este caso se utilizan los siguientes campos: Q_UND_VENTA, Q_UND_SCARGO_AUT, Q_UND_SCARGO_MAN, Q_UND_SCARGO_NET y N_CAPACIDAD_UNIDAD _LT.

Para obtener las distintas PKS de las dimensiones se hacen joins (encontrar los registros coincidentes) entre la tabla ODS y las distintas tablas dimensionales. Esto se puede realizar debido a que existe el mismo id en la

tabla ODS y en las tablas de dimensiones lo que permite obtener las PKS de la segunda tabla mencionada, mediante la coincidencia de este código o id. De esta manera, en nuestra tabla de hechos tenemos todas las PKS de las tablas de dimensiones (en esta tabla se llaman FK (Foreign Key o Clave Foránea)) y los indicadores calculados lo que permite saber, según cada indicador, el contexto que lo rodea.

Para obtener ese contexto solo es necesario hacer un join entre la tabla de hechos y la tabla dimensional mediante la FK (en la tabla de hechos) y la PK (en la tabla dimensional) y traer todos los campos descriptivos que necesitemos conocer. Por ejemplo, si tenemos como indicador el volumen de hectolitros vendidos y queremos saber cuánto compró cada cliente y cuál es el nombre tenemos que hacer un join entre la tabla de hechos y la tabla dimensional de clientes y se podrá obtener toda esta información. Esta es una manera sencilla de describir la forma en que se desarrollan los reportes o consultas de validación en SQL (esto se detallará un poco más en profundidad en la OLA 4 y 5).

Finalmente, la vista temporal de la tabla de hechos ventas diarias generada en la consulta SQL que se observa en la Figura 33, se guarda en una ruta del Data Lake tal como se ha descripto anteriormente (Figura 27).

2.6.2.2 Desarrollo de tabla de hechos objetivos ventas:

Desarrollo de tabla ODS

Para el desarrollo de la tabla de hechos objetivos ventas también se realizará un ODS, pero la diferencia principal de esta tabla con la de ventas diarias es que se nutre de datos provenientes del sistema origen GVQ, para ser específicos de dos tablas replicadas y necesarias para este proyecto las cuales se detallan:

- DLK_OBJ_MES: contiene los objetivos de ventas de la empresa a nivel mensual.

- DLK_OBJ_PESOS_DIAS: basándose en la tabla anterior contiene un cálculo del peso u objetivo de venta de la empresa a nivel diario.

```
%sql
DROP VIEW IF EXISTS OBJETIVOS_VENTAS_ODS;
CREATE OR REPLACE TEMP VIEW OBJETIVOS_VENTAS_ODS AS (
  SELECT
    LEFT(P.DIA, 10)                                    AS TIEMPO_DIA,
    M.SOCIEDAD_ID                                      AS SOCIEDAD_ID,
    CAST(M.PRODUCTO_ID AS INT)                         AS PRODUCTO_ID,
    CAST(M.SUBREGION_ID AS INT)                        AS SUBREGION_ID,
    CAST(M.TERRITORIO_ID AS INT)                       AS TERRITORIO_ID,
    CAST(M.CLIENTE_ID AS INT)                          AS CLIENTE_ID,
    P.PESO_DIA                                         AS PESO_DIA,
    M.Q_CUOTA                                          AS Q_CUOTA,
    CAST(DATE_FORMAT(CURRENT_DATE,'yyyyMMdd') AS INT)  AS LAST_UPDATE,
    CAST(M.ANIOMES AS INT)                             AS PERIODO
  FROM
    LK.DlkObjMes M
    INNER JOIN LK.DLK_OBJ_PESOS_DIAS_AR P ON M.SUBREGION_ID = P.SUB_REGION_ID
    AND M.PERIODO_ID = P.PERIODO_ID)
```

Figura 34 Vista temporal de la tabla ODS objetivos ventas.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 34 se detalla la consulta SQL realizada para obtener los distintos campos que conforman la tabla ODS de objetivos ventas. Dicha tabla cuenta con diez campos provenientes de las dos tablas de GVQ mencionadas de las cuales se obtienen datos mediante un join entre los campos SUBREGION_ID y REGION_ID.

De esta tabla los dos campos más importantes son el PESO_DIA proveniente de la tabla DLK_OBJ_PESOS_DIAS y el campo Q_CUOTA de la tabla DLK_OBJ_MES debido a que se utilizará la tabla de hechos para el cálculo del indicador.

Esta tabla ODS se almacena en el DL de la forma descripta en la Figura 27.

Desarrollo de tabla de hechos:

Una vez que tenemos nuestra ODS de objetivos ventas finalmente podemos desarrollar la tabla de hechos. Para ello es necesario combinar o joinear los registros provenientes de la tabla ODS haciendo los cálculos y transformaciones necesarias para obtener los KPIS de la tabla con las tablas de dimensiones ya analizadas con el fin de obtener las claves primarias de estas.

