UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Gestión de calidad de datos en arquitecturas de Big Data

Informe de Proyecto de Grado presentado por

Carolina Cortés Lasalle

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Dra. Adriana Marotta

Montevideo, 20 de julio de 2024

Agradecimientos

En primer lugar, quiero agradecer a Adriana por su orientación y apoyo. Gracias por transmitirme tu calma en los momentos en los que la necesité.

Gracias Lore y Cami por sus comentarios, ayudas y sugerencias para que este proyecto salga de la mejor manera posible.

Gracias a mi familia por su apoyo durante estos casi 6 años de carrera. En particular, quiero agradecer a mis padres por las oportunidades educativas que me dieron, las cuales fueron fundamentales para que yo hoy pueda estar culminando una carrera de grado.

También quiero agradecer a todos mis amigos y compañeros por todo su apoyo durante estos años de carrera, pero sobre todo en el transcurso de este proyecto. En especial, quiero agradecer a los compañeros que me dió esta carrera, no hubiera llegado hasta acá sin ustedes.

Por último, quiero agradecer a todas las personas que contribuyeron, de alguna manera, a mi formación académica, profesional y personal a lo largo de los años.

Resumen

El término "Big Data" (BD) hace referencia a grandes colecciones de datos heterogéneos, que se generan a altas velocidades (Marotta y Serra, s.f.). Hoy en día, estas colecciones de datos son utilizadas por herramientas de Analítica Avanzada (AA) y Business Intelligence (BI) (B. Inmon, Levins, y Srivastava, 2021). A lo largo de los años han surgido distintos sistemas para soportar los diversos requisitos de análisis de los datos en BD, como Data Warehouses (DW) y Data Lakes (DL). Sin embargo, ninguno de estos sistemas logra brindar soporte para heramientas de AA y BI, a la vez (B. Inmon y cols., 2021; Marotta y Serra, s.f.). Es por esto que en la actualidad, la comunidad académica busca hacer una transición a una nueva arquitectura de análisis de BD, denominada "Data Lakehouse", la cual logra integrar las capacidades de DW y DL en un mismo sistema unificado. Sin embargo, actualmente no existe una arquitectura establecida para este sistema, como sí existen para los DW y DL (Armbrust, Ghodsi, Xin, y Zaharia, 2021; B. Inmon y cols., 2021).

Por otro lado, debido al volumen de datos de BD y los distintos requerimientos de análisis impuestos por diferentes usuarios, la gestión de la calidad de datos cobra un papel sumamente relevante en estas arquitecturas y su gestión debe adaptarse a toda la variabilidad que estas presentan. En la actualidad no existen técnicas o metodologías globalmente aceptadas, que sean específicas para la gestión de calidad de datos en arquitecturas de este tipo (Ravat y Zhao, 2019a; Armbrust y cols., 2021; Zouari, Kabachi, Boukadi, y Ghedira Guegan, 2021; Nargesian, Zhu, Miller, Pu, y Arocena, 2019).

En respuesta a estas problemáticas, este trabajo propone una arquitectura genérica de BD que logre combinar las capacidades de DW y DL, incorporando la gestión de calidad de datos dentro de la misma. A su vez, para verificar la factibilidad técnica y tecnológica de la propuesta, se desarrolló un prototipo reducido de la arquitectura propuesta.

Palabras clave: Data Warehouse, Big Data, Data Lake, Data Lakehouse, Calidad de Datos.

Índice general

Glosario

BD *Big Data*

DW *Data Warehouse*

ETL *Extraction, Transformation and Load*

ELT *Extraction, Load and Transformation*

DBMS *Data Base Management System*

ACID *Atomicity, Consistency, Isolation, Durability*

BI *Business Intelligence*

DL *Data Lake*

DLH *Data Lakehouse*

AA *Analítica Avanzada*

DS *Data Science*

DST *Data Scientist*

ML *Machine Learning*

CD *Calidad de Datos*

GCD *Gestión de calidad de datos*

HDFS *Hadoop Distributed File System*

CES *Datos de centros educativos de Educación Secundaria*

CEIP *Datos de centros educativos de Educación Inicial y Primaria*

datos_estudiantes_año *Datos de acceso a plataformas de CEIBAL por parte de estudiantes para un año determinado*

acceso_plataformas *Datos de acceso a plataformas por parte de estudiantes para los años 2019, 2020, 2021 y 2022*

centros_educativos *Datos de centros educativos de ANEP*

Capítulo 1

Introducción

En este capítulo se presenta la motivación, el contexto, los objetivos y resultados esperados del proyecto. Al final del capítulo se presenta la organización del documento.

1.1. Motivación

El término "Big Data" hace referencia a grandes colecciones de datos heterogéneos, que se generan a altas velocidades (Marotta y Serra, s.f.). Hoy en día, estas colecciones de datos son utilizadas por herramientas de Analítica Avanzada y Business Intelligence (B. Inmon y cols., 2021).

El primer acercamiento a un sistema de Big Data fueron los Data Warehouses, los cuales se encargan de almacenar de forma centralizada datos de una organización, que son accedidos a través de herramientas de Business Intelligence para ayudar en la toma de decisiones. Debido a la diversidad de datos existentes hoy en día y el procesamiento requerido por distintos usuarios (Científico de Datos, Analista de Datos, entre otros), estos sistemas empezaron a presentar algunas desventajas; ya que no son capaces de almacenar datos no estructurados (audio, video, imágenes, entre otros), realizar procesamiento de datos en tiempo real y brindar soporte a herramientas de Analítica Avanzada (B. Inmon y cols., 2021; Marotta y Serra, s.f.).

Para eliminar algunas de las desventajas de los Data Warehouses, surgen los Data Lakes. Estos sistemas permiten almacenar grandes volúmenes de datos heterogéneos en sistemas de almacenamiento de bajo costo, los cuales son accedidos por herramientas de Analítica Avanzada. A pesar de remediar las desventajas de los Data Warehouse, la primera arquitectura de Data Lake presentaba la desventaja de convertirse en un "pantano de datos" (*data swamp*), debido a la cantidad de datos almacenados, sin ningún tipo de gestión de metadatos y procesamiento de los mismos (Hai, Quix, y Jarke, 2021; Giebler, Gröger,

Hoos, Schwarz, y Mitschang, 2019).

Actualmente, la comunidad académica busca realizar una transición hacia una tercera generación de arquitecturas de Big Data, a partir del surgimiento del concepto "Data Lakehouse". Este término hace referencia a un sistema de Big Data el cual logra integrar las capacidades de Data Warehouses y Data Lakes en un mismo sistema unificado. Sin embargo, actualmente no existe una arquitectura establecida para este sistema, como sí existen para los Data Warehouses y Data Lakes (Armbrust y cols., 2021; B. Inmon y cols., 2021).

Por otro lado, la Gestión de Calidad de Datos en sistemas de información implica un conjunto variado de tareas, como *data profiling*, definición de un modelo de calidad, implementación y ejecución de mediciones de calidad, entre otras. Debido al volumen de datos de Big Data y los distintos requerimientos de análisis impuestos por diferentes usuarios, la gestión de la calidad de datos cobra un papel sumamente relevante en estas arquitecturas y su gestión debe adaptarse a toda la variabilidad que estas presentan. En la actualidad no existen técnicas o metodologías globalmente aceptadas, que sean específicas para la gestión de calidad de datos en arquitecturas de este tipo (Ravat y Zhao, 2019a; Armbrust y cols., 2021; Zouari y cols., 2021; Nargesian y cols., 2019).

1.2. Contexto

Este proyecto de grado se desarrolla en el marco del proyecto de investigación "Calidad de Datos en la Preparación para el Análisis de Big Data", financiado por CSIC (Comisión Sectorial de Investigación Científica), del grupo de investigación GEMA (Gestión, Modelado y Análisis de Datos) del Insituto de Computación (INCO) de la Facultad de Ingeniería.

El objetivo general de este proyecto es brindar soluciones para la gestión de calidad de datos en arquitecturas de Big Data, siguiendo un enfoque basado en el contexto.

Dentro de los objetivos específicos de este proyecto se encuentra la definición de una arquitectura genérica para el análisis de Big Data que logre combinar las capacidades de Data Warehouses y Data Lakes, y la propuesta de una estrategia para la gestión de calidad de datos en la arquitectura definida.

1.3. Objetivos

Se plantean 3 objetivos principales para este proyecto, descritos a continuación.

1. Definición de una arquitectura de Big Data genérica, que logre soportar el procesamiento y almacenamiento de grandes volúmenes de datos he-

terogéneos, brindando capacidades de los sistemas de Data Warehouse y Data Lakes.

2. Incorporación de un proceso de gestión de calidad de datos dentro de la arquitectura propuesta. Para esto se debe considerar un proceso genérico de gestión de calidad de datos y, para las etapas de este proceso que correspondan, incorporarlas dentro de la arquitectura propuesta. La resolución de este problema va a requerir el manejo de metadatos, tanto relativos a calidad de datos, como a los distintos procesos involucrados en la arquitectura. Por lo tanto, un sub-objetivo de este objetivo, es la definición de un modelo de metadatos general de la arquitectura, que se enfoque especialmente en los metadatos relativos a la gestión de calidad.

3. Desarrollo de un prototipo de la arquitectura propuesta, el cual debe incorporar un proceso de gestión de calidad de datos y el modelo de metadatos definido.

1.4. Resultados esperados

Al finalizar el proyecto se espera obtener los siguientes resultados:

1. Estado del arte de arquitecturas de Big Data, considerando propuestas teóricas y aquellas disponibles en el mercado.
2. Especificación de una arquitectura de Big Data genérica que incorpore la gestión de calidad de datos.
3. Prototipo de la arquitectura propuesta que muestre su factibilidad técnica y tecnológica.

1.5. Organización del documento

Este documento se organiza en 7 capítulos incluyendo el actual.
En el siguiente capítulo se presentan descripciones de algunos términos y conceptos relevantes para una adecuada comprensión del trabajo. En el capítulo 3 se presenta el estudio del estado del arte realizado durante el transcurso del proyecto. En el capítulo 4 se presenta la arquitectura propuesta, mientras que en el capítulo 5 se presentan distintos aspectos que contribuyen al gobierno de datos dentro de la arquitectura, como el proceso de gestión de calidad de datos y el modelo de metadatos. Por último, en el capítulo 6 se presenta la implementación del prototipo de la arquitectura, mientras que en el capítulo 7 se presentan las conclusiones y el trabajo a futuro.

Capítulo 2

Marco teórico

En este capítulo se presentan descripciones de conceptos claves para una adecuada comprensión del trabajo.

2.1. Data Warehouse

Un Data Warehouse (en adelante, DW) es una colección de datos orientados a temas, integrados, históricos y no volátiles, que se utiliza para ayudar en la toma de decisiones de una organización (W. H. Inmon, 2005).

Cuando se habla de datos orientados a temas, se hace referencia a que los datos se organizan en torno a conceptos de negocio de la organización. Por otro lado, estos datos suelen obtenerse de diversas fuentes dentro de la organización, por lo que puede ser necesario integrarlos (datos integrados). A su vez, es de utilidad para la toma de decisiones, contar con datos históricos que permitan hacer proyecciones a futuro u obtener tendencias en los datos, por lo que se manejan los datos con una referencia temporal asociada (datos históricos). Por último, los datos almacenados en estos sistemas deben ser no volátiles para permitir realizar análisis sobre los mismos en un período de tiempo prolongado, donde los mismos se mantienen estables (Marotta y Serra, s.f.).

En la Figura 2.1 se presenta la arquitectura clásica de los sistemas de DW.

Estos sistemas están formados por 3 capas, la capa de procesamiento de los datos (ETL), la capa de almacenamiento (DW) y la capa Frontend (Herramientas de visualización de datos). Los procesos ETL se encargan de extraer los datos de las fuentes, transformarlos e integrarlos, y cargar los datos en el DW. Los datos almacenados en el DW se modelan siguiendo un enfoque de modelado particular conocido como modelado multidimensional (Yessad y Labiod, 2016). Por último, los datos son consultados por herramientas de visualización.

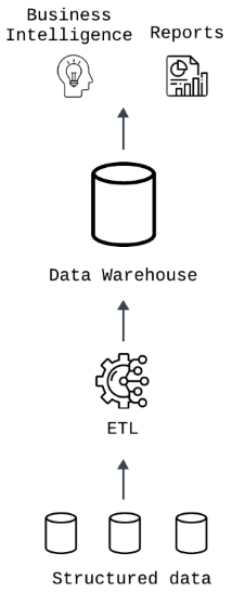

Figura 2.1: Arquitectura Data Warehouse

Una característica de estos sistemas es que siguen un enfoque *schema-on-write*; ya que previamente a la carga de los datos se debe definir el esquema que estos deben cumplir (Harby y Zulkernine, 2022).

Por último, es importante mencionar que los principales usuarios de los sistemas de DW son los *Data Analysts* (en adelante, DA). Estos usuarios buscan analizar viejos patrones de datos, utilizando herramientas de cálculo y visualización sobre datos con un alto nivel de refinamiento (B. Inmon y cols., 2021).

2.1.1. ETL y ELT

Los procesos ETL (Extraction, Transformation and Load, en adelante ETL) son procesos que se encargan de la extracción de datos de diversas fuentes, luego, realizan transformaciones sobre los mismos, y por último, se encargan de cargarlos en un sistema de almacenamiento (Marotta y Serra, s.f.).

Por otro lado, los procesos ELT (Extraction, Load and Transformation, en adelante ELT) se encargan de extraer datos de las fuentes, cargarlos a un sistema de almacenamiento, y luego de la carga, realizan transformaciones sobre los mismos (Marotta y Serra, s.f.).

2.1.2. Modelado Multidimensional

El modelado multidimensional permite estructurar los datos para realizar un análisis de los mismos según diversos criterios. Este modelo define cubos que permiten analizar datos de un dominio a través de múltiples dimensiones.

Los cubos están formados por los siguientes elementos (Ver Figura 2.2).

1. **Dimensiones:** Las dimensiones son los criterios de análisis de los datos y se corresponden con los ejes en los cubos. Las dimensiones pueden estar compuestas por diversas **Jerarquías**, las cuales definen categorías sobre los datos de una misma dimensión, a partir de niveles (Marotta y Serra, s.f.).
2. **Medidas:** Las medidas son los valores a analizar y se obtienen a partir de la intersección de valores de cada una de las dimensiones (Marotta y Serra, s.f.).

La realidad se modela con un conjunto de cubos.

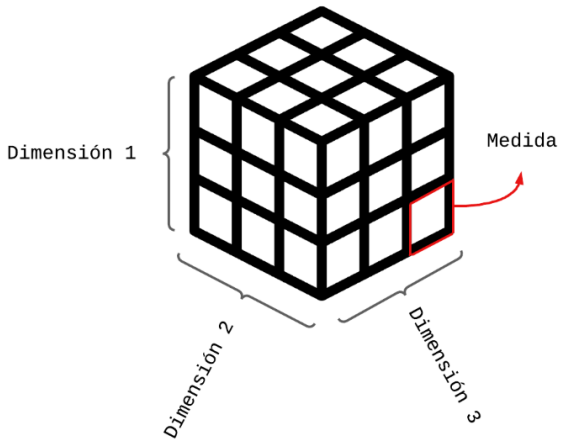

Figura 2.2: Diagrama de cubo.

Por otro lado, existen diversas operaciones que permiten manipular los cubos, en particular, las operaciones Drill-Up y Drill-Down que permiten moverse en los niveles de una jerarquía de una dimensión, y obtener distintas versiones de una misma medida (Marotta y Serra, s.f.). Estas distintas versiones de una misma medida se conocen como "Agregaciones".

Modelo CMDM

El Modelo CMDM permite realizar el diseño conceptual de un modelo multidimensional, independientemente de las tecnologías que se van a utilizar para

desarrollarlo (Marotta y Serra, s.f.).

A continuación se desciben las estructuras básicas de este modelo (Marotta y Serra, s.f.).

1. **Dimensiones:** Una dimensión posee un nombre y una jerarquía con niveles.

2. **Jerarquías:** Las jerarquías están compuestas por uno o varios niveles, donde se tiene una relación 1 a N entre elementos del nivel superior al inferior.

3. **Niveles:** Representan un conjunto de datos que puede no ser atómico.

4. **Relaciones dimensionales:** Representan un cruzamiento entre un conjunto de dimensiones. En el diagrama se representan las dimensiones a cruzar y se asocian las medidas correspondientes para ese cruzamiento.

5. **Cubos:** Un cubo representa un cruzamiento particular de determinados niveles de jerarquías, de las dimensiones de una Relación Dimensional.

En la Figura 2.3 se presenta un ejemplo de dimensión con su respectiva jerarquía, compuesta por 3 niveles, Continente, País y Ciudad.

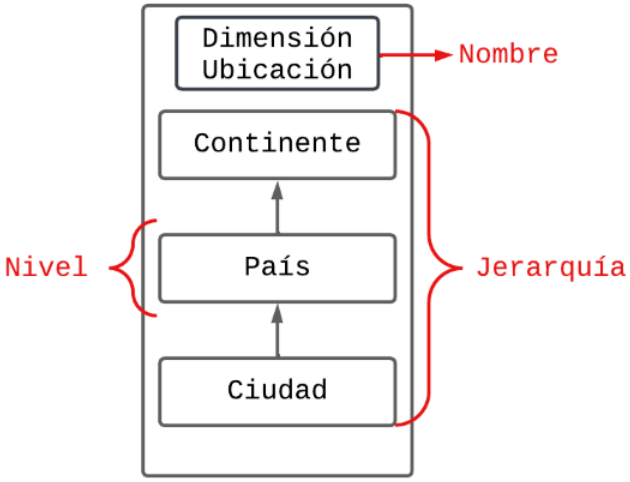

Figura 2.3: Dimensión Ubicación.

A su vez, en la Figura 2.4 se ve un ejemplo de cómo se relacionan los valores de los diferentes niveles de la jerarquía, donde se puede ver una relación 1 a N desde los niveles superiores a los inferiores.

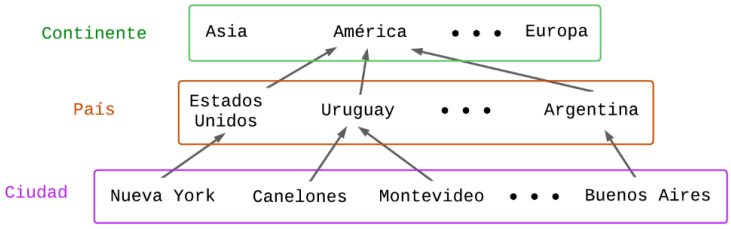

Figura 2.4: Jerarquía de la dimensión Ubicación.

Por último, en la Figura 2.5 se puede ver un ejemplo de relación dimensional, donde se cruzan las dimensiones "Ubicación", "Productos" y "Vendedores", y se obtiene una medida "cant_unidades_vendidas", que refleja la cantidad de unidades vendidas para un producto, un vendedor y una ubicación particular.

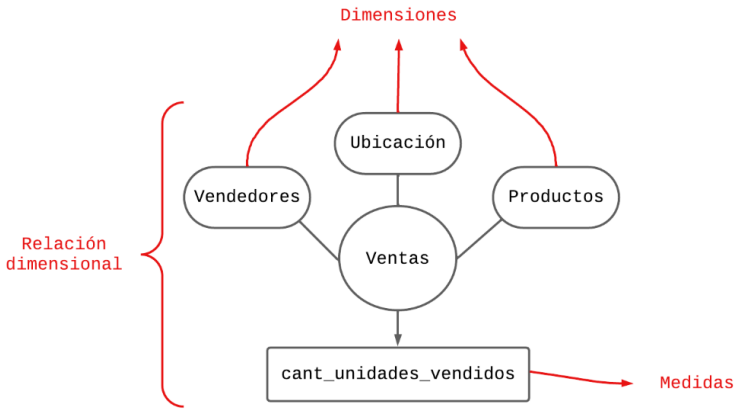

Figura 2.5: Relación dimensional.

2.1.3. Sistemas de Procesamiento Analítico en Línea

Los sistemas de procesamiento analítico en línea (Online Analytical Processing, en adelante OLAP) son un conjunto de herramientas de frontend, que permiten analizar datos desde distintas perspectivas (Marotta y Serra, s.f.). Para poder realizar este tipo de análisis, estos sistemas almacenan los datos siguiendo un modelo multidimensional. En particular, estas herramientas permiten navegar los cubos del modelo multidimensional, permitiendo visualizar métricas desde distintas perspectivas.

Los sistemas OLAP se clasifican según el sistema de almacenamiento utilizado, como se describe a continuación (Marotta y Serra, s.f.).

1. **Multidimensional Online Analytical Processing (MOLAP):** Se almacenan los datos en cubos multidimensionales y se utiliza el leguaje MDX para consultar los datos en los cubos. Este enfoque utiliza estructuras especiales (*proprietary formats*) para almacenar los datos y presenta buena performance en consultas dimensionales.

2. **Relational Online Analytical Processing (ROLAP):** Se almacenan los datos en bases de datos relacionales y se utiliza el lenguaje SQL para realizar operaciones OLAP (operaciones de manipulación de cubos) sobre los datos. Este enfoque puede almacenar grandes volúmenes de datos y es más flexible que el enfoque MOLAP a la hora de escalar el sistema, pero puede tener menor performance en las consultas dimensionales, en comparación con el enfoque MOLAP.

3. **Hybrid Online Analytical Processing (HOLAP):** Combina los enfoques MOLAP y ROLAP, almacenando datos en cubos multidimensionales (performance en consultas) y en bases de datos relacionales (escalabilidad del sistema).

En el enfoque ROLAP el modelado de los datos es sumamente importante; ya que para realizar el análisis de los mismos se accede directamente a la base de datos. Existen diversos esquemas de modelado de datos para aumentar la performance de las consultas dimensionales realizadas sobre las bases de datos relacionales. A continuación se describen 2 de estos esquemas.

El primer enfoque de modelado es el "Esquema estrella" (Ver Figura 2.6). Este esquema está compuesto por dos tipos de tablas, la tabla de hechos (*fact table*) y las tablas dimensionales (*dimension table*). La tabla de hechos almacena las medidas numéricas del negocio y las tablas dimensionales almacenan la información de las dimensiones, incluyendo la información de las jerarquías, las cuales se encuentran desnormalizadas dentro de estas tablas. Luego, la tabla de hechos se asocia a las distintas dimensiones a través de relaciones 1 a N (Marotta y Serra, s.f.).

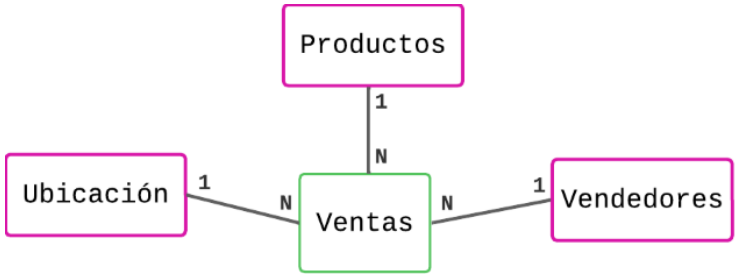

Figura 2.6: Esquema estrella.

El segundo enfoque de modelado es el "Esquema snowflake" (Ver Figura 2.7). Este enfoque utiliza los mismos tipos de tabla (*dimension* y *fact table*),

pero las tablas dimensionales se descomponen en varias tablas que forman las jerarquías. Por lo tanto, este enfoque se puede pensar como un "Esquema estrella" con las dimensiones normalizadas según las jerarquías, generando una tabla para cada nivel de las mismas (Marotta y Serra, s.f.).

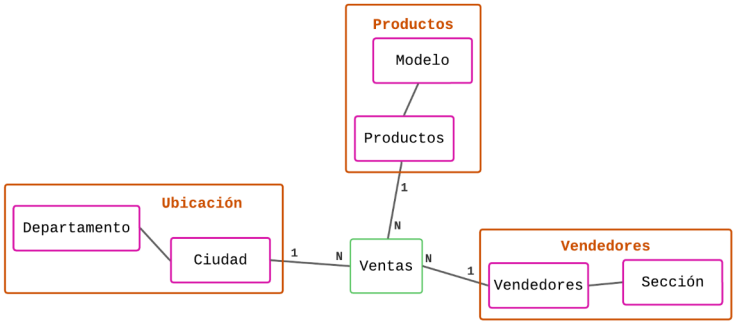

Figura 2.7: Esquema Snowflake.

Existen otros tipos de esquemas de modelado como el esquema *Data Vault* (Linstedt y Olschimke, 2015) y el esquema *Star Cluster*(Marotta y Serra, s.f.).

2.1.4. Business Intelligence

Se llama Business Intelligence (en adelante, BI) al proceso de análisis de datos orientado al negocio, utilizado para ayudar en la toma de decisiones de una organización. En la mayoría de los casos, se utilizan herramientas OLAP (Marotta y Serra, s.f.).

2.2. Big Data

No existe una definición formal adoptada para el término "Big Data" (en adelante, BD), sino que se lo define a partir de las características de los datos que abarca. Estas características se conocen como las V's de BD, las cuales a lo largo de los años han ido aumentando, llegando a más de 50 (Elouataoui, El Alaoui, el Mendili, y Youssef, 2022). Sin embargo, existen 4 V's que son las más utilizadas, las cuales se presentan a continuación (Marotta y Serra, s.f.).

1. Volumen: hace referencia a los grandes volúmenes de datos que se manejan en BD.
2. Variedad: hace referencia a la heterogeneidad y diversidad de los datos.
3. Velocidad: hace referencia a la velocidad con la cual se producen los datos.
4. Veracidad: hace referencia a la calidad, integridad y credibilidad de los datos.

2.2.1. Analítica Avanzada

La Analítica Avanzada (en adelante, AA) hace referencia a un conjunto de técnicas y herramientas que permiten realizar análisis más sofisticados sobre los datos, que los que se pueden realizar con herramientas de BI. Algunos ejemplos de estas técnicas son Minería de Datos, Aprendizaje Automático (*Machine Learning*, en adelante, ML), Procesamiento de Lenguaje Natural, Análisis Estadísticos, entre otros (*Definition of Advanced Analytics - Gartner Information Technology Glossary*, s.f.).

2.2.2. Data Science

La Ciencia de Datos (*Data Science*, en adelante, DS) es una disciplina que combina técnicas de distintos campos como la Matemática, Estadística, Analítica Avanzada, entre otros, para derivar conocimiento a partir de los datos de una organización. El conocimiento obtenido es utilizado para la toma de decisiones y planificación estratégica (*What is Data Science? | IBM*, 2021).

Quienes llevan a cabo esta disciplina se denominan Data Scientists (en adelante, DST).

2.2.3. Clasificación de datos

Los datos se pueden clasificar según distintos criterios. A continuación se presentan dos formas de clasificarlos, pero existen otras.

La primera clasificación agrupa los datos según las características de su estructura en 3 tipos, como se describe a continuación.

1. **Datos estructurados:** Datos que presentan un esquema bien definido, a partir del cual se puede interpretar la semántica de los mismos (Batini y Scannapieco, 2016).
2. **Datos semi-estructurados:** Datos que son parcialmente estructurados o que presentan un esquema descriptivo (Batini y Scannapieco, 2016).
3. **Datos no estructurados:** Cualquier secuencia de símbolos codificada en algún lenguaje, sin un esquema definido que permita inferir su semántica (Batini y Scannapieco, 2016).

La segunda clasificación agrupa los datos según características y propiedades similares. A continuación se describen algunas categorías descritas en (*Executing Data Quality Projects*, 2021).

1. **Datos maestros:** Los datos maestros (*master data*) describen personas, lugares y objetos del negocio de una organización. Esta categoría de datos suele agruparse en *master records* los cuales pueden hacer referencia a datos de referencia (*reference data*).

2. **Datos transaccionales:** Los datos transaccionales (*transactional data*) describen eventos internos o externos que ocurren mientras una organización lleva a cabo su negocio. Esta categoría se agrupa en *transactional records* los cuales pueden hacer referencia a datos maestros y referenciales.

3. **Datos de referencia:** Los datos de referencia (*reference data*) son conjuntos de valores utilizados por procesos y sistemas, así como *master* y *transactional records*. Algunos ejemplos de datos de referencia son códigos de países, campos demográficos, tipos de productos, entre otros. Los datos de referencia pueden ser definidos por una organización particular o pueden ser definidos como estándares por organizaciones globales.

4. **Metadatos:** Los metadatos (*metadata*) son "datos sobre datos". Estos describen datos para facilitar el acceso, interpretación y uso de los mismos.

5. **Datos históricos:** Los datos históricos (*historical data*) son conjuntos de datos recopilados sobre eventos pasados, los cuales tienen asociado una marca de tiempo.

6. **Datos temporales:** Los datos temporales (*temporary data*) son datos que se almacenan en memoria (almacenamiento temporal) para acelerar procesamientos.

2.2.4. Procesamiento de Stream y Batch

En el contexto de BD, existen dos grandes tipos de procesamiento para los datos: el procesamiento batch y el procesamiento de stream.

Procesamiento Batch

El procesamiento batch se realiza sobre grandes conjuntos de datos, los cuales se subdividen en particiones más pequeñas (*batch*). Luego, cada partición se procesa de manera individual, cada cierto período de tiempo. En general, este tipo de procesamiento se realiza en simultáneo sobre las distintas particiones.

La principal ventaja de este procesamiento es la posibilidad de ejecutar procesos sobre grandes conjuntos de datos, en partes más pequeñas, para garantizar la eficiencia de los mismos (Benjelloun y cols., 2020).

Este tipo de procesamiento es útil cuando es más importante el procesameinto de grandes volúmenes de datos, en vez de la velocidad de procesamiento.

Procesamiento de Stream

El procesamiento de stream se utiliza cuando es necesario procesar los datos a medida que son recibidos por el sistema.

Este es continuo a medida que los datos llegan al sistema, lo que puede ocasionar que se acumulen datos esperando a ser procesados.

No se debe confundir el procesamiento de stream con el procesamiento de datos en tiempo real; ya que en este último el procesamiento y los resultados deben ser inmediatos, mientras que en el procesamiento de stream, el procesamiento y los resultados son continuos, pero no inmediatos (Benjelloun y cols., 2020).

El procesamiento de stream se utiliza cuando es importante obtener resultados lo antes posible. A su vez, los datos adecuados para este tipo de procesamiento se caracterizan por la velocidad con la cual se obtienen, y no tanto por el volumen de los mismos (Benjelloun y cols., 2020).

2.3. Data Lake

Un Data Lake (en adelante, DL) es un sistema de almacenamiento que permite "ingerir"(*ingest*) datos crudos (*raw data*) de diversas fuentes, en su formato original (Hai y cols., 2021). A su vez, tiene la capacidad de almacenar grandes volúmenes de datos de distintos tipos (estructurados, semi-estructurados y no-estructurados), en sistemas de almacenamiento de bajo costo (Giebler y cols., 2019). Por otra parte, debe permitir realizar consultas y análisis sobre los datos, de manera *on-the-fly*.

El término *on-the-fly* u *on-demand* hace referencia a que la definición de esquemas, integración de datos o indexado de datos se debería realizar al momento de acceder a los mismos (Hai y cols., 2021).

Por último, es importante mencionar que los principales usuarios de estos sistemas son los DST. Estos usuarios utilizan los datos para encontrar patrones y tendencias, que puedan ser de utilidad para la organización (B. Inmon y cols., 2021).

2.4. Gobierno de Datos

"La Gobernanza de Datos es todo lo que se realiza para garantizar que los datos sean seguros, privados, precisos, disponibles y utilizables. Incluye las acciones que las personas deben realizar, los procesos que deben seguir y la tecnología que los respalda durante todo el ciclo de vida de los datos."(*What is Data Governance?*, s.f.)

A su vez, los autores de (B. Inmon y cols., 2021) indican que los objetivos del gobierno de datos pueden variar según la organización, pero existen algunos componentes comunes que se describen a continuación.

1. La definición de un modelo de Gobierno de Datos, donde se establezcan roles y responsabilidades, relacionados a los procesos de los datos.

2. Principios, políticas y procedimientos para guiar comportamientos y decisiones con respecto a los datos.

3. Calidad de datos.

4. Gestión de metadatos.

5. Gestión del ciclo de vida de los datos, para asegurarse que todas las etapas soportan el uso de datos para obtener valor empresarial.