Vista para generar la FT_OBJETIVOS_VENTAS

```
%sql
DROP VIEW IF EXISTS FT_OBJETIVOS_VENTAS_UY;
CREATE OR REPLACE TEMP VIEW FT_OBJETIVOS_VENTAS_UY AS
SELECT
  CAST(REPLACE(obj.TIEMPO_DIA,'-','') AS INT)                AS fk_tiempo_dia,
  soc.PK_SOCIEDAD                                            AS fk_sociedad,
  cli.PK_CLIENTE_HIST                                        AS fk_cliente_hist,
  can.PK_CANAL                                               AS fk_canal,
  seg.PK_SEGMENTO                                            AS fk_segmento,
  pcto.PK_PRODUCTO                                           AS fk_producto,
  ter.PK_EC_TERRITORIO                                       AS fk_ec_territorio,
  SUM((obj.Q_CUOTA * obj.PESO_DIA) / 100)                    AS hl_objetivo,
  CAST(DATE_FORMAT(CURRENT_DATE,'yyyyMMdd') AS INT)          AS last_update,
  obj.PERIODO                                                AS periodo
FROM lk.Ods_Objetivos_Ventas_Uy obj
---dim sociedades
INNER JOIN lk.Dim_Sociedades_Uy soc
    ON soc.SOCIEDAD_ID = 2
--dim productos
INNER JOIN lk.Dim_Productos_Uy pcto
    ON pcto.PRODUCTO_ID = obj.PRODUCTO_ID
--dim cliente historico
INNER JOIN (SELECT PK_CLIENTE_HIST, CLIENTE_ID,SOCIEDAD_ID, SEGMENTO_VTA_ID, SEGMENTO_MKT_ID,
CANAL_ID, SUBCANAL_ID
FROM (SELECT estado_registro,ROW_NUMBER() OVER (PARTITION BY cliente_id
ORDER BY cliente_id, fecha_fin_historia desc) AS fila , * FROM lk.dim_clientes_hist_Uy)
WHERE fila =1) cli
    ON cli.CLIENTE_ID = obj.CLIENTE_ID
--dim canales
INNER JOIN lk.Dim_Canales_Uy can
    ON cli.canal_id = can.canal_id
    AND cli.subcanal_id = can.subcanal_id
--dim segmentos
INNER JOIN lk.Dim_Segmentos_Uy seg
    ON seg.segmento_venta_id = cli.segmento_vta_id
    AND seg.segmento_mkt_id = cli.segmento_mkt_id
--dim territorio
INNER JOIN (SELECT * FROM
                (SELECT estado_registro,ROW_NUMBER() OVER
                (PARTITION BY territorio_id ORDER BY territorio_id, fecha_fin_historia desc)
                AS fila , * FROM lk.dim_territorio_hist_Uy)
            WHERE fila =1) ter
    ON ter.TERRITORIO_ID = obj.TERRITORIO_ID
WHERE obj.PERIODO > 202012
GROUP BY 1,2,3,4,5,6,7,9,10
```

Figura 35 Vista temporal de la tabla de hechos objetivos ventas.
Fuente: Elaboración propia (basada en la práctica).

En esta Figura 35 se observa la vista temporal final de la tabla de hechos objetivos ventas a través de la cual se puede conocer el volumen de hectolitros objetivo, es decir, el volumen de HL que se quiere vender.

Este es el indicador principal y podemos obtener el contexto de este a partir de cada una de las dimensiones (sociedades, productos, clientes, canales

y segmentos) que están “combinadas” con la tabla ODS y se observan en la figura mencionada.

Por último, esta vista temporal también es almacenada en el Data Lake como se describió en la Figura 27 y se le crea una tabla externa tal como se observa en la Figura 28.

2.6.3 Desarrollo de pipelines

Para finalizar los desarrollos de las tablas de hechos descriptas, el último paso es crear los PPLS de canalización que ejecuten las notebooks en forma automática utilizando triggers.

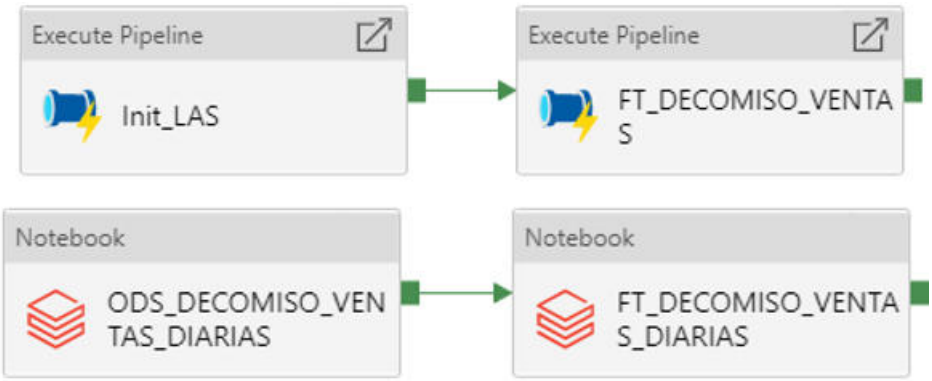


Figura 36 Desarrollo de PPL para la tabla de hechos ventas diarias.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 36 se observa en la parte superior dos pipelines, el primero llamada “Inir_LAS” tiene como función conectarnos a las bases de datos del DL y obtener todos los parámetros necesarios para la ejecución de notebooks en Databricks. El segundo PPL tiene como función llamar a otro pipeline que contiene las notebooks que se observan en la parte inferior de la Figura 36.

Estas dos notebooks son las ya descriptas, en primer lugar, está la notebook que contiene la tabla ODS desarrollada para ventas diarias y, en segundo lugar, se observa la notebook que contiene el desarrollo de la tabla de hechos ventas diarias.

Estos pipelines se encuentran “triggereados”, es decir, que los PPLS tiene desarrollado un triggers en su interior que tiene como función ejecutar los procesos en forma automática. Estos procesos del PPL (las notebooks de ODS y tabla de hechos de ventas diarias) se ejecutarán en forma automática todos los días a partir de las 9 am hora argentina.

Con todos los pasos descriptos en esta etapa, finalmente, se tiene el modelo dimensional de gestión comercial replicado al Data Lake, utilizando ADF para la ingesta de tablas y la ejecución de los pipelines en forma automática y Databricks para los procesos ETL y la creación del modelo.

2.7 Ola 4: Pruebas y ajustes del modelo

En esta ola se realizaron todas las pruebas de validación de datos entre el nuevo modelo dimensional generado en el Data Lake y el modelo OLAP que ya existía en el Data Warehouse BIPROD.

De estas validaciones se produjeron cambios y ajustes en las consultas SQL realizadas para la obtención de los indicadores de las tablas de hechos. Estos ajustes se realizaron debido a que en un principio las validaciones generaban resultados con muchas diferencias significativas entre el modelo de BIPROD y el modelo del DL, tanto en la cantidad de registros como en el valor de los KPIS.

Luego de estas pruebas de validación y los cambios y ajustes de las consultas se obtuvieron las detalladas en la OLA 3 y de esta forma los dos modelos coincidieron casi a la perfección, debido a que la réplica nunca va a ser exactamente igual ya que se realizan casteos y redondeos decimales en ambos modelos. Las diferencias se consideraron despreciables ya que eran a nivel decimal y tanto los registros como los indicadores tienen valores de millones o miles de millones por lo cual esta diferencia representaba un porcentaje inferior al 0.5 %.

Diferencias bandeja - truck 30/11

```
%sql
SELECT * FROM (
SELECT
      cabe.codmovid,
      BICXTIPMOV,
      g.DtoCod,
      BICPLANRO,
      BICCLIID,
      BICPRDID,
      (LinTotDtoE + LinTotDtoF + LinTotDtoL + LinTotDtoP
      + LinTotDtoR + LinTotDtos + LinTotBon)
      * (1 - 2 * MOV.codmovcrd) * case when g.DtoFchBja
      is not null or h.codadcid is not null then 0 else 1
      end  AS N_IMPORTE_DTO,
      BIIMPDTO,
      BISERNROCT AS COMPROBANTE,
      cabe.clmser,
      cabe.clmnrocte,
      BICFCHEMI,
      BICFCHCIE
    FROM
   Bivta3011 B
   left join PCLMOV cabe on B.BICPLANRO = cabe.clmnropla
   and b.BICCLIID = cabe.cliid
   and cabe.empid = b.BICEMPID
   and cabe.clmfchemi = b.BICFCHEMI
   and b.BICXTIPMOV = cabe.codmovid
   and trim(SUBSTRING(b.BISERNROCT,1,instr(b.BISERNROCT, '-')-1))
   = trim(cabe.clmser)
   and trim(SUBSTRING(b.BISERNROCT,instr(b.BISERNROCT, '-')+1,8))
   = trim(SUBSTRING(lpad(cast(cabe.clmnrocte as string),10, '0'),0,8))
   INNER JOIN PSTMLN det on cabe.clmnroint = det.clmnroint
   and cabe.clmnropla = det.ClmNroPla --detalle movimientos
   INNER JOIN PTBL03 mov on cabe.codmovid = mov.codmovid --planilla
   INNER JOIN PLANIL planil on cabe.empid = planil.empid
   and cabe.clmnropla = planil.planro
   AND det.prdid = B.BICPRDID ---productos
   INNER JOIN PCLIEN F ON F.empid = cabe.empid and F.cliid = cabe.cliid
   INNER JOIN PSTOCK pcto on det.prdid = pcto.prdid --forma de pago
   LEFT JOIN PTBL01 soc on soc.empid = cabe.empid
   left join MovDto g on g.DtoFchBja = '00000000'
   and g.DtoCod = cabe.codmovid
   left join PTB03E h on h.codmovadci = cabe.codmovid and  h.codadcid <> ''
 WHERE
   soc.empid = 5
 ) G WHERE BIIMPDTO <> N_IMPORTE_DTO
```

Figura 37 Validación del indicador N_IMPORTE_DTO para Paraguay.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 37 se observa un ejemplo de prueba de validación de un indicador de la tabla ODS que luego formaría parte de la tabla de hechos. El indicador de este ejemplo es el N_IMPORTE_DTO, que informa cuál es el importe descontado en una venta realizada.