2.5. Calidad de Datos

No hay una definición concreta de "Calidad de Datos" (en adelante, CD), pero se puede partir de la definición de "Calidad". La "Calidad" presenta diversas definiciones como "conjunto de características de un producto que influyen en su capacidad para satisfacer necesidades declaradas o implícitas", "Adecuación para su uso (*fitness for use*)" y "Satisfacción del usuario" (Batini y Scannapieco, 2016).

En general, se tiene una concepción de que la calidad de los datos está asociada a la "exactitud" de los mismos. Sin embargo, existen otras dimensiones como Completitud, Consistencia, entre otras, que permiten caracterizar otros aspectos de la calidad de los datos (Batini y Scannapieco, 2016).

Existe una jerarquía entre los conceptos de calidad, presentada en la Figura 2.8. Las **Dimensiones** capturan aspectos específicos de calidad a alto nivel. Luego, cada dimensión se puede subdividir en un conjunto de **Factores** que se enfocan en características particulares, los cuales pueden medirse con diversas **Métricas**. Las métricas definen cómo se mide un factor de calidad y quedan definidas por un nombre, una descripción, las unidades de medición y la granularidad de la medida (dependiente del modelo de datos). Por último, cada métrica se puede implementar con varios **Métodos de medición**, quienes se encargan de tomar las medidas, correspondientes a una métrica, sobre un dato particular.

A continuación se describen algunas dimensiones de calidad consideradas para el desarrollo de este proyecto (Batini y Scannapieco, 2016) (Etcheverry, Marotta, Sanz, y Serra, s.f.). Notar que estas dimensiones permiten medir la calidad sobre conjuntos de datos estructurados o semi-estructurados.

1. **Exactitud:** Mide la cercanía entre un valor v y un valor v' (donde v' es el valor correcto en la realidad, que v busca representar).

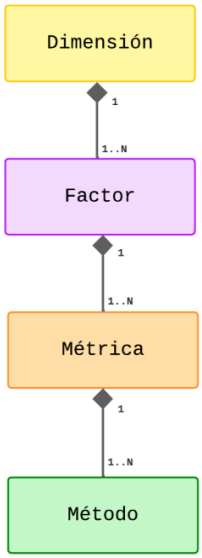

Figura 2.8: Jerarquía de conceptos de calidad.

   a) **Exactitud Sintáctica:** Busca medir si los valores en el sistema se corresponden con valores válidos del dominio, sin importar sin son los valores reales. Esto es, mide la cercanía de un valor v con los elementos del dominio D correspondiente.
   b) **Exactitud Semántica:** Busca medir qué tan bien se representan los estados del mundo real en el sistema, esto es, mide la cercanía de un valor v con el verdadero valor v'.
   c) **Precisión:** Busca medir la precisión de los datos, esto es, qué tan detallados son los datos.

2. **Completitud:** Busca medir si el sistema contiene toda la información de interés.
   a) **Densidad:** Mide la cantidad de información existente en el sistema y cuánta información falta.
   b) **Cobertura:** Mide la porción de los datos de la realidad contenidos en el sistema.

3. **Consistencia:** Busca medir la satisfacción de reglas semánticas definidas sobre los datos.
   a) **Integridad de dominio:** Mide la satisfacción de reglas sobre el contenido de un atributo.
   b) **Integridad intra-relación:** Mide la satisfacción de reglas entre atributos de un mismo dataset.

c) **Integridad inter-relación:** Mide la satisfacción de reglas entre atributos de distintos datasets.

4. **Unicidad:** Mide el nivel de duplicación de datos en el sistema.

   a) **No-duplicación:** Hay duplicación en los datos si la misma entidad aparece repetida de forma exacta. Luego, este factor busca medir el grado de datos "no duplicados".

   b) **No-contradicción:** Hay contradicción si la misma entidad aparece repetida con contradicciones. Luego, este factor busca medir el grado de datos "no contradictorios".

Por último, en la Figura 2.9 se presenta el proceso de Gestión de Calidad de Datos (en adelante, GCD) para Sistemas de Información adoptado para el desarrollo de este trabajo (Etcheverry y cols., s.f.).

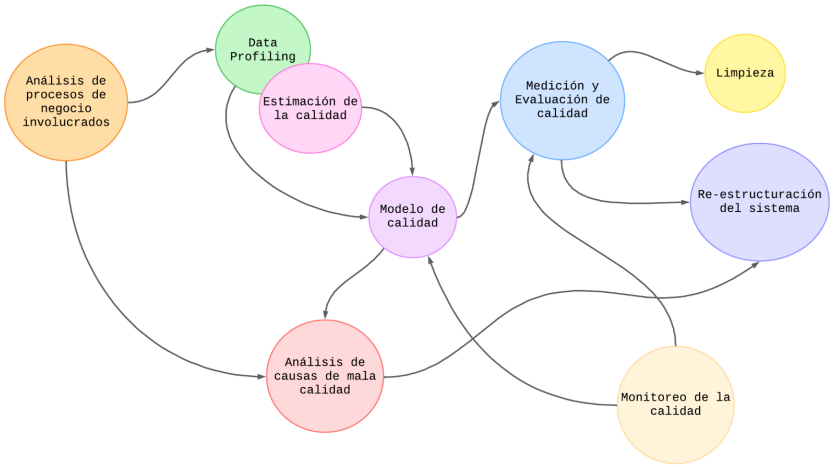

Figura 2.9: Proceso de Gestión de Calidad de Datos para Sistemas de Información

A continuación se describen las distintas etapas de este proceso (Etcheverry y cols., s.f.).

1. **Análisis de procesos de negocio involucrados:** Se realiza un análisis preliminar de los procesos de negocio que utilizan el sistema. Esto permite entender el negocio y tener un primer acercamiento a los datos.

2. **Data Profiling y Estimación de Calidad:** Se realiza un primer relevamiento del estado de los datos del sistema, para identificar su estructura, relaciones, volumen, problemas y frecuencia de ocurrencia de los mismos, y patrones en los datos. A su vez, algunas técnicas o herramientas de Data

Profiling permiten obtener medidas sobre la calidad de los datos (Por ejemplo, cantidad de datos *NULL*, cantidad de datos en blanco, entre otros), permitiendo hacer una primera estimación de la calidad de los mismos. Esto puede ser útil en casos donde se busque estimar costos de corrección de los datos, re-estructuración del sistema, limpieza, entre otros aspectos.

3. **Modelo de Calidad:** Define qué características de calidad se utilizan, sobre qué datos aplican y cómo se miden esas características.

4. **Medición y Evaluación de calidad:** Se mide la calidad para obtener información de la calidad actual de los datos y para analizar el costo de mejorar la calidad. Para medir se debe definir previamente el Modelo de Calidad, los Métodos de Medición y los Metadatos de Calidad (el modelo de datos que va a almacenar las medidas de calidad obtenidas). Por otro lado, para realizar la Evaluación se debe contar con requerimientos de usuarios que establecen la calidad necesaria que deben poseer los datos. Durante la etapa de Evaluación se busca determinar si la calidad de los datos es correcta o no, con respecto a los requerimientos de los usuarios.

5. **Análisis de causas de mala calidad:** Se analiza cuáles son las causas que generan problemas de calidad en los datos (Por ejemplo, transformaciones que insertan problemas de calidad en los datos, datos con errores provenientes de la fuente, entre otros).

6. **Limpieza de datos:** Se identifican y eliminan inconsistencias, discrepancias y errores en los datos para mejorar la calidad.

7. **Re-estructuración del sistema:** Se realizan modificaciones en el sistema para incluir procesos de limpieza de los datos o modificar procesos que generen errores de calidad en los datos.

8. **Monitoreo de la calidad:** Se monitorea la calidad a medida que el sistema obtiene nuevos datos, pudiendo generar cambios en el Modelo de Calidad y generar nuevas mediciones y evaluaciones de calidad.

Capítulo 3

Trabajo relacionado

En este capítulo se presenta el estudio del estado del arte realizado durante el desarollo de este trabajo.

3.1. Arquitecturas de Big Data

El primer acercamiento a un sistema de BD fueron los Sistemas de DW, los cuales se encargan de recopilar datos de toda una organización y almacenarlos de forma centralizada en un DW. Estos datos, luego, son accedidos por herramientas de BI, para ayudar en la toma de decisiones de la organización.

Debido a la diversidad de datos existentes hoy en día y el procesamiento requerido de los mismos, estos sistemas dejaron de ser eficientes para el almacenamiento de datos de BD. A continuación se mencionan las desventajas de estos sistemas.

1. No logran almacenar datos no estructurados como audio, video, imágenes, entre otros. (Harby y Zulkernine, 2022)

2. No es posible realizar análisis de datos en tiempo real, debido a la demora de los procesos ETL. (Harby y Zulkernine, 2022)

3. No brindan soporte a herramientas de AA (Armbrust y cols., 2021)

4. En general utilizan formatos propietarios, lo que dificulta posibles migraciones de datos hacia otros sistemas (B. Inmon y cols., 2021).

Para remediar algunas de estas desventajas es que surge un nuevo sistema conocido como DL (Figura 3.1). Los DL utilizan sistemas de almacenamiento de bajo costo, permitiendo almacenar datos en diversos formatos, siguiendo un enfoque *schema-on-read*. Luego, el esquema de los datos se define a la hora de la lectura de los datos, en caso de ser necesario, pero no es un requisito para

almacenarlos en el sistema (Hai, Koutras, Quix, y Jarke, 2023).

Estos sistemas eliminan las desventajas de los DW, permitiendo almacenar datos en cualquier formato de forma rápida (No se tiene la demora de los procesos ETL), utilizando sistemas de almacenamiento de bajo costo, y brindando soporte a herramientas de AA. Por otro lado, a diferencia de los DW, los DL utilizan procesos ELT.

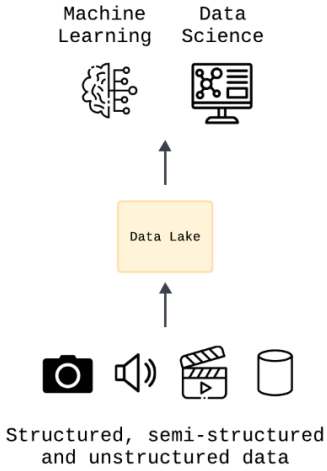

Figura 3.1: Arquitectura Data Lake

La primera arquitectura de DL tenía la gran desventaja de convertirse en un pantano de datos (*data swamp*), debido al almacenamiento constante de datos entrantes al sistema, sin ningún tipo de procesamiento sobre los datos almacenados y sin una correcta gestión de metadatos (Harby y Zulkernine, 2022). Es por esto que surgieron diversas arquitecturas para gestionar los datos dentro de un DL, realizando distintos procesamientos para disminuir el riesgo de que este se convierta en un data swamp. Por lo tanto, por más que estos sistemas logran mejorar las desventajas de los DW, también presentan los desafíos de la integración de datos heterogéneos, y la gestión de metadatos.

Actualmente, la comunidad científica investiga un nuevo sistema que busca integrar las capacidades de ambos sistemas (DW y DL), conocido como Data Lakehouse (en adelante, DLH). Este sistema debe poder dar soporte a herramientas de AA, así como de BI, mientras almacena diversos tipos de datos, garantizando buena performance en el acceso a los mismos. Sin embargo, no existe una definición concreta y estándar de este sistema, ni una arquitectura de referencia. Por otro lado, en el mercado se pueden encontrar sistemas de diversos proveedores denominados “Data Lakehouse”, los cuales plantean que estas plataformas presentan todas las características de DL y DW en un sistema

unificado (Armbrust y cols., 2021; B. Inmon y cols., 2021).

3.2. Arquitecturas de Data Lake

En las siguientes secciones se presentan los resultados del estado del arte realizado para las arquitecturas de DL. En particular, se analizaron los trabajos (Hlupić, Oreščanin, Ružak, y Baranović, 2022), (B. Inmon, 2016), (Quix y Hai, 2019), (Giebler, Groger, Hoos, Schwarz, y Mitschang, 2020), (Zhao, Megdiche, Ravat, y Dang, 2021), (Oukhouya, El Haddadi, Er-raha, y Asri, 2021), (Herden, 2020) y (Hai y cols., 2021).

Se pueden encontrar 3 tipos de arquitecturas de DL: Arquitecturas en capas, Arquitecturas en Zonas y la Arquitectura de *Data Pond*. A su vez, existen distintos ejemplos de arquitecturas basadas en zonas, las cuales difieren en la cantidad de zonas dentro del DL.

3.2.1. Arquitecturas en capas

La primera arquitectura de DL definida, consistía de dos capas (Figura 3.2), la Landing Zone y la Raw Zone (Hlupić y cols., 2022).

La **Landing Zone** se encarga de la extracción de los datos de las fuentes y del almacenamiento temporal de los mismos. Por otro lado, la **Raw Zone** es donde se almacenan los datos en forma permanente dentro del DL (Hlupić y cols., 2022).

La principal desventaja de esta arquitectura es que tiene el riesgo de convertirse en un *data swamp*; ya que solo se almacenan los datos, no se realiza procesamiento sobre los mismos y tampoco hay una gestión de metadatos. Luego, el DL se convierte en un gran repositorio de datos, que crece en volumen y se pierde conocimiento de lo que se tiene almacenado.

A partir de esta arquitectura surgen otras, donde se incorporan nuevas capas para incluir procesamiento de datos.

Los autores de (Quix y Hai, 2019) presentan una arquitectura que se divide en cuatro capas, cada una con responsabilidades bien definidas, definiendo un pipeline que deben atravesar los datos para poder ser accedidos por los usuarios del sistema (Figura 3.3). A continuación se describen las distintas capas que componen al sistema.

1. **Ingestion layer:** Se encarga de "ingerir" los datos de las distintas fuentes, y en caso de ser posible, también se encarga de la extracción de metadatos de las fuentes (Hlupić y cols., 2022). A su vez, esta capa cuenta con el componente DQ Control, que se encarga de controlar que los datos ingeridos tengan cierto nivel de calidad (Quix y Hai, 2019).

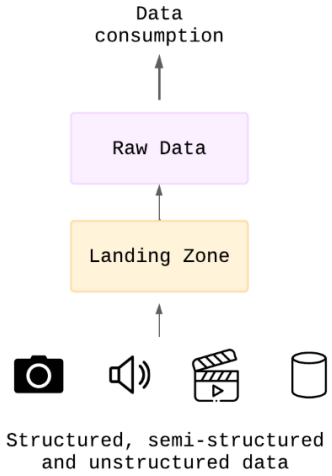

Figura 3.2: Two Layered Architecture

2. **Storage Layer**: Capa donde se almacena el repositorio de datos crudos (*raw data repository*) y el de metadatos (*metadata repository*) (Hlupić y cols., 2022). El repositorio de metadatos almacena los metadatos que se extrajeron en la **Ingestion Layer** y metadatos que se van a generar a partir de los procesos llevados a cabo sobre los datos en otras capas (Quix y Hai, 2019).

3. **Transformation Layer**: Permite realizar operaciones de *data cleaning*, *data integration* y *data transformation* sobre datos extraídos de la **Storage Layer**. En esta capa se crean *data marts* específicos para los distintos casos de uso del sistema (Quix y Hai, 2019).

4. **Interaction layer**: Este componente permite la interacción del usuario con los datos. El usuario accede a los metadatos para conocer qué datos están disponibles en el DL, para luego poder consultarlos. Esta capa debe soportar visualización y filtrado de los datos (Quix y Hai, 2019).

La principal desventaja de esta arquitectura es que los datos solo se acceden desde la Interaction Layer, donde ya están modelados para satisfacer determinados casos de uso. Sin embargo, para algunos usuarios, en particular, los DST, puede ser útil acceder a datos crudos o con menor nivel de procesamiento para poder realizar análisis sobres los mismos, sin interferir con datos que ya están modelados para casos de uso específicos.

Para finalizar, se analizó la arquitectura presentada en (Hai y cols., 2021) y (Hai y cols., 2023). Esta arquitectura está dividida en 3 capas, según los procesos llevados a cabo luego de la ingestión de los datos, y un componente de almacenamiento (Hai y cols., 2021) (Ver Figura 3.4).