Se puede observar que en la consulta SQL, en la sentencia SELECT hay campos provenientes de la tabla BIVTA (los que están escritos en mayúscula en la Figura 36) y campos provenientes de diferentes tablas de Truck. Los campos provenientes de la primer tabla (BIVTA) son una copia exacta del DW BIPROD, debido a que se trata de un archivo de texto provisto por el cliente que contiene los registros obtenidos mediante extracciones al Data Warehouse el cual fue replicado al DL para realizar estas validaciones y ajustes en las transformaciones ETL.

En la figura se puede observar que se realiza el cálculo del indicador N_IMPORTE_DTO utilizando las tablas provenientes de Truck y luego también se extrae el campo BIIMPDTO proveniente de la tabla BIVTA. El segundo campo es equivalente al indicador N_IMPORTE_DTO en el DW BIPROD, es decir, estos dos campos son el mismo, solo tienen nombres diferentes, pero deben tener los mismos valores. La diferencia radica en que BIIMPDTO proviene ya calculado del DW y el otro se calcula realizando lógicas en las que intervienen las tablas orígenes de Truck que fueron replicadas al DL.

Entonces para finalizar la validación del indicador hay que comparar los dos campos mencionados, si se obtiene alguna diferencia es porque la consulta SQL desarrollada sobre las tablas de Truck no es correcta. Por este motivo, se observa casi al final de la Figura 36 una sentencia WHERE, específicamente la que se describe: **"…WHERE BIIMPDTO <> N_IMPORTE_DTO"**, esto se traduce en obtener todos los registros de la consulta donde los valores campos BIIMPDTO y N_IMPORTE_DTO sean diferentes.

Entonces, si obtenemos resultados la conclusión es que la consulta SQL aplicada es incorrecta, caso contrario la consulta es correcta.

Teniendo esto presente, se puede observar en la última línea de la Figura 36 la siguiente frase: **"Query returned no results"** que se traduce en: la consulta no retornó resultados por lo que se puede concluir que la sentencia SQL implementada es correcta y, en consecuencia, el indicador N_IMPORTE_DTO está validado.

Este proceso de pruebas y validación se realizó para cada uno de los indicadores descriptos en la OLA 3 teniendo en cuenta diferentes fechas y agrupándolos según cada contexto provisto por las tablas dimensionales.

2.8 Ola 5: Implementación en producción y desarrollo de consultas específicas

Finalmente, con todo el modelo dimensional en el Data Lake puesto a prueba y validado se puede comenzar con la última etapa del proyecto, la implementación en producción y el desarrollo de consultas específicas para los reportes de Power BI.

2.8.1 Creación de consultas específicas necesarias para Power BI

Al comenzar a trabajar con los visualizadores de Power BI surgió la necesidad de crear una consulta específica en SQL que funcione como una tabla agregada de las tablas de hechos creadas anteriormente.

```
%sql
DROP VIEW IF EXISTS vw_ventas_objetivos_resum;
CREATE TEMP VIEW vw_ventas_objetivos_resum AS
 SELECT
  coalesce(vw_dr.fk_tiempo,vw_o.fecha) as fk_tiempo,
  coalesce(vw_dr.fk_sociedad,vw_o.fk_sociedad) as fk_sociedad,
  coalesce(vw_dr.fk_ec_territorio,vw_o.fk_ec_territorio) as fk_ec_territorio,
  coalesce(vw_dr.fk_canal,vw_o.fk_canal) as fk_canal,
  coalesce(vw_dr.fk_segmento,vw_o.fk_segmento) as fk_segmento,
  coalesce(vw_dr.fk_producto,vw_o.fk_producto) as fk_producto,
  hl_venta_bruta,
  hl_venta_neta,
  hl_objetivo
FROM vw_ventas_diarias_resumida vw_dr
FULL JOIN vw_objetivos_resumida vw_o
  ON (vw_o.fecha = vw_dr.fk_tiempo
      AND vw_o.fk_sociedad = vw_dr.fk_sociedad
      AND vw_o.fk_ec_territorio = vw_dr.fk_ec_territorio
      AND vw_o.fk_canal = vw_dr.fk_canal
      AND vw_o.fk_segmento = vw_dr.fk_segmento
      AND vw_o.fk_producto = vw_dr.fk_producto)
```

Figura 38 Vista temporal de tabla que combina la tabla de hechos ventas diarias y objetivos ventas. Fuente: Elaboración propia (basada en la práctica).

En la Figura 38 se observa la consulta específica realizada en SQL que crea una vista temporal. Esta vista combina información de las tablas de hechos ventas diarias y objetivos ventas.

Este desarrollo a vista surgió como una necesidad de los visualizadores de Power Bi que necesitaban tener los datos de las ventas junto a los objetivos de ventas en un único reporte de visualización.

A raíz de ello se procedió a realizar la consulta SQL que se aprecia en la Figura 38, que contiene información de ambas tablas de hechos con los indicadores relevantes para el reporte que se necesitaba realizar. En este caso se puede apreciar que tenemos la cantidad de hectolitros vendidos en forma bruta y neta (información proveniente de la tabla de hechos ventas diarias) y la cantidad de hectolitros objetivo (información proveniente de la tabla de hechos objetivo ventas). También tenemos campos por los cuales se realiza el join entre ambas tablas y los que proveen el contexto al “unirlos” con las tablas de dimensiones.

Esta vista temporal se creó para todos los países y también fueron necesarios algunos desarrollos adicionales requeridos para los reportes de Power Bi. Todas estas vistas se almacenaron en el Data Lake de la misma forma que se describió en la Figura 27 pero en su propia ruta.

Por último, esta vista almacenada en el DL se replicó al sistema Presto utilizando una notebook de Databricks para que de esta forma los visualizadores de Power BI se conecten a dicho sistema y pueda obtener los datos de las vistas y aplicar los filtros que consideren necesarios para el desarrollo de los reportes finales.

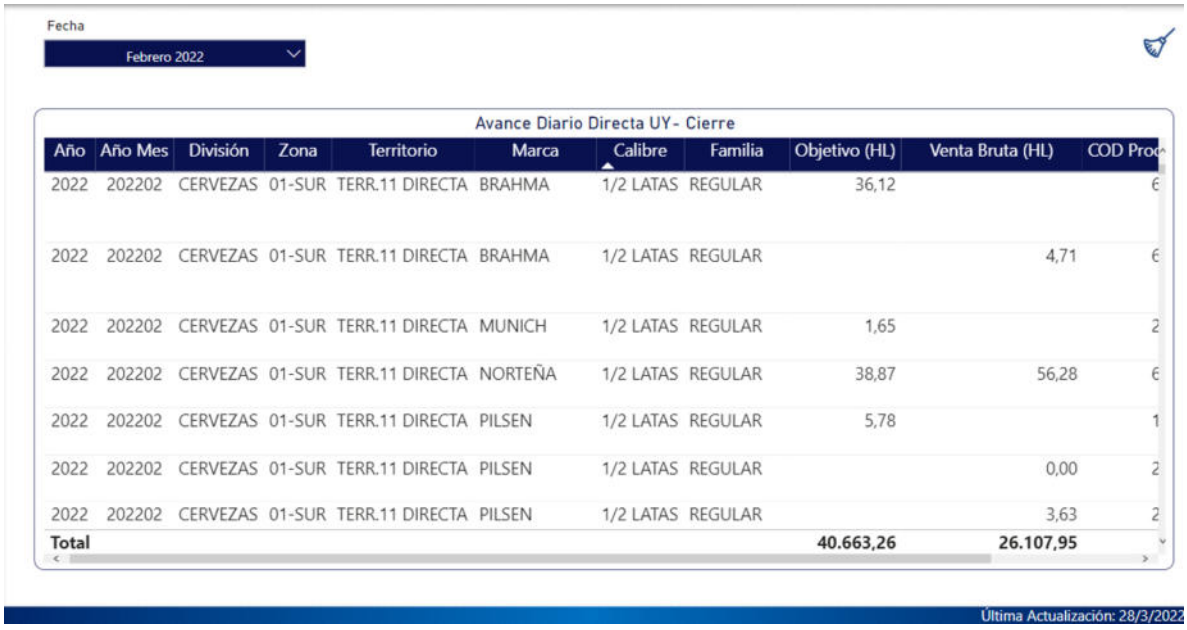


Figura 39 Reporte que combina las tablas de ventas diarias y objetivos ventas
Fuente: Elaboración propia (basada en la práctica).

En la Figura 39 se observa el reporte de Power BI que analiza el avance de ventas diario. Para ello muestra información de la venta de hectolitros diaria y el objetivo de venta de estos hectolitros agrupándolos por mes (febrero 2022), la división, la zona, el territorio, la marca, el calibre, la familia, etc.

Para realizar este reporte se utilizó la réplica en Presto de la vista temporal descripta en la Figura 38.

2.8.2 Proceso de implementación en producción

En primer lugar, lo que se debe hacer es crear una nueva rama de Git para poder guardar las notebooks y pipelines de desarrollo en esa rama para luego solicitar el pasaje de esta al ambiente productivo.