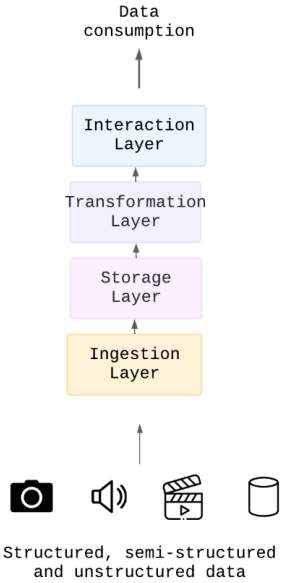

Figura 3.3: Multi-Layered Architecture propuesta en (Quix y Hai, 2019)

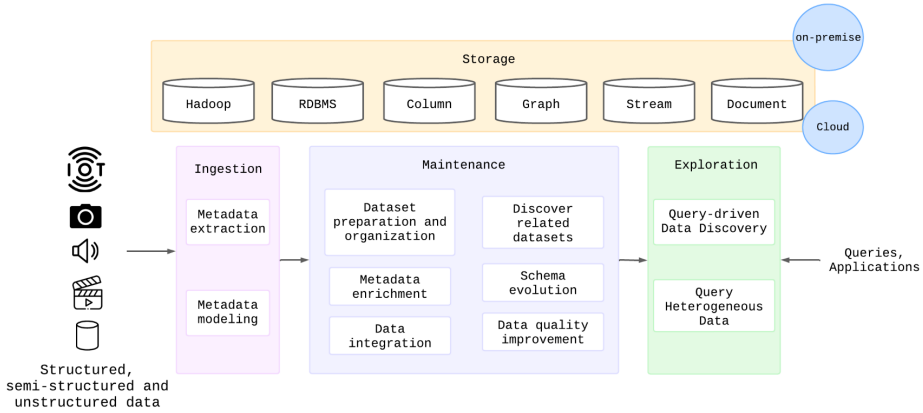

Figura 3.4: Arquitectura basada en capas propuesta en (Hai y cols., 2023) y (Hai y cols., 2021).

A continuación se describen los distintos componentes.

1. **Ingestion:** Se encarga de extraer datos de diversas fuentes, así como de la extracción y modelado de metadatos.

2. **Maintenance:** Se encarga de distintos procesos que permiten preparar y organizar los datos para ser consultados más adelante. Observar que dentro de esta zona se tiene un componente denominado *Data quality*

improvement.

3. **Exploration:** Permite acceder a los datos dentro del DL de dos formas: *query-driven data discovery* y *heterogeneous data querying*. La primera forma de explorar los datos, permite obtener los k datasets más relacionados al dataset pasado como parámetro. La segunda forma de acceder a los datos dentro del DL es a través de una interfaz unificada, que permite realizar consultas sobre datos heterogéneos estructurados. A su vez, la última capa puede ser accedida por herramientas de visualización y AA.

4. **Storage:** Componente de almacenamiento que tiene la característica de almacenar datos en distintos formatos, mediante diversas tecnologías.

Creemos que esta arquitectura tiene muchos aspectos a destacar como la extracción, modelado y almacenamiento de metadatos, el componente de mejora de la calidad de los datos y las distintas formas de consultar datos. Sin embargo, no contiene una zona donde los DST puedan acceder a cualquier dato del sistema para realizar análisis sobre los mismos, sino que deben utilizar la misma interfaz que los usuarios finales.

3.2.2. Data Pond Architecture

Esta arquitectura se divide en 5 *ponds* (estanques), según la estructura de los datos y el uso de los mismos (Figura 3.5). A continuación se describe cada uno de estos componentes .

1. **Raw Data Pond**: Componente que se encarga de la extración de los datos de las fuentes y el almacenamiento temporal de los mismos. Los datos almacenados en este componente son eliminados una vez que estos son enviados a otro *pond* para ser procesados (B. Inmon, 2016).

2. **Application Data Pond**: Es un DW donde se almacenan datos del Raw Data Pond, luego de ser integrados y haber atravesado diversos procesos ETL. Este *pond* puede ser accedido por aplicaciones externas al sistema (B. Inmon, 2016).

3. **Analog Data Pond**: Almacena datos semi-estructurados, generados a altas velocidades (Hlupić y cols., 2022). Sobre estos datos se aplican técnicas de reducción (*data reduction*), que buscan reducir el espacio de almacenamiento (B. Inmon, 2016).

4. **Textual data pond**: Almacena textos no estructurados y lleva a cabo procesos de desambiguación de textos, para poder asociar el contexto a los mismos (B. Inmon, 2016).

5. **Archival Data Pond**: Almacena datos que no han sido utilizados recientemente, pero que podrían ser reutilizados en futuros análisis. Este componente recibe datos del Application Data Pond, Analog Data Pond y Textual Data Pond (B. Inmon, 2016).

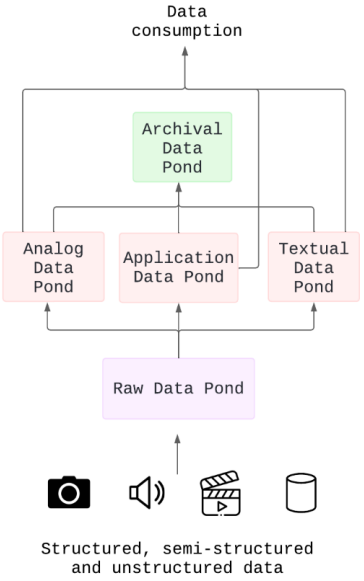

Figura 3.5: Data Pond Architecture propuesta en (B. Inmon, 2016)

Las principal desventaja de esta arquitectura es que no hay almacenamiento de los datos crudos (*raw data*), luego de ser ingeridos de la fuente, por lo que se elimina la posibilidad de reprocesamiento (Hlupić y cols., 2022). En particular, si se modifica la lógica de procesamiento de los datos, estos cambios no se van a ver reflejados en los demás componentes; ya que los datos crudos a partir de los cuales surgen los datos en los *ponds* nunca fueron almacenados.

Otra desventaja de esta arquitectura es que los datos se encuentran divididos en distintos data ponds, por lo que para ser consultados de forma unificada se debería incorporar un nuevo componente al sistema, que funcione como una interfaz unificada para acceder a los datos (Hlupić y cols., 2022). Por último, esta arquitectura no cuenta con un espacio para que usuarios como los DST puedan explorar los datos, sin interferir con las transformaciones de los mismos asociadas a casos de uso particulares.

3.2.3. Arquitecturas en zonas

La arquitectura más utilizada en la academia es la arquitectura basada en zonas. Esta arquitectura divide el DL en zonas según los distintos procesamientos que se llevan a cabo sobre los datos. Existen distintas propuestas para este tipo de arquitectura que difieren en la cantidad de zonas dentro del DL. En particular, se analizaron los trabajos de (Giebler y cols., 2020), (Sharma, 2018), (Zhao y cols., 2021), (Oukhouya y cols., 2021) y (Herden, 2020). A su vez, estos trabajos se agruparon según si la arquitectura cuenta con gobierno de datos, o

no.

Arquitecturas sin Gobierno de datos

El autor de (Sharma, 2018) presenta la arquitectura *Zaloni Architecture*, la cual modela al DL en 5 zonas distintas (Figura 3.6). A continuación se describen las distintas zonas que la conforman.

1. **Transient Landing Zone:** Es la zona encargada de "ingerir" (*data ingestion*) datos de las fuentes y almacenarlos de forma temporal, hasta que sean procesados. Se realizan análisis de *data compliance* (Se verifica que los datos cumplan con ciertos estándares o reglas establecidos por distintos organismos, con respecto a la seguridad y privacidad de los datos) y de *data quality* (Calidad de los datos).

2. **Raw Zone:** Almacena los datos crudos, que no fueron descartados por los procesos de la **Transient Landing Zone**, de forma persistente. Esta zona puede ser accedida por DST y DA para descubrir conjuntos de datos.

3. **Trusted Zone:** Extrae datos de la **Raw Zone** y realiza transformaciones (*data cleaning* y *data validation*) de los datos con un propósito de negocio específico. En esta zona se pueden almacenar datos maestros (master data) y datos de referencia (reference data).

4. **Refined zone:** Se modelan los datos según necesidades de negocio específicas. Además, los datos se integran siguiendo un formato común.

5. **Sandbox:** Funciona como un área de prueba, donde se puede acceder a datos de las demás zonas. Esta zona permite que los DST puedan manipular los datos, pudiendo enviar resultados de sus análisis hacia la Raw Zone, permitiendo que estos funcionen como nuevos datos de entrada.

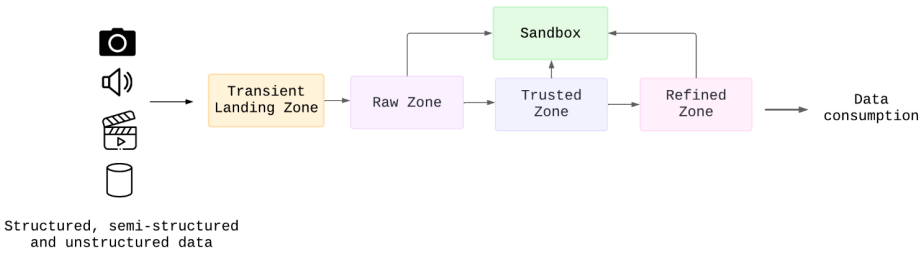

Figura 3.6: Zaloni Zone Architecture propuesta en (Sharma, 2018)

Notar que ninguna de las arquitecturas descritas hasta el momento permite modelar los datos para poder llevar a cabo un análisis OLAP de los mismos.

A continuación se presenta la arquitectura propuesta en (Giebler y cols., 2020), la cual se basa en la arquitectura de (Sharma, 2018). Esta arquitectura incorpora dos nuevas zonas (Figura 3.7), para permitir el modelado de datos para ser consultados por herramientas de BI. A continuación se definen las distintas zonas que componen esta arquitectura.

1. **Landing Zone**: Almacena los datos crudos de forma transitoria hasta que son procesados y enviados a la **Raw Zone** (Giebler y cols., 2020). Esta zona es análoga a la **Transient Landing Zone** de Zaloni.

2. **Raw Zone**: Almacena los datos crudos extraídos de las fuentes (Giebler y cols., 2020). A su vez, los datos en esta zona están historizados (*historized*), esto es, se puede hacer un seguimiento, en el tiempo, de los cambios en los datos. Esta zona es análoga a la **Raw Zone** de Zaloni.

3. **Harmonized Zone**: Obtiene datos de la **Raw Zone**, "a demanda". Esta zona se encarga de integrar los datos en un esquema consolidado, siguiendo un mismo enfoque de modelado. De esta forma, se logra obtener una visión unificada de los datos, permitiendo acceder a estos a través de una única interfaz (Giebler y cols., 2020). Esta zona es muy similar a la **Trusted Zone** de Zaloni.

4. **Distilled Zone**: Se encarga de realizar procesamientos más complejos sobre los datos (Ejemplo: cálculo de KPIs, agregaciones de los datos, entre otros), en comparación a los procesamientos realizados en zonas anteriores. A su vez, es la primera zona que realiza procesamientos sobre los datos, buscando adaptarlos a los casos de uso del sistema (Giebler y cols., 2020). Esta zona es muy similar a la **Refined Zone** de Zaloni, aunque creemos que algunas características de la **Refined Zone** también se pueden identificar en la **Delivery Zone**, descrita a continuación.

5. **Delivery Zone y Explorative Zone**: Ambas zonas se derivan de la idea de Sandbox de la arquitectura de Zaloni. Mientras que la **Delivery Zone** contiene datos procesados y adaptados para varias aplicaciones de negocio (incluyendo análisis OLAP), la **Explorative Zone** es donde los DST pueden realizar análisis sobre los datos almacenados en las distintas zonas (Giebler y cols., 2020).

Por otro lado, estos autores incorporan el concepto de áreas protegidas dentro de su arquitectura. Por lo tanto, en caso que existan datos críticos o confidenciales, cada zona debería contar con un área "protegida" donde se realizan procesamientos sobre estos datos y donde se almacenan los mismos (Giebler y cols., 2020). A su vez, para acceder a estas áreas se deben tener credenciales específicas y solo los usuarios autorizados pueden acceder a las mismas. En el caso de la zona Sandbox, si los DST acceden a datos de las áreas protegidas de otras zonas, cualquier almacenamiento temporal de los datos y transformaciones realizadas en esta zona, se debe llevar a cabo en el área protegida de la misma

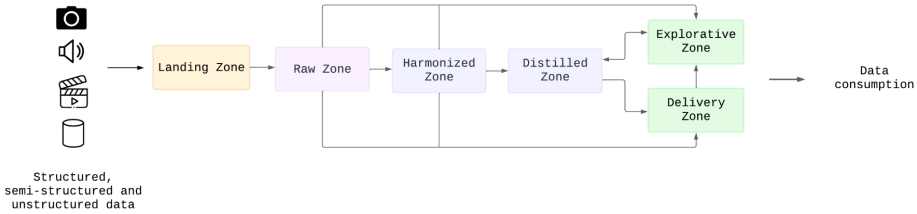

Figura 3.7: Data Vault based Zone Architecture propuesta en (Giebler y cols., 2020)

(Giebler y cols., 2020).

La principal desventaja de esta arquitectura es el modelado de los datos en la Harmonized Zone. Esta zona exige que todos los datos sigan un mismo enfoque de modelado, por lo que se debe invertir tiempo en modelar los datos para que conformen con determinado enfoque, cuando a futuro puede que no sea beneficioso, y se tenga que deshacer ese modelado. A su vez, dependiendo del enfoque de modelado seleccionado, esta transformación podría generar esperas innecesarias para el acceso a datos, debido al doble modelado de los datos en algunos casos.

Por últmo, se analizó la arquitectura presentada en (Herden, 2020) (Figura 3.8). A continuación se describen las distintas zonas que se definen en esta arquitectura.

1. **Raw Data Store:** Zona donde se almacenan los datos en formato crudo. Notar que esta zona también se encarga de llevar a cabo los procesos de extracción de los datos de las fuentes (*ingestion process*).

2. **Data Preparation:** Zona donde se generan los perfiles de los datos (*data profile*), donde se busca comprender distintos aspectos de los mismos con respecto a estandarización, limpieza y resumen.

3. **Processed Data Store:** Zona donde se almacenan los perfiles de los datos. A su vez, los datos pueden ser accedidos en esta zona mediante herramientas de AA.

4. **Analytics Sandbox:** Zona donde los DST pueden trabajar con los datos.

La principal desventaja de esta arquitectura es que no realiza procesamiento sobre los datos, sino que solo se generan perfiles de los mismos, donde se identifican distintas características de los mismos. Por último, queremos destacar que este autor hace mención a algunas problemáticas de los DL, como la gestión de metadatos y seguridad, así como gobierno de datos. En particular, el autor menciona que para tener un buen gobierno de datos, se debe contar con un catálogo de datos en la Raw Data Store, donde se lleve un registro de todo el

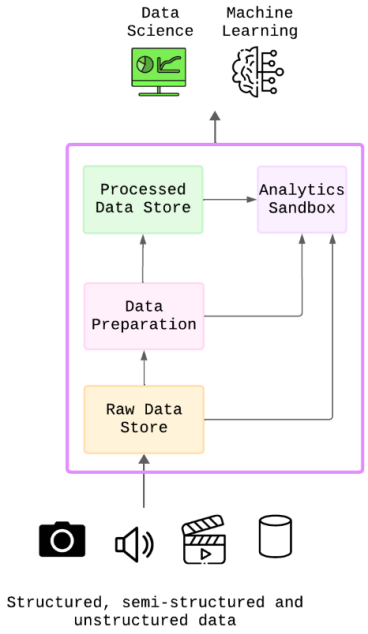

Figura 3.8: Arquitectura basada en zonas propuesta por (Herden, 2020).

ciclo de vida de los datos. También menciona que dependiendo de los tipos de datos utilizados y las regulaciones existentes, puede ser importante llevar a cabo procesos de seguridad y privacidad de los mismos, como la anonimización de estos. Por último, menciona que tanto el gobierno de datos, como los procesos de seguridad llevados a cabo generan metadatos, por lo que los DL deberían mantener un respositorio de metadatos, el cual puede ser accedido por todos los componentes del DL. Sin embargo, a pesar de mencionar aspectos como la gestión de metadatos y seguridad, así como el gobierno de datos, este autor no los incluye en la arquitectura propuesta.