Las dos maneras que utilizamos para crear una rama de Git son realizarlo directamente desde la página de GitHub ingresando a nuestra cuenta o hacerlo

desde Azure Data Factory para que los pipelines de canalización que se creen y modifiquen se vayan guardando y actualizando automáticamente en la rama.

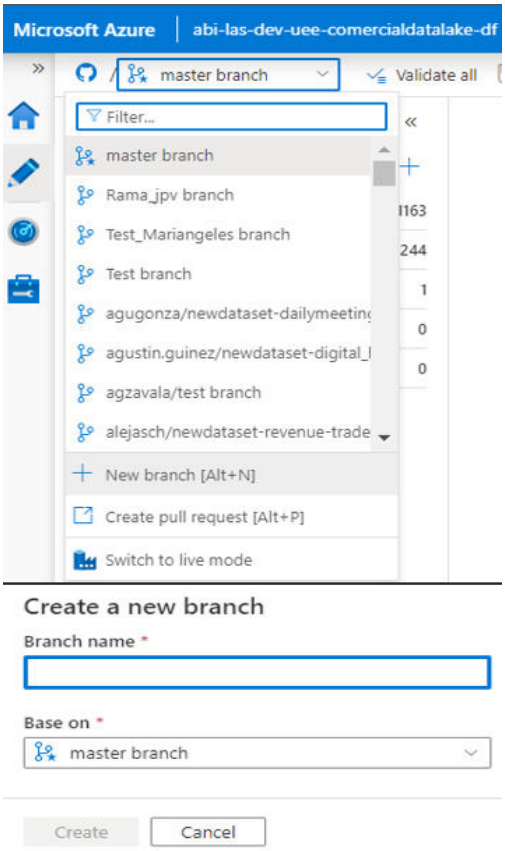


Figura 40 Crear rama en GitHub desde ADF.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 40 se observa el proceso para crear una nueva rama de GitHub desde Azure Data Factory. Se puede observar en la parte superior que en este ejemplo se está ubicado en la rama master y casi al final del listado desplegable aparece un ítem con un símbolo + y el texto “New Branch”, al hacer clic en este se abre una ventana emergente como se muestra en la parte inferior de la Figura 40 y allí se solicita que se ingrese un nombre para la nueva rama y se seleccione sobre cual rama se basará la nueva (para este ejemplo es la rama master).

Una vez elegido un nombre se podrá dar clic en el botón “Create” y ya se tendrá la rama actualizada con todos los pipelines y elementos que actualmente

se encuentran en master. Es necesario aclarar que todos los pipelines y artefactos que se hallan en la rama master fueron validados, aprobados y ya están presentes en el ambiente productivo.

Una vez ubicados en la nueva rama todos los cambios, actualizaciones o artefactos nuevos que se desarrollen dentro de Azure Data Factory se guardarán automáticamente en esta. Sin embargo, las notebooks en Databricks o documentación necesaria para los pasajes a producción no se guardan automáticamente y deben incorporarse siguiendo otros pasos que se detallarán.

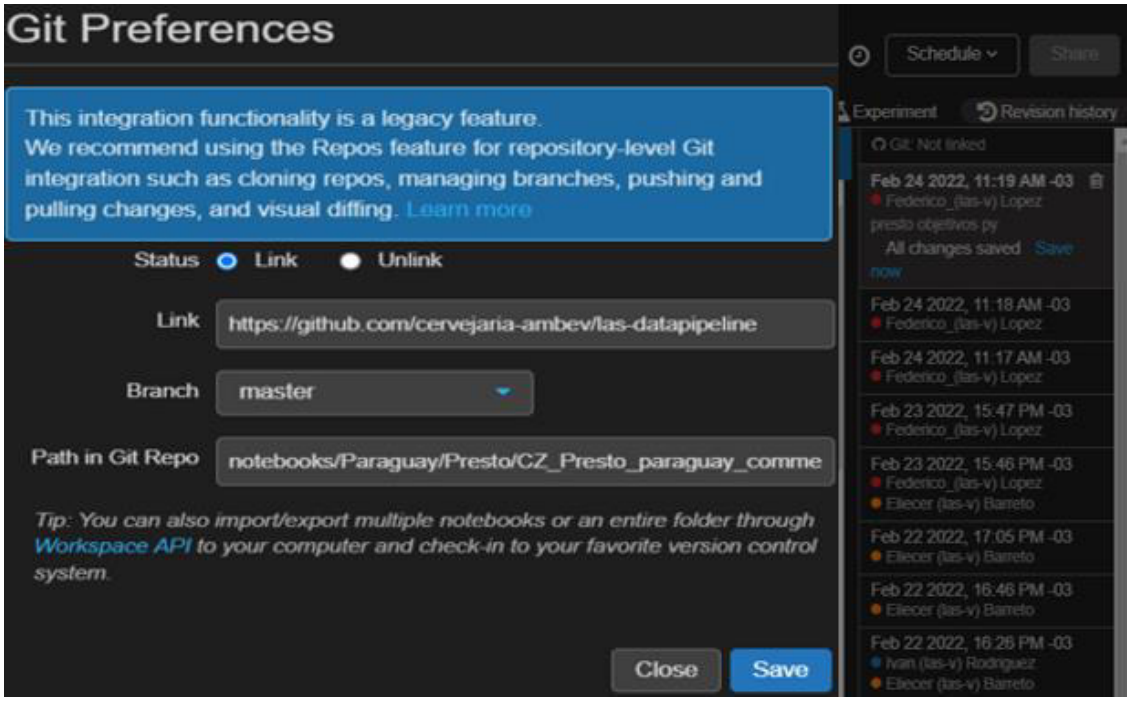


Figura 41 Añadir notebook de Databricks a una rama de GitHub.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 41 se aprecia cómo se añade una nueva notebook o los cambios de esta a la rama de GitHub que se creó.

El proceso es sencillo solo hace falta ubicarse en la notebook que se quiere agregar a la rama y en la parte superior derecha de esta se puede observar un botón con el texto “Revision History”. Este botón contiene el historial de cambios que se realizó en la notebook, junto con el nombre de los usuarios que realizaron dichos cambios, pero también allí se encuentra la integración de

Databricks con GitHub, se puede apreciar un botón con el texto “Git: Not linked” y al hacer clic en él se abre la ventana emergente que se observa a la izquierda de la Figura 41.

En esta ventana emergente es donde se puede “linkear” o añadir la notebook de Databricks que se desee a la rama de Git de preferencia, solo basta con dar clic en “Branch” y se desplegará una lista con todas las ramas que se encuentran creadas en el repositorio del cliente. Una vez que se selecciona la rama en la que se está trabajando hay que dar clic en “Save” y los cambios se añadirán a la rama seleccionada. Para añadir la documentación de pasaje a producción, solo es necesario ir a la página de GitHub, buscar la rama que se desee y seleccionar en las opciones la de añadir archivos, buscar estos en la computadora y subirlos a la rama.

El paso final para subir el trabajo realizado a producción es solicitar un “Pull Request” en Git.

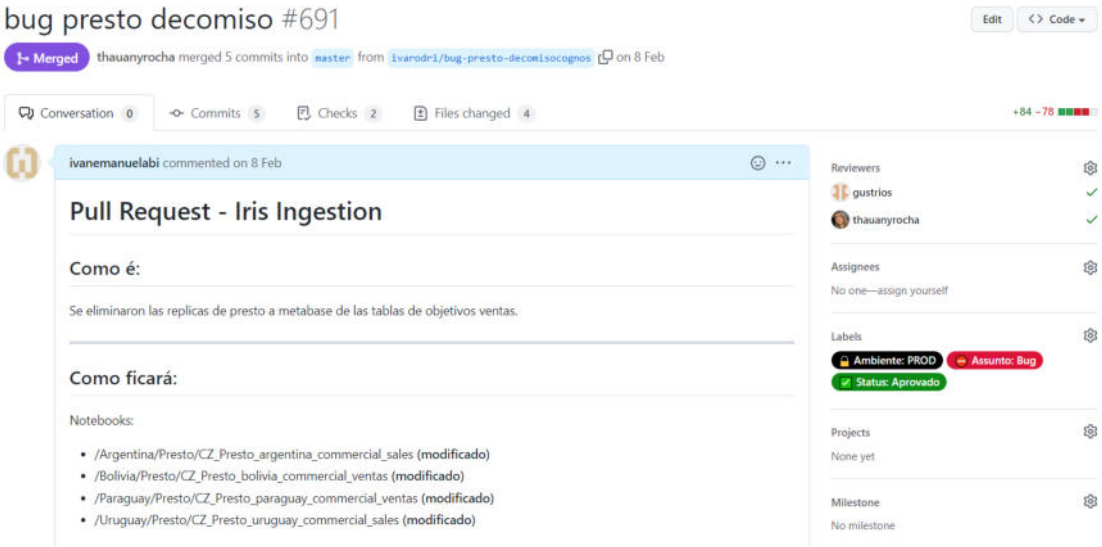


Figura 42 Solicitud de pasaje a producción en GitHub.
Fuente: Elaboración propia (basada en la práctica).

En la Figura 42 se observa una solicitud de pasaje a producción ya aprobada. Básicamente, una vez que están todos los cambios y el desarrollo fue validado hay que dirigirse a la página de a GitHub y dar clic en “Create Pull

Request" de la rama en la que se está trabajando, completar cuál fue el trabajo que se realizó tal como se observa en la imagen (notebooks, pipelines o documentación que fue modificada o creada desde cero), seleccionar una persona para que revise el trabajo (líder de ingeniería) y esperar a que los administradores de Git aprueben el pedido.