Arquitecturas con Gobierno de datos

En esta sección se presentan las arquitecturas de DL que incorporan aspectos de gobierno de datos.

Los autores de (Zhao y cols., 2021), presentan una arquitectura formada por 4 zonas, descritas a continuación (Figura 3.9).

1. **Raw Data Zone:** Se encarga de extraer los datos de la fuente y almacenarlos en formato "crudo". Los datos batch se almacenan en el mismo formato que tenían en la fuente, mientras que los datos de stream se almacenan luego del procesamiento de stream, en caso que se deseen almacenar.

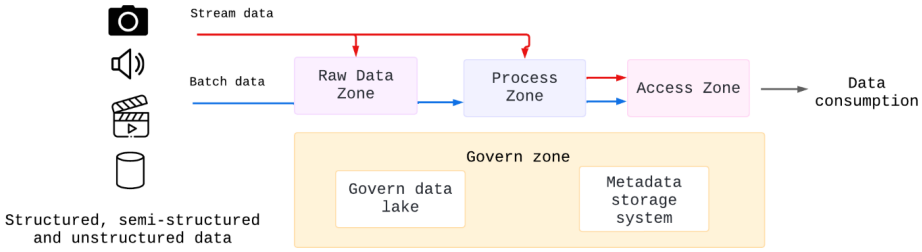

Figura 3.9: Arquitectura en zonas propuesta por los autores de (Zhao y cols., 2021)

2. **Process Zone:** Permite que los usuarios realicen transformaciones sobre los datos según sus necesidades, permitiendo almacenar resultados intermedios y procesos. A su vez, se pueden realizar procesamientos de stream y batch sobre los datos.

3. **Access Zone:** Los datos procesados puede ser accedidos en esta zona para ser analizados o consumidos por distintas herramientas. Los datos deben poder ser accedidos por herramientas de visualización, análisis en tiempo real, soporte de decisiones, BI y ML.

4. **Govern Zone:** Esta zona se encarga de asegurar la seguridad y calidad de los datos, así como llevar a cabo la gestión de metadatos y del ciclo de vida de los datos (*data life-cycle*). Está compuesta por el componente *Metadata storage* que lleva a cabo la gestión de metadatos, y el componente *Govern data lake* que se encarga de llevar a cabo mecanismos de seguridad, como autenticación, autorización y cifrado, así como políticas de seguridad para evitar ataques internos y externos al sistema.

La principal desventaja que observamos en esta arquitectura es la falta de una zona o componente donde los DST puedan interactuar con los datos del sistema, sin interferir con el procesamiento de los mismos para otros usos.

Por otro lado, queremos destacar la existencia de la *Govern Zone*; ya que es la primera arquitectura que tiene en cuenta aspectos como la gestión de metadatos y mecanismos de seguridad. Creemos que esta zona es sumamente importante; ya que sin una correcta gestión de metadatos, el DL se puede convertir en un *data swamp*. A su vez, creemos que es importante tener mecanismos de seguridad para controlar el acceso a las distintas zonas y datos; ya que pueden haber datos sensibles o confidenciales a los cuales no todo usuario deba acceder.

Por último, se presenta la arquitectura propuesta en (Oukhouya y cols., 2021), similar a la presentada en (Zhao y cols., 2021) y (Sharma, 2018) (Ver Figura 3.10). A continuación describimos las distintas zonas de esta arquitectura.

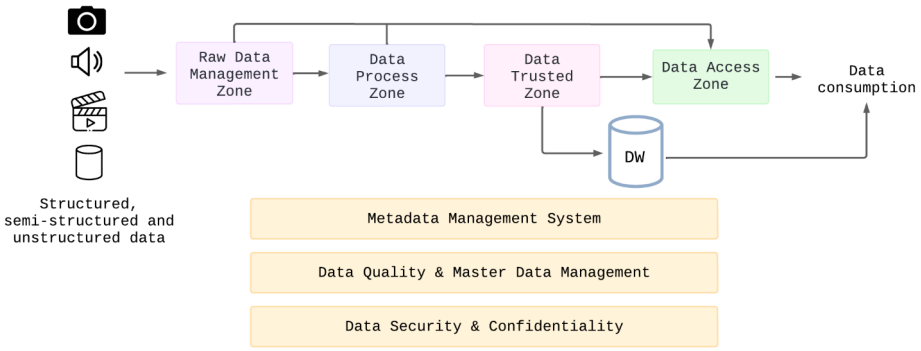

Figura 3.10: Arquitectura basada en zonas propuesta por (**Oukhouya y cols., 2021**).

1. **Raw data management zone:** Permite extraer datos de la fuente e integrarlos en el mismo formato que tenían en la fuente.
2. **Data process zone:** Esta zona se encarga de realizar transformaciones sobre los datos.
3. **Data Trusted Zone:** Esta zona almacena datos con un alto nivel de limpieza. A su vez, estos datos se pueden utilizar como fuente de datos para un DW.
4. **Data Access Zone:** Esta zona permite que los usuarios accedan a todas las zonas del DL.
5. **Governance zone:** Esta zona se encarga de la gestión de metadatos (Componente *Metadata Management System*), de calidad (Componente *Data Quality and Master Data Management*) y de seguridad y confidencialidad de los datos (Componente *Data Security & Confidentiality*).

Esta arquitectura, al igual que la presentada en (Zhao y cols., 2021), cuenta con un componente que se encarga del Gobierno de los datos dentro del DL. Sin embargo, al igual que la arquitectura descrita anteriormente, no cuenta con una zona donde los DST puedan explorar los datos y realizar análisis sobre los mismos, sin distorsionar el modelado de los mismos para casos de uso de otros usuarios.

3.3. Data Lakehouse

En esta sección se presentan los resultados del estado del arte realizado para el concepto de DLH. En la subsección 3.3.1 se presentan distintas definiciones académicas del concepto DLH. Por otro lado, en la subsección 3.3.2 se presenta

un conjunto de características que debería cumplir un DLH, obtenidas a partir del análisis de las propuestas académicas. Por último, en la subsección 3.3.3 se presenta el estudio de arquitecturas de DLH proporcionadas por proveedores de cloud, y la clasificación de las mismas según las características definidas en la subsección 3.3.2.

3.3.1. Definiciones

En esta subsección se presentan distintos trabajos que buscan definir un concepto de "Data Lakehouse". En particular, se analizaron las publicaciones (Schneider, Gröger, Lutsch, Schwarz, y Mitschang, 2023), (Armbrust y cols., 2021), (B. Inmon y cols., 2021), (Harby y Zulkernine, 2022), (Mazumdar y cols., 2023) y (Orescanin y Hlupic, 2021).

Definición de DLH presentada en (Schneider y cols., 2023)

Los autores de este trabajo realizan un estudio de la literatura existente con respecto al concepto "Data Lakehouse". A continuación se presentan las distintas definiciones obtenidas a partir del estudio.

1. **Definición de (Armbrust y cols., 2021):** "Sistema de gestión de datos, basado en sistemas de almacenamiento de bajo costo que permiten tener un acceso directo a los datos almacenados. A su vez, posee características de gestión de los DBMS tradicionales, así como transacciones ACID, control de versiones de datos, auditoría, indexación, almacenamiento en caché y optimización de consultas". Las primeras dos características "sistema de almacenamiento de bajo costo y acceso directo al almacenamiento" se corresponden con características de DLs, mientras que las demás características se pueden encontrar en un DW.

2. **Definición de (Hansen, s.f.):** "Un DLH es un enfoque arquitectónico que logra administrar todo tipo de datos y soporta distintas cargas de trabajo (BI, AA, streaming)". Con respecto a esta definición, los autores plantean que es una definición muy amplia y que las distintas formas de integrar un DW con un DL (Sección 3.4) se pueden ver como enfoques arquitectónicos que cumplen con esta definición.

Además de analizar los distintos conceptos del término DLH, también investigan las distintas formas de integrar un DL con un DW. Los enfoques de integración que analizan son similares a los que se presentan en la Sección 3.4, pero incorporan un cuarto enfoque de integración, denominado "Integrated Architecture". En este enfoque no se diferencia un DL y un DW, sino que es un único sistema que logra proporcionar las funcionalidades de ambos sistemas.

Luego del análisis de las distintas nociones de DLH, justifican que las características de los DW y DL con respecto al acceso a datos y tipo de almacenamiento, son opuestas entre sí, por lo que no creen que sea posible crear una

única plataforma que garantice todas las propiedades de ambos sistemas. A partir de esta idea es que proponen su propia definición, presentada a continuación.

Definición de (Schneider y cols., 2023): "Un DLH es una plataforma integrada que utiliza el mismo tipo de almacenamiento y formato de datos para reportes y OLAP, así como para Minería de datos, Aprendizaje automático y cargas de trabajo de streaming"

Además de plantear la definición anterior, establecen distintos requisitos que debe seguir un DLH (Ver Tabla 3.1).

Tabla 3.1: Requisitos que debe cumplir un DLH propuesto por (Schneider y cols., 2023)

Requisitos	Descripción
Mismo tipo de almacenamiento y mismo formato de almacenamiento	El requisito "'mismo tipo de almacenamiento"hace referencia a que todos los datos y metadatos deben estar almacenados en un único sistema de almacenamiento, altamente escalable (Por ejemplo, HDFS D.2). Por otro lado, el requerimiento "mismo formato de almacenamiento" hace referencia a que todos los datos, excepto los metadatos, deben estar almacenados utilizando un mismo formato (Por ejemplo: .csv o .parquet)
CRUD para todos los datos	Es necesario para todos los usos del sistema poder almacenar (Create), acceder (Retrieve), modificar (Update) y eliminar (Delete) datos.
Colecciones de datos relacionales	Se deben poder definir relaciones a nivel lógico sobre archivos almacenados en el sistema.
Query language	Se debe tener un lenguaje declarativo estructurado que permita consultar datos estructurados.
Garantizar la consistencia de los datos	El DLH debe proveer formas de garantizar la consistencia de los datos en una colección, tanto para análisis OLAP como para minería de datos.
Aislamiento y atomicidad	Debe proveer aislamiento y atomicidad, por lo menos para los datos que se usen para reportes y OLAP.

Requisitos	Descripción
Direct read access	Debe permitir el acceso directo a los datos almacenados en el sistema, así como a los metadatos de los mismos por parte de diversas herramientas (ML, Minería de datos, BI, entre otros)
Procesamiento de stream y batch unificado	Debe soportar el procesamiento casi-en-tiempo-real de registros de datos estructurados, por lo menos para las operaciones append, update y read. A su vez, debe poder combinar procesamiento de stream y batch, manteniendo la integridad de los datos.

Por último, estos autores analizan distintas herramientas existentes en el mercado, y las comparan contra los requisitos de la Tabla 3.1, llegando a la conclusión que las herramientas Delta Lake, Apache Hudi y Apache Iceberg (Ver Anexo C) cumplen con la definición de DLH. A su vez, plantean que estas herramientas siguen el enfoque arquitectónico "Integrated Architecture".

Definición de DLH presentada en (Armbrust y cols., 2021)

Definen un DLH como "un sistema de gestión de datos, basado en sistemas de almacenamiento de bajo costo que permiten tener un acceso directo a los datos almacenados. A su vez, posee características de gestión de los DBMS tradicionales, como transacciones ACID, control de versiones de datos, auditoría, indexación, almacenamiento en caché y optimización de consultas". Por último, presentan una arquitectura de DLH que toma como sistema de almacenamiento un DL y construyen sobre este distintos componentes que permiten llevar a cabo la gestión de metadatos, dar soporte a herramientas de ML y DS, y brindar acceso a los datos, garantizando buena performance para las consultas SQL (Ver Anexo A.1.1). Este trabajo es la primera aproximación a la plataforma Lakehouse de Databricks (*The scope of the lakehouse platform*, s.f.) (Ver Anexo A.2.2).

Definición de DLH presentada en (Orescanin y Hlupic, 2021)

Estos autores se basan en la definición planteada en (Armbrust y cols., 2021) y proponen un DLH como la integración de un DW y un DL (Ver Figura 3.11). A su vez, proponen que el acceso al DLH se haga a través de un capa virtual.

En la Tabla 3.2 se describen los distintos componentes de esta arquitectura.

Concepto de DLH presentado en (Harby y Zulkernine, 2022)

Los autores de (Harby y Zulkernine, 2022) no presentan una definición concreta del concepto DLH, pero proponen una arquitectura de DLH y presentan

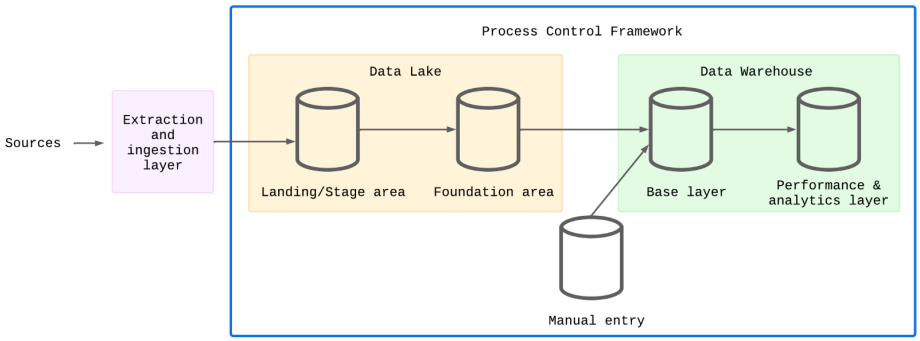

Figura 3.11: Arquitectura propuesta por (Orescanin y Hlupic, 2021)

Componente	**Descripción**
Extraction & ingestion layer	Se encarga de extraer los datos de la fuente.
Landing/Stage area (DL)	Área de almacenamiento temporal, donde los datos que se extraen de las fuentes se almacenan hasta que sea momento de procesarlos. Una vez enviados para ser procesados, se eliminan de esta área.
Foundation area (DL)	Almacena datos históricos que pueden haber sufrido cambios.
Manual entry area (DW)	Contiene atributos, dimensiones y jerarquías que no existen en las fuentes de datos.
Base layer (DW)	Almacena datos transformados.
Performance & analytics layer (DW)	Almacena tablas adicionales para análisis, con los datos agregados de la Base layer.
Process Control Framework (PCF)	Framework que se implementa como una aplicación web, que se encarga de la planificación y ejecución de trabajos. A su vez, se debe encargar del paralelismo y optimización de performance.

Tabla 3.2: Componentes de la arquitectura propuesta por (Orescanin y Hlupic, 2021)

una serie de características que debería tener este sistema (Ver Anexo A.1.2). A su vez, para proponer la arquitectura de DLH, los autores investigan distintas herramientas que permitan cumplir con los requisitos establecidos. Sin embargo, no existe una definición clara de cada componente de la arquitectura que proponen.

Concepto de DLH presentado en (B. Inmon y cols., 2021)

En (B. Inmon y cols., 2021) se define un DLH como "un sistema que permite realizar análisis de datos, utilizando estructuras de datos y características de gestión de datos similares a las de un DW, mientras se almacenan los datos en un sistema de almacenamiento de bajo costo, como un DL".

Por otro lado, el autor también menciona que un DLH está formado por dos grandes componentes: un DL donde se almacenan datos crudos de distinto tipo (estructurados, semi-estructurados, no-estructurados) y una infraestructura analítica (*analytical infrastructure*) donde los usuarios finales pueden acceder a los datos. Esta infraestructura analítica debe poseer procesos ETL, aspectos de gestión de metadatos (información de los datos almacenados en el sistema, relaciones entre datos, granularidad de los datos, existencia de claves, actualización del dato, procesamientos realizados sobre los datos, entre otros) y aspectos de calidad de datos.

Por último, el autor menciona que si a un DL se le incorpora una infraestructura analítica, a este nuevo sistema se lo puede llamar DLH.

Concepto de DLH presentado en (Mazumdar y cols., 2023)

Nuevamente, los autores de (Mazumdar y cols., 2023) no dan una definición concreta para el concepto DLH, pero presentan diversas características y capacidades que debe tener este tipo de sistema (Ver Anexo A.1.3).

A su vez, proponen una arquitectura de DLH (Ver Figura 3.12) que luego implementan utilizando las tecnologías Apache Iceberg, Amazon S3 y Dremio Sonar.

A continuación se describen los componentes de la arquitectura de DLH propuesta por los autores.

1. **Storage:** Componente donde se almacenan los datos luego de la ingestion.
2. **Storage engine:** Componente que se encarga de la gestión de datos, incluyendo la compactación, reparticionamiento e indexación de los mismos.
3. **File formats:** Componente que almacena los datos crudos utilizando formatos abiertos de archivos.
4. **Table formats:** Componente que funciona como una capa de metadatos sobre los datos almacenados en la capa "File formats".

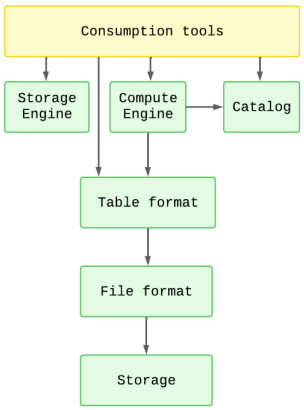

Figura 3.12: Arquitectura propuesta por (Mazumdar y cols., 2023)

5. **Catalog:** Componente que permite la búsqueda y consulta de datos en el DLH.

6. **Compute engine:** Componente encargado de realizar procesamiento de datos.

3.3.2. Discusión de propuestas académicas

A partir del análisis de las distintas propuestas de DLH encontradas en la literatura, identificamos diversos aspectos que debería proporcionar un DLH, listados a continuación.

1. **R1:** Almacenamiento de distintos tipos de datos (estructurado, semi-estructurado y no-estructurado) (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021; Harby y Zulkernine, 2022; Orescanin y Hlupic, 2021).

2. **R2:** Formatos abiertos (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021).

3. **R3:** Almacenamiento de grandes volúmenes de datos en sistemas de bajo costo (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021).

4. **R4:** Transacciones ACID (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021; Orescanin y Hlupic, 2021; Schneider y cols., 2023; Harby y Zulkernine, 2022).

5. **R5:** Consultas con baja latencia (Indexación, caching, etc.) (Armbrust y cols., 2021; Mazumdar y cols., 2023).

6. **R6:** Gobierno de datos (B. Inmon y cols., 2021; Mazumdar y cols., 2023).

7. **R7:** Soporte para herramientas de BI (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021; Harby y Zulkernine, 2022; Orescanin y Hlupic, 2021; Schneider y cols., 2023).

8. **R8:** Soporte para herramientas de AA (B. Inmon y cols., 2021; Mazumdar y cols., 2023; Armbrust y cols., 2021; Harby y Zulkernine, 2022; Orescanin y Hlupic, 2021; Schneider y cols., 2023).

9. **R9:** Acceso directo para lectura de datos (*direct read access*) (B. Inmon y cols., 2021; Armbrust y cols., 2021; Schneider y cols., 2023).

A partir de estas características clasificamos las distintas propuestas de DLH (Ver Tabla 3.3).

Características	(Armbrust y cols., 2021)	(Orescanin y Hlupic, 2021)	(Mazumdar y cols., 2023)
(R1)	X	X	X
(R2)	X	X	X
(R3)	X	X	
(R4)	X	parcial	X
(R5)	X	parcial	X
(R6)	parcial		parcial
(R7)	X	X	X
(R8)	X	X	X
(R9)	X	X	X

Tabla 3.3: Comparación de propuestas de DLH según características identificadas.

Notar que la arquitectura de DLH presentada en (Orescanin y Hlupic, 2021) cumple parcialmente las características (R4) y (R5), debido a que estas solo se cumplen para los datos almacenados en el DW. Por otro lado, las propuestas (Armbrust y cols., 2021) y (Mazumdar y cols., 2023) cumplen parcialmente (R6); ya que solo incluyen gestión de metadatos. No se clasificó la propuesta de los autores (Harby y Zulkernine, 2022); ya que no se tiene una descripción clara de los componentes de la arquitectura que proponen.

3.3.3. Propuestas de proveedores

Además de realizar un estudio de propuestas académicas que tratan el concepto DLH, se investigaron distintas plataformas proporcionadas por proveedores de cloud.

A continuación se describen aspectos generales identificados en estos sistemas.

1. La mayoría de estos sistemas utilizan DLs como almacenamiento base de la arquitectura. A su vez, los datos almacenados en el DL se almacenan en formatos de archivos abiertos. En el caso de la plataforma *Lakehouse Dremio* (Ver Anexo A.2.4), por más que solo brinda un motor de consultas SQL, este motor trabaja sobre datos almacenados en un DL.

2. Para brindar propiedades de DWs sobre los datos almacenados en DLs, estos sistemas utilizan diversos formatos de tabla que funcionan como una capa de metadatos sobre los datos almacenados en formatos de archivos abiertos, optimizados con fines analíticos (Ver Sección 3.7).

3. Algunas propuestas soportan el acceso a los datos por parte de herramientas de AA (*Databricks Lakehouse*, *Azure Synapse Analytics*, *BigLake* y *Cloudera Data Platform*), mientras que otras permiten el acceso a datos mediante motores de consultas SQL (*Lakehouse Dremio* y *Snowflake*). Por otro lado, todas permiten el acceso a datos por parte de herramientas de BI.

4. La gran mayoría permite el procesamiento de datos, tanto de stream como batch.

5. Algunas de estas plataformas permiten la gestión del gobierno de datos (*Databricks Lakehouse* y *Cloudera Data Platform*), mientras que otras requieren de la integración con otros servicios para brindar estas funcionalidades (*Snowflake*, *Azure Synapse Analytics* y *BigLake*).

En el Anexo A.2 se puede encontrar una descripción más detallada de cada una de las plataformas investigadas.

Por último, en la Tabla 3.4 se presenta la clasificación de estas plataformas según las características de DLH identificadas en la subsección anterior.

3.4. Enfoques de integración de un Data Warehouse y un Data Lake

Para presentar las distintas formas en las cuales se puede integrar un DW con un DL, nos basamos en el trabajo de (Herden, 2020). Para integrar estos dos sistemas el autor utiliza una arquitectura de DL basada en zonas (Ver Sección 3.2.3).

En las siguientes secciones se describen los distintos enfoques de integración.

3.4.1. Secuencial

El primer patrón de integración implica utilizar un DW y DL en forma secuencial (Ver Figura 3.13). Esto es, todos los datos deben pasar previamente

Requisito	Azure Synapse Analytics	Databricks Lakehouse	Snowflake	Lakehouse Dremio	BigLake	Cloudera Data Platform
(R1)	X	X	X		X	X
(R2)	X	X	X		X	X
(R3)	X	X	X		X	X
(R4)	X	X	X	X	X	X
(R5)	X	X	X	X	X	X
(R6)		X				X
(R7)	X	X	X	X	X	X
(R8)	X	X			X	X
(R9)	X	X	X	X	X	X

Tabla 3.4: Comparación de plataformas de proveedores según características de DLH identificadas en la subsección 3.3.2.

por el DL y luego se cargan en el DW.

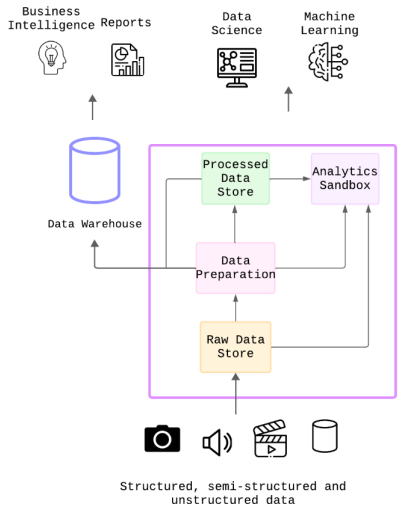

Figura 3.13: Integración secuencial

En este patrón el DL reemplaza todos los procesos ETL del DW. Este enfoque presenta una gran desventaja en el caso que se esté integrando un DW ya existente; ya que esto tiene un gran costo de reimplementación de procesos dentro del DL (Herden, 2020). Para evitar esta reimplementación de procesos se podrían implementar dentro del DL únicamente aquellos procesos que son

útiles tanto para el DL como para el DW y mantener los procesos específicos del DW dentro de los procesos ETL.

3.4.2. Paralela

El segundo patrón implica integrar de forma paralela los dos sistemas (Ver Figura 3.14). En este caso ambos sistemas funcionan de forma independiente.

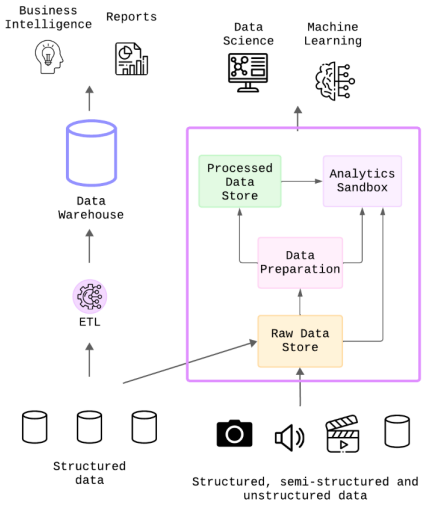

Figura 3.14: Integración paralela

En caso que se quiera integrar un DL a un DW ya existente, el esfuerzo de implementación recae únicamente en la construcción del DL. Sin embargo, la gran desventaja de este enfoque es que no hay una integración entre los datos del DL y del DW, debido a que estos sistemas funcionan de forma independiente. Por un lado, las aplicaciones que consumen datos del DW no logran acceder a datos del DL. Por otro lado, es probable que no se puedan comparar los resultados de las herramientas de AA con los de las herramientas de BI, debido a que el DL contiene datos que el DW no posee.

3.4.3. Offloading

El tercer patrón se caracteriza por el proceso de Offloading, el cual se encarga de enviar datos del DW a la zona Raw Data Store del DL (Ver Figura 3.15). Los datos que se envían mediante este proceso son datos que no son accedidos frecuentemente (*cold data*) y datos maestros (*master data*).

3.4.4. Híbrida

A continuación se describen distintos patrones de integración que combinan los patrones de las secciones 3.4.1, 3.4.2 y 3.4.3.

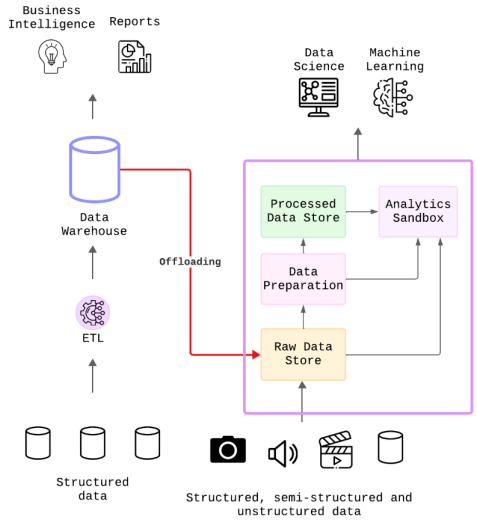

Figura 3.15: Offloading del DW al DL

Secuencial con Offloading

El primer enfoque híbrido incorpora el proceso de Offloading desde el DW al Raw Data Store del DL, en una integración Secuencial de ambos sistemas (Ver Figura 3.16).

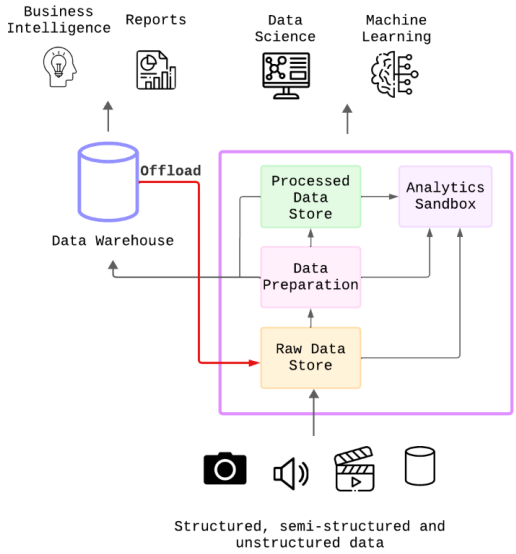

Figura 3.16: Integración secuencial con Offloading

Este enfoque sigue teniendo el problema de la reimplementación de procesos ETL dentro del DL en el caso de sistemas de DW ya existentes.

Paralela con Offloading

El segundo enfoque híbrido incorpora el proceso de Offloading desde el DW al Raw Data Store del DL, además de un proceso de Onloading desde el DL al DW, en el enfoque de integración Paralelo de ambos sistemas (Ver Figura 3.17).

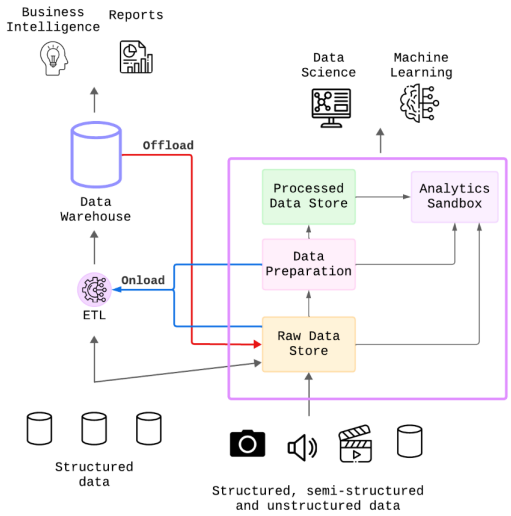

Figura 3.17: Integración paralela con Offloading

Al incorporar el proceso de Onloading se logra una completa integración entre ambos sistemas, desventaja del enfoque paralelo; ya que se logran compartir datos entre ambos sistemas.

Luego de analizar los distintos enfoques, creemos que el enfoque Paralelo con Offloading es el más adecuado, por las siguientes razones:

1. Permite integrar un DW nuevo o uno ya existente con un DL, sin necesidad de reimplementar procesos.
2. Se pueden realizar procesamientos comunes (limpieza, integración) en el DL y enviar los resultados a los procesos ETL, mientras que estos realizan procesamientos específicos para el DW, en paralelo.
3. El proceso de Offloading permite liberar espacio dentro del DW y así mantener la performance del acceso a los datos, sin perder esos datos.
4. El proceso de Onloading permite que las aplicaciones que acceden al DW, puedan acceder a datos de las fuentes no estructuradas, luego de que estos fueron procesados dentro del DL.

5. Los procesos de Offloading y Onloading permiten que las herramientas de BI y AA puedan acceder a todos los datos; ya que se comparten datos entre ambos sistemas.

3.5. Procesamiento de stream y batch

Existen diversas arquitecturas que permiten brindar un procesamiento híbrido de datos, esto es, procesamiento de stream y batch. A continuación, se describen las características de las arquitecturas Lambda, Kappa y BRAID (Giebler, Stach, Schwarz, y Mitschang, 2018), mencionando sus respectivas ventajas y desventajas. Se utiliza el trabajo (Giebler y cols., 2018) como base para describir estas 3 arquitecturas.

3.5.1. Arquitectura Lambda

Esta arquitectura presenta dos ramas de procesamiento, una dedicada al procesamiento batch (Batch Layer) y la otra dedicada al procesamiento de stream (Stream Layer). Cada una de estas ramas contiene su propia lógica de procesamiento, aislada de la otra rama (Ver Figura 3.18).

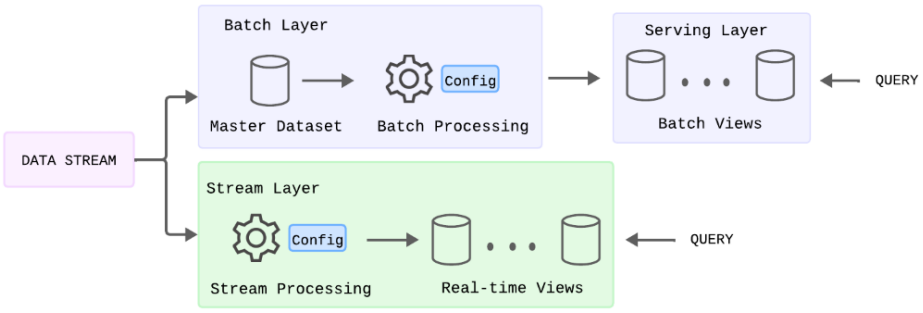

Figura 3.18: Arquitectura Lambda

La *Stream Layer* permite modificar su lógica de procesamiento, pudiendo propagar estos cambios sobre los datos entrantes de forma inmediata. Sin embargo, la *Batch Layer* no permite que se realicen cambios sobre la lógica de procesamiento una vez iniciado el mismo, siendo esta una de las desventajas de esta arquitectura.