Una vez que se aprueba el trabajo ya se encuentra en el ambiente productivo y se ejecutará automáticamente mediante la utilización de Triggers.

3. Conclusiones

En base a los resultados obtenidos luego de las primeras semanas del uso de los reportes de Power Bi y la ejecución del proyecto en producción se llegó a la conclusión de que los tres objetivos propuestos en esta PPS fueron alcanzados.

El primer objetivo consistió en migrar la información que resultara relevante para el desarrollo de los reportes a la plataforma en la nube de Azure. El objetivo requirió de más tiempo que el estipulado, debido a que hubo que realizar varias ingestas de tablas origen que no se habían considerado en un inicio como relevantes para el proyecto, pero que finalmente fueron necesarias. Sin embargo, se ajustaron los tiempos acordes a estas necesidades y el objetivo se logró conseguir.

El segundo objetivo de lograr la reingeniería y adaptación del modelo dimensional OLAP de gestión comercial también presentó ciertas complicaciones a la hora de comprender el código escrito en GeneXus y adaptarlo a código SQL, pero a pesar de ello se lograron obtener los resultados y performance esperados del modelo.

Finalmente, el último objetivo de alimentar la herramienta de reportes Power Bi en forma eficiente también se considera alcanzado. Aunque, en la actualidad,

siguen surgiendo necesidades de adaptación de los reportes o realización de consultas específicas para lograr mayor performance o una respuesta más rápida.

En síntesis, el trabajo realizado colaboró para que la compañía cliente conozca cómo fueron pensados, desarrollados e implementados los reportes de ventas diarias y objetivos. Esto les permitirá en un futuro poder mantener y actualizar la información y herramientas que crean necesarias para los reportes. También les ofrece una visión para comprender en qué punto se encuentra la empresa a fines de convertirse en Data Driven (basar todas sus decisiones en datos) y les otorga herramientas para entender cuáles son los próximos pasos y objetivos a seguir para poder conseguirlo.

Reflexión sobre la práctica profesional supervisada como espacio de formación

Considero que la práctica profesional supervisada le aportó mucho valor a mi experiencia y formación como estudiante y empleado de una empresa de sistemas. Me permitió administrar, gestionar, organizar y desarrollar un proyecto de inicio a fin con la finalidad de alcanzar los objetivos planteados en la práctica en el tiempo y forma establecidos.

Siento que fortaleció mis habilidades y capacidades de interacción con otros sectores de la empresa, me permitió conocer e interiorizarme más en el negocio de la compañía y establecer lazos más fuertes con mis compañeros.

En cuanto a la redacción del informe fue un desafío muy interesante de cual no tenía mucha experiencia debido a que en el transcurso de la carrera he escrito otros informes, pero no del volumen y magnitud de este. Sin embargo, fue una experiencia muy gratificante que me permitió darme cuenta de las diferentes herramientas y habilidades adquiridas durante el transcurso de mi formación profesional.

Finalmente considero que la PPS es totalmente necesaria para que finalicemos nuestra formación y nos graduemos ya que aporta un gran valor agregado a nuestro desarrollo como profesionales de la informática. Nos permite aplicar todos los conocimientos y habilidades adquiridos durante la carrera. A medida que la desarrollamos podemos notar cómo cada una de las materias, de alguna forma, aportan su grano de arena a nuestro desarrollo profesional, algunas en mayor medida que otras, pero todas se complementan y nos permiten convertirnos en profesionales altamente capacitados.

4. Bibliografía

Apache Spark. Recuperado de https://spark.apache.org/. Consulta enero de 2022.

Azure Databricks. Recuperado de https://docs.microsoft.com/en-us/azure/databricks/scenarios/what-is-azure-databricks. Consulta enero de 2022.

Azure Data Factory. Recuperado de https://docs.microsoft.com/es-es/azure/data-factory/introduction. Consulta enero de 2022.

IBM Cognos. Recuperado de https://www.ibm.com/docs/es/cognos-analytics/10.2.2?topic=SSEP7J_10.2.2/com.ibm.swg.ba.cognos.wig_cr.10.2.2.doc/c_gtstd_c8_bi.html. Consulta febrero de 2022.

Ing. Bernabéu, R (2010) “HEFESTO: Metodología para la Construcción de un Data Warehouse”. Recuperado de https://www.businessintelligence.info/resources/assets/hefesto-v2.pdf. Consulta diciembre de 2021.

Ing. Cabanchik, A (s.f). “Introducción al Data Warehouse”. Consulta diciembre de 2021.

OLAP Avanzado. Recuperado de https://www.sinnexus.com/business_intelligence/olap_avanzado.aspx. Consulta diciembre de 2021.

Power BI. Recuperado de https://powerbi.microsoft.com/es-es/what-is-power-bi/. Consulta febrero de 2022.

Python. Recuperado de https://www.python.org/about/. Consulta enero de 2022.