Debido a la separación de la lógica de procesamiento de ambas capas, las vistas generadas a partir del procesamiento pueden diferir. Esto genera que si los usuarios desean tener una vista global sobre los datos de ambas ramas, tengan que combinar los resultados de forma manual.

3.5.2. Arquitectura Kappa

Esta arquitectura busca eliminar la última desventaja de la arquitectura Lambda; ya que utiliza un único motor de procesamiento para realizar tanto el procesamiento de stream como el de batch (Ver Figura 3.19).

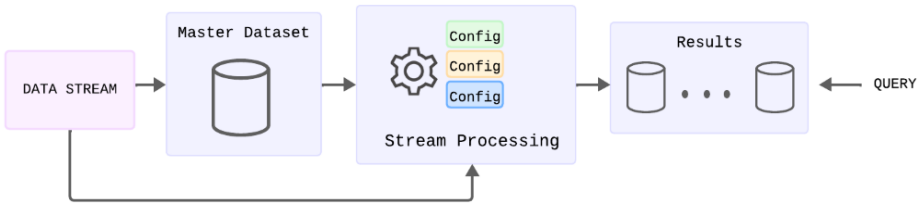

Figura 3.19: Arquitectura Kappa

Por lo tanto, a medida que los datos llegan al sistema, se almacenan en un repositorio llamado Master Dataset, mientras que a su vez, son enviados al motor de procesamiento para realizar el procesamiento de stream. Al utilizar un único motor de procesamiento, se pueden definir trabajos en paralelo para realizar el procesamiento de los datos, basta con mantener distintos archivos de configuración.

La principal desventaja de esta arquitectura tiene que ver con el poder de cómputo necesario para poder realizar procesamiento batch en paralelo al procesamiento de stream; ya que se usa un mismo motor de procesamiento.

3.5.3. Arquitectura BRAID

Ambas arquitecturas descritas anteriormente permiten realizar un procesamiento de datos de stream y batch, con sus respectivas ventajas y desventajas. Sin embargo, ninguna de ellas permite intercambiar resultados intermedios entre ambos procesamientos. Frente a este requisito es que surge la arquitectura BRAID (Ver Figura 3.20).

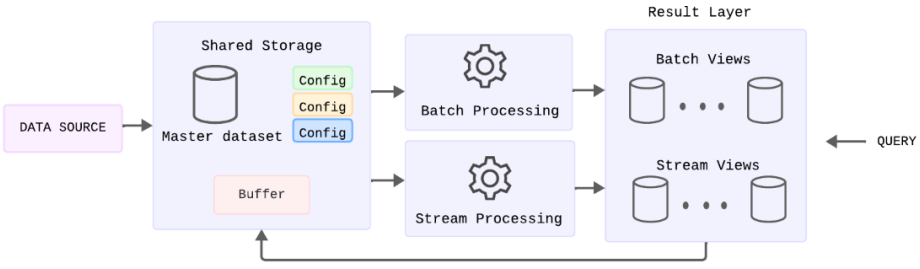

Figura 3.20: Arquitectura BRAID

Cuando los datos llegan al sistema se almacenan en un repositorio común, denominado Master Dataset, ubicado en el componente Shared Storage de la arquitectura, el cual está disponible para ambos procesamientos (stream y batch). Por otro lado, los datos sobre los cuales se realizará un procesamiento en tiempo real se almacenan en un buffer. Luego, los datos son propagados a alguna de las dos ramas de procesamiento, batch o stream.

El componente Shared Storage es quien permite compartir resultados intermedios entre las dos ramas de procesamiento; ya que los resultados de cualquiera de las dos ramas pueden ser enviados hacia este componente para actuar como entrada para la otra rama. Este componente también almacena los archivos de configuración donde se define la lógica de procesamiento para ambas ramas. Cada rama puede modificar los archivos de configuración para cambiar la lógica de procesamiento.

Por otro lado, la rama denominada Batch Processing Branch, es la que se encarga de realizar el procesamiento batch, leyendo periódicamente datos del repositorio Master Dataset y realizando un procesamiento sobre los mismos. Por otro lado, la rama denominada Stream Processing Branch, es la que se encarga de realizar el procesamiento de stream de los datos que llegan al sistema. Por último, los resultados se envían a un mismo componente de almacenamiento, denominado Result Layer.

Esta arquitectura logra resolver los problemas de la arquitectura Lambda, pero además, permite compartir resultados entre las ramas.

3.6. Modelado de metadatos en Data Lakes

Una de las características de los DL es su capacidad de almacenar distintos tipos de datos, ya sean datos estructurados, semi-estructurados o no estructurados. Sin embargo, el almacenamiento de datos sin metadatos asociados, puede llevar a este sistema a convertirse en un *data swamp*. Este problema hace que el DL sea percibido por los usuarios como un sistema poco confiable. Por lo tanto, para evitar que el DL se convierta en un *data swamp*, es imprescindible contar con una buena gestión de metadatos (Ravat y Zhao, 2019b).

Se analizaron distintas propuestas de modelos de metadatos, pero nos vamos a enfocar en los trabajos (Ravat y Zhao, 2019b), (Megdiche, Ravat, y Zhao, 2021), (Eichler, Giebler, Gröger, Schwarz, y Mitschang, 2020) y (Oukhouya y cols., 2021).

Existen distintas formas de clasificar los metadatos dentro de un DL, por lo que previo a la definición del modelo de metadatos, los distintos autores definen en qué clasificación se van a basar. Los autores de (Ravat y Zhao, 2019b)

se basan en la clasificación de metadatos que los subdivide en Inter-metadata e Intra-metadata. Esta clasificación tiene en cuenta la información dentro de un dataset (intra-metadata) y las relaciones que pueden existir entre distintos datasets (inter-metadata). Los autores de (Megdiche y cols., 2021), (Eichler y cols., 2020) y (Oukhouya y cols., 2021) se basan en los trabajos de (Ravat y Zhao, 2019b), por lo que todos utilizan la misma clasificación.

Por otro lado, todos los autores se basan en un arquitectura de DL basada en zonas para definir su modelo, las cuales difieren en la cantidad de zonas que posee cada una.

En (Ravat y Zhao, 2019b) se presenta un modelo de metadatos centrado en los datos, donde se modelan aspectos como:

1. Metadatos de los datasets.
2. Relaciones entre los datasets.
3. Acceso a los datos por parte de usuarios con roles definidos y las aplicaciones o procesos utilizados para acceder a los datos.

Por otro lado, los autores en (Megdiche y cols., 2021) presentan el mismo modelo de (Ravat y Zhao, 2019b), al cual le incorporan metadatos asociados a los procesos que se llevan a cabo dentro del DL. A su vez, estos autores incorporan 2 entidades dentro de su modelo dedicadas a la calidad de los datos (Quality Measure y Quality Analysis), aspecto sumamente importante para este trabajo. Igualmente, creemos que estas entidades no son suficientes para poder modelar los metadatos de calidad; ya que faltan aspectos como las Dimensiones, Factores y Métricas de calidad, asociados a las mediciones (Quality Measure). A su vez, estos autores extienden este modelo en su trabajo (Zhao y cols., 2021), donde incorporan metadatos de los análisis realizados por herramientas de AA. Algunos de los aspectos modelados son la implementación de los análisis (algoritmos utilizados para el análisis y sus respectivos parámetros), modelos que se obtienen como resultado del análisis y evaluación del modelo obtenido (métricas utilizadas para realizar la evaluación y valores obtenidos).

Por su parte, los autores de (Oukhouya y cols., 2021) proponen 8 funcionalidades que debe cumplir un sistema de gestión de metadatos y comparan distintos modelos contra esas funcionalidades. En particular, estos autores analizan el trabajo de (Ravat y Zhao, 2019b) y (Megdiche y cols., 2021), donde reconocen que no se tienen en cuenta aspectos de granularidad de los datos y tampoco se modela la evolución de los datasets. Luego, los aspectos que modelan los autores de (Oukhouya y cols., 2021) son:

1. Información relevante de cada dataset, incluyendo aspectos referentes a la calidad del mismo.
2. Relaciones entre los distintos datasets.

3. Granularidad de los datos dentro de los datasets. Los autores definen un concepto Dataset, al cual se lo relaciona con otro concepto DataItem el cual tiene un atributo de tipo donde se indica si ese DataItem es una columna, celda, tabla, clave, valor, etc.

4. Evolución de esquemas de los datasets. Los autores definen un concepto *Change* el cual lleva registro de los cambios en los atributos del dataset, los cuales pueden ser de tipo, nombre o valor.

5. Acceso a los datos por parte de usuarios y las aplicaciones utilizadas para ello.

Por último, los autores de (Eichler y cols., 2020) analizan varias propuestas de modelos de metadatos, entre ellas la de los autores de (Megdiche y cols., 2021), argumentando, nuevamente, que los mismos no consideran aspectos de la granularidad de los datos.

Los aspectos que creemos importante del modelo de (Eichler y cols., 2020) son:

1. Modelado de la zona del DL donde se encuentra el dato, permitiendo hacer un seguimiento de las zonas por donde ha pasado el dato.

2. Granularidad del dato.

3. Fuente de donde proviene el dato.

A partir del estudio de estos modelos, creemos que los siguientes aspectos deberían incluirse en un modelo de metadatos de un DL.

1. Información de cada dataset almacenado en la arquitectura, incluyendo el tipo (estructurado, no estructurado, semi estructurado), información de la fuente de donde se obtuvo el dataset, fecha de extracción, entre otros aspectos. (Ravat y Zhao, 2019b)(Megdiche y cols., 2021)(Eichler y cols., 2020)(Oukhouya y cols., 2021).

2. Relaciones entre datasets (Ravat y Zhao, 2019b)(Megdiche y cols., 2021)(Oukhouya y cols., 2021).

3. Granularidad de los datos (Eichler y cols., 2020) (Oukhouya y cols., 2021).

4. Vinculación de los datos con las respectivas zonas o componentes del sistema (Eichler y cols., 2020).

5. Evolución de los datasets (Oukhouya y cols., 2021).

6. Procesos y Análisis llevados a cabo sobre los datos (Megdiche y cols., 2021) (Zhao y cols., 2021).

7. Acceso a los datos y aplicaciones utilizadas para ello (Ravat y Zhao, 2019b)(Megdiche y cols., 2021) (Oukhouya y cols., 2021).

8. Aspectos de calidad de los datos (Megdiche y cols., 2021).

9. Aspectos de los análisis realizados sobre los datos por parte de DST o expertos de ML (implementación, evaluación y resultados) (Zhao y cols., 2021).

3.7. Formatos de datos

En esta sección se describen distintos formatos de almacenamiento de datos que se consideran en las arquitecturas de BD. En particular, se mencionan diversos formatos de archivos (Subsección 3.7.1) y formatos de tabla (Subsección 3.7.2).

3.7.1. Formatos de archivos

En esta subsección se describen diversos formatos de archivos desarrollados por diversas organizaciones, para optimizar consultas analíticas.

Estos tipos de archivos pueden almacenar los datos usando un enfoque orientado a filas o columnas. El enfoque orientado a filas almacena los valores de cada fila de forma contigua (Ver Figura 3.21). Por otro lado, el orientado a columnas define grupos de filas y almacenan los datos de las columnas de cada grupo de forma contigua (Ver Figura 3.22).

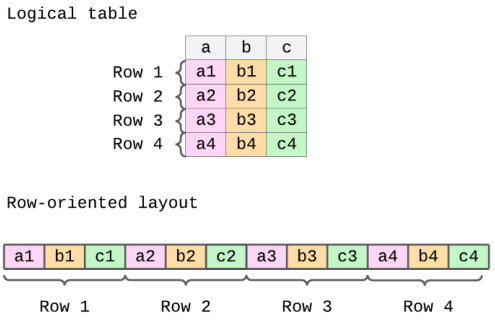

Figura 3.21: Almacenamiento de datos orientado a filas.

Apache Parquet

Apache Parquet es un formato abierto de archivo orientado a columnas, cuya estructura se presenta en la Figura 3.23.

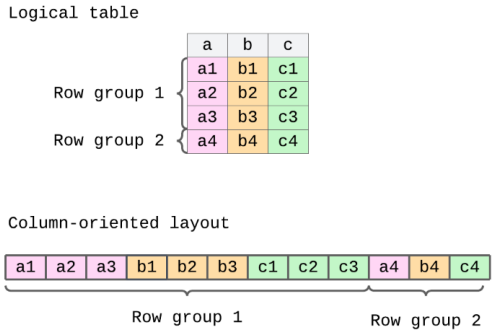

Figura 3.22: Almacenamiento de datos orientado a columnas.

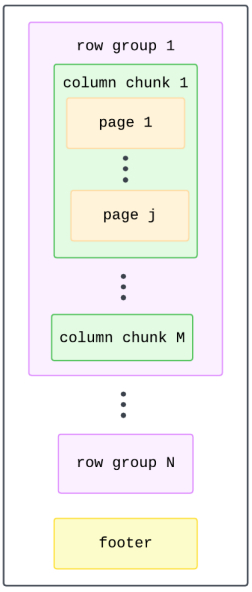

Figura 3.23: Estructura de archivos Parquet

Los archivos *Parquet* se dividen en *row groups*, donde cada uno de ellos contiene *column chunks*, para cada una de las columnas en el dataset. Luego, si se tiene una tabla con M columnas y se dividen las filas en N grupos, el archivo va a tener N *row groups*, cada uno con M *column chunks*. Los *column chunks* almacenan información de la columna correspondiente. A su vez, se dividen en *pages* que son quienes almacenan los datos de cada columna. Por otro lado, en el *footer* del archivo se almacenan metadatos del archivo, de las columnas y algunas estadísticas de los datos de las columnas (por ejemplo, valores mínimos y máximos en una columna) (*File Format*, s.f.).

A la hora de realizar consultas analíticas, los formatos orientados a columnas suelen ser más performantes que los orientados a filas (Ver Anexo B). Por último, este formato de archivo también cuenta con otras optimizaciones como compresión de datos a nivel de columna y codificación, pero presenta la desventaja de inferir mayores tiempos de escritura; ya que no solo se deben escribir los datos, sino que también se deben almacenar metadatos (*What Is Apache Parquet? | Dremio*, s.f.).

Optimized Row Columnar

El formato de archivo Optimized Row Columnar (en adelante, ORC) es un formato de archivo columnar. El archivo se subdivide en *stripes*, las cuales almacenan un conjunto de filas. Luego, cada *stripe* se subdivide en *data chunks* donde cada *chunk* almacena datos para una columna específica (*Background - Apache ORC*, s.f.) (Ver Figura 3.24). A su vez, al final del archivo se almacenan metadatos, como el esquema de los datos, estadísticas de las columnas e índices.

Por otro lado, el archivo también puede almacenar índices para columnas específicas y tener un acceso más rápido a los datos (*Background - Apache ORC*, s.f.).

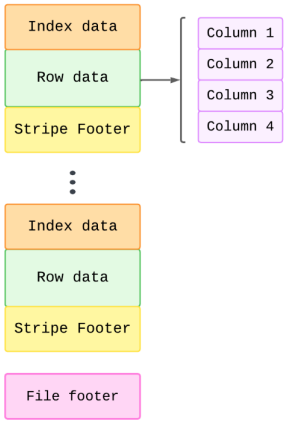

Figura 3.24: Estructura de archivos ORC.

Por último, este archivo también presenta funcionalidades de compresión de datos, soporte para tipos de datos complejos y *predicate pushdown* (Ver Anexo B) (*Background - Apache ORC*, s.f.).

Avro

Avro es un framework de serialización de datos, donde se define el formato de archivo *Avro*, orientado a filas.

Este formato almacena los datos serializados y el esquema de los mismos en un mismo archivo (Ver Figura 3.25). Los datos se pueden serializar siguiendo un formato binario o JSON. (*Specification*, s.f.).

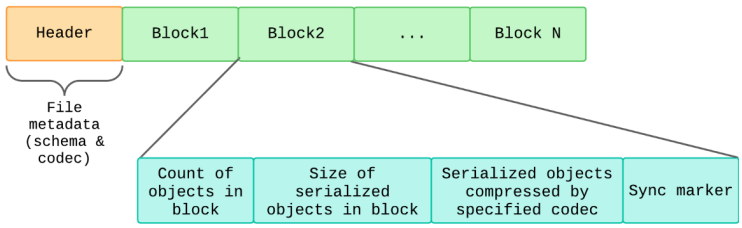

Figura 3.25: Estructura de archivos Avro.

Este formato también presenta funcionalidades de compresión a nivel de archivo, soporte de tipos de datos complejos y soporte para la evolución de esquemas (campos faltantes, nuevos campos y campos editados) (*Specification*, s.f.).

Además, es útil en casos donde se trabaja con datos de stream; ya que el formato binario reduce el tamaño de los datos a transmitir.

3.7.2. Formatos de tabla

En esta subsección se describen algunos formatos de tabla *open-source* existentes en el mercado, los cuales buscan brindar funcionalidades de DW y bases de datos relacionales para datos almacenados en DLs.

La mayoría de estos formatos de tabla utilizan como almacenamiento algunos de los formatos de archivo descritos en la subsección 3.7.1.

Apache Iceberg

Apache Iceberg define un formato de tabla sobre un conjunto de archivos de datos (*data files*) en formato *Parquet*, *Avro* u *ORC*. El estado de estas tablas se mantiene en archivos de metadatos, los cuales almacenan información como el esquema de la tabla, configuraciones de particiones de los datos, snapshots del contenido de la tabla (Representa el estado de la tabla en un momento determinado en el tiempo), entre otros (*Spec - Apache Iceberg*, s.f.).

Por otro lado, este formato de tabla presenta un log transaccional para garantizar transacciones ACID, también posee soporte para evolución de esquemas, permite realizar *time-travel queries* (permiten consultar estados previos de la tabla) y *rollbacks* de la tabla (*Introduction - Apache Iceberg*, s.f.).

Ver Anexo C.1 para una descripción detallada de este formato de tabla.

Apache Hudi

Apache Hudi (Hadoop Upserts Deletes and Incrementals) provee un formato de tabla el cual utiliza los esquemas de *Avro* para almacenar, gestionar y evolucionar el esquema de las tablas (*Use Cases | Apache Hudi*, s.f.). Sobre este formato de tablas se tienen garantizadas transacciones ACID; ya que se mantiene un log transaccional donde se almacenan las acciones realizadas sobre la tabla (*Use Cases | Apache Hudi*, s.f.).

Ver Anexo C.2 para una descripción detallada de este formato de tabla.

Delta table

Las *Delta tables* son formatos de tabla que definen un log transaccional sobre archivos *Parquet*. Este formato de tabla garantiza transacciones ACID, gracias al log transaccional, así como evolución de esquemas y *time-travel queries* (*delta/PROTOCOL.md at master · delta-io/delta*, s.f.).

Ver Anexo C.3 para una descripción detallada de este formato de tabla.

3.8. Análisis final

A partir del análisis del estado del arte de las arquitecturas de BD (Secciones 3.1, 3.3 y 3.2), logramos identificar algunas características que deben cumplir los sistemas de BD, presentadas en la Tabla 3.5. Estos criterios se obtuvieron principalmente de los trabajos de tipo "survey" donde se analizan distintas arquitecturas de BD.

Para finalizar, se clasifican las distintas propuestas teóricas y de proveedores analizadas en la sección 3.2, y subsecciones 3.3.1 y 3.3.3, según las características definidas (Ver Tabla 3.6 y 3.7).

Características	Referencias
(C1) Ingestión y almacenamiento de datos heterogéneos (estructurados, semi estructurados y no estructurados)	(Hai y cols., 2021; Zagan y Danubianu, 2020; Ramchand y Mahmood, 2022; Giebler y cols., 2019; Schneider y cols., 2023; Harby y Zulkernine, 2022)
(C2) Almacenamiento de datos crudos	(Hai y cols., 2021; Zagan y Danubianu, 2020; Ramchand y Mahmood, 2022; Giebler y cols., 2019)
(C3) Almacenamiento de grandes volúmenes de datos en sistemas de bajo costo	(Hai y cols., 2023, 2021; Zagan y Danubianu, 2020; Ramchand y Mahmood, 2022; Giebler y cols., 2019; Harby y Zulkernine, 2022)
(C4) Extracción, modelado y almacenamiento de metadatos (Metadata management)	(Hai y cols., 2023, 2021; Giebler y cols., 2019; Schneider y cols., 2023; Mazumdar y cols., 2023)
(C5) Acceso a datos por parte de herramientas de BI	(Hai y cols., 2023, 2021; Schneider y cols., 2023; Harby y Zulkernine, 2022; Mazumdar y cols., 2023)
(C6) Acceso a datos por parte de herramientas de AA	(Hai y cols., 2023, 2021; Schneider y cols., 2023; Harby y Zulkernine, 2022; Mazumdar y cols., 2023)
(C7) Procesamiento y transformación de los datos (data integration, data cleaning, metadata enrichment)	(Hai y cols., 2023, 2021; Ramchand y Mahmood, 2022; Harby y Zulkernine, 2022; Mazumdar y cols., 2023)
(C8) Almacenamiento de datos procesados	(Giebler y cols., 2019; Schneider y cols., 2023)
(C9) Gobierno, calidad y seguridad de datos dentro de la arquitectura	(Hlupić y cols., 2022; Ramchand y Mahmood, 2022; Giebler y cols., 2019; B. Inmon y cols., 2021; Mazumdar y cols., 2023)
(C10) Procesamiento de datos de stream y batch	(Benjelloun y cols., 2020; Giebler y cols., 2018; Schneider y cols., 2023)
(C11) Transacciones ACID	(Schneider y cols., 2023; Armbrust y cols., 2021; Mazumdar y cols., 2023; B. Inmon y cols., 2021; Harby y Zulkernine, 2022).
(C12) Consultas con baja latencia (Indexación, caching)	(Armbrust y cols., 2021; Mazumdar y cols., 2023; Harby y Zulkernine, 2022).

Tabla 3.5: Características de arquitecturas de Big Data

	Two-Layered	Data Pond	Multi-layer	Zaloni Architecture	(Giebler y cols., 2020)	(Herden, 2020)	(Zhao y cols., 2021)	(Oukhouya y cols., 2021)	(Hai y cols., 2023) y (Hai y cols., 2021)	(Armbrust y cols., 2021)	(Orescanin y Hlupic, 2021)	(Mazumdar y cols., 2023)
(C1)	X	X	X	X	X	X	X	X	X	X	X	X
(C2)	X		X	X	X	X	X	X	X	X	X	X
(C3)	X	X	X	X	X	X	X	X	X	X	X	X
(C4)			X				X	X	X	X		X
(C5)					X			X		X	X	X
(C6)	X	X	X	X	X	X	X	X	X	X	X	X
(C7)		X	X	X	X	X	X	X	X	X	X	X
(C8)		X	X	X	X		X	X	X	X	X	
(C9)							X	X				
(C10)		X					X					
(C11)										X	parcial	X
(C12)										X	parcial	X

Tabla 3.6: Clasificación de propuestas teóricas según las características presentadas en la Tabla 3.5

	Azure Synapse Analytics	**Databricks Lakehou-se**	**Snowflake**	**Lakehouse Dremio**	**BigLake**	**Cloudera Platform**
(C1)	X	X	X		X	X
(C2)	X	X	X		X	X
(C3)	X	X	X		X	X
(C4)		X	X			X
(C5)	X	X	X	X	X	X
(C6)	X	X			X	X
(C7)	X	X	X		X	X
(C8)	X	X	X		X	X
(C9)		X				X
(C10)	X	X	X		X	X
(C11)	X	X	X	X	X	X
(C12)	X	X	X	X	X	X

Tabla 3.7: Clasificación de arquitecturas de proveedores según las características presentadas en la Tabla 3.5

Capítulo 4

Arquitectura de Big Data propuesta

En este capítulo se presenta la arquitectura genérica de BD propuesta. Para ello, en la primera sección se define una arquitectura funcional abstracta, la cual busca incorporar las distintas características de arquitecturas de BD identificadas en la sección 3.8. Luego, en la segunda sección del capítulo se presenta la arquitectura final, la cual busca instanciar la arquitectura funcional de la primera sección, inspirándose en arquitecturas de BD existentes. En la tercera sección se verifica si la arquitectura propuesta cumple con las características identificadas en la Sección 3.8. Por último, en la cuarta sección se proponen mejoras sobre la arquitectura propuesta, basadas en tecnologías de proveedores, que permiten definir la arquitectura como un DLH.

4.1. Arquitectura funcional abstracta

A partir de las características definidas en 3.8 para arquitecturas de BD se define una arquitectura que refleja las funcionalidades que poseen estas arquitecturas, la cual se puede ver en la Figura 4.1.

A continuación se describen los distintos componentes y funcionalidades principales que conforman esta arquitectura.

1. **Data Ingestion:** A partir de la caracterísitca **(C1)** es claro que el sistema debe poder ingerir datos de fuentes heterogéneas, por lo que en el diagrama se puede visualizar un componente que se encarga de la ingestión de datos con distintos formatos.

2. **Data storage:** Este componente busca incluir las características **(C1)**, **(C2)**, **(C3)** y **(C8)**, ya que proporciona un sistema de almacenamiento, de bajo costo cuando es posible, en el cual se pueden almacenar grandes

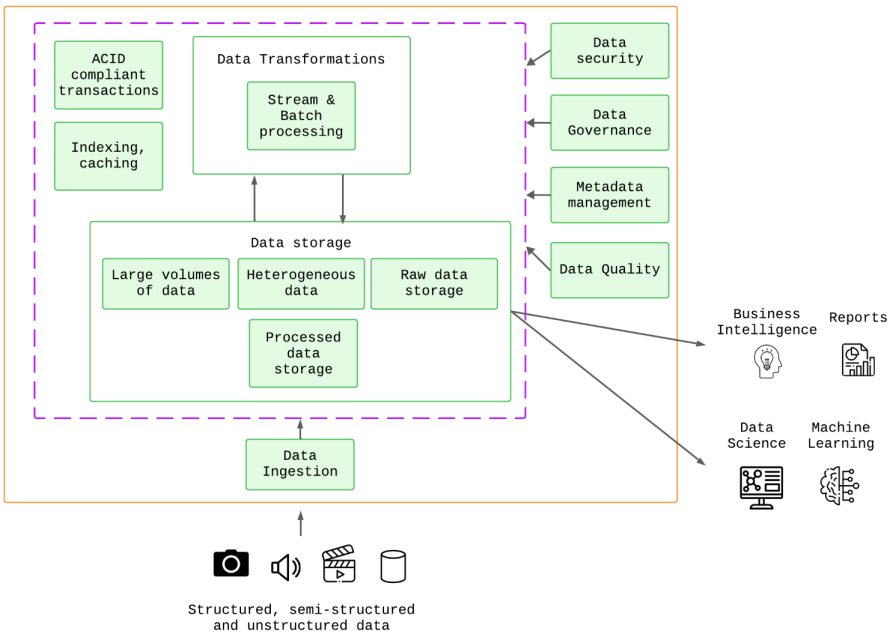

Figura 4.1: Arquitectura funcional generada a partir de criterios definidos en 3.8

volúmenes de datos, datos crudos, datos con distintos formatos y datos procesados.

3. **Acceso a datos por parte de herramientas de BI y AA:** Para cumplir con las características **(C5)** y **(C6)**, esta arquitectura debe permitir el acceso a datos por ambos tipos de herramientas.

4. **Transformaciones de datos:** Para cumplir con las características **(C7)** y **(C10)** se tiene el componente *Data Transformations*. Este componente debe permitir el procesamiento de datos, tanto de stream como batch.

5. **Gestión de Metadatos, gobierno, calidad y seguridad de datos:** Para cumplir con **(C9)** se incorporan los componentes *Data Quality*, *Data Governance* y *Data Security*, los cuales se encargan de la gestión de estos 3 aspectos. A su vez, para cumplir con **(C4)** se incorpora el componente *Metadata management* el cual debe permitir la extracción, modelado y almacenamiento de metadatos. Por último, estos componentes actúan sobre los componentes de procesamiento y almacenamiento de datos (*Data Transformations* y *Data Storage*).

6. **Transacciones ACID, Indexación y caching de datos:** Para cumplir con las características **(C11)** y **(C12)** se incorporan los componentes *Indexing, caching* y *ACID compliant transactions*, los cuales actúan sobre

el procesamiento y almacenamiento de datos (Componentes *Data Storage* y *Data transformations*).

4.2. Descripción de la arquitectura de Big Data propuesta

Para definir esta arquitectura se tomó la arquitectura de la sección 4.1 y se instanciaron sus componentes en base a arquitecturas existentes para las distintas cargas de trabajo de BD (BI y AA). La arquitectura propuesta está formada principalmente por un DL, al cual se le pueden integrar diversos DWs (Ver Figura 4.2).

Se elige un DL como principal sistema de almacenamiento y procesamiento de la arquitectura; ya que las características (C1), (C3) y (C6), son típicas de estos sistemas. A su vez, existen diversas arquitecturas de DL que permiten cumplir con las características (C2), (C4) y (C9), por lo que adoptando una de estas arquitecturas, se logra cumplir con esas características. Por otro lado, para cumplir con (C5), (C11) y (C12) se incorporan diversos DW en la arquitectura, mientras que para cumplir con (C10), se incorpora una arquitectura híbrida, que permite realizar procesamientos de stream y batch. Por último, las características (C7) y (C8) se pueden encontrar en ambos sistemas (DW y DL).

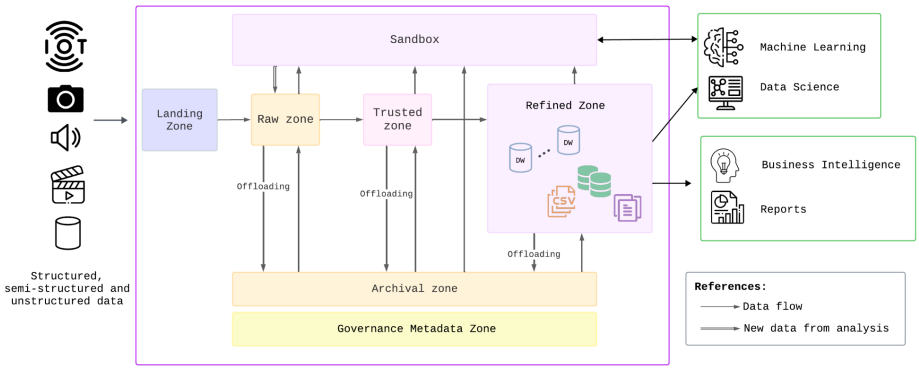

Figura 4.2: Arquitectura propuesta simplificada

Esta sección se divide en 6 subsecciones. La primera subsección busca argumentar la elección de la arquitectura de DL adoptada. La segunda subsección describe los distintos componentes de la arquitectura de DL adoptada. En la tercera subsección se presenta el enfoque utilizado para integrar DWs al DL. A su vez, en la cuarta subsección se presenta la integración de una arquitectura híbrida para el procesamiento de stream y batch. Por último, la quinta subsección describe los distintos usuarios del sistema e indica el acceso de los mismos

a los distintos componentes de la arquitectura.

4.2.1. Discusión de propuestas de arquitecturas de Data Lake

A continuación se presenta una discusión de las distintas arquitecturas de DL investigadas en la Sección 3.2, argumentando las razones por las cuales se decidió utilizar una arquitectura basada en zonas.

Debido a que se quiere evitar que el DL se convierta en un data swamp, la Two Layered Architecture es descartada. Esto se debe a que esta arquitectura no tiene ningún mecanismo para gestionar los metadatos dentro del DL y tampoco realiza procesamientos de los mismos, simplemente los almacena.

La Data Pond Architecture se descartó debido a las siguientes razones:

1. Los DST necesitan acceder a los datos en distintos formatos y etapas de procesamiento para poder realizar análisis sobre los mismos, y obtener resultados que pueden ser de ayuda para la organización. Sin embargo, esta arquitectura no cuenta con un componente sobre el cual los DST puedan acceder a los datos y operar con ellos, sin distorsionar los datos que fueron modelados para un caso de uso específico.

2. No se tiene un almacenamiento de los datos crudos (*raw data*). Por lo tanto, si se realizan cambios sobre algún procesamiento que se lleva a cabo dentro del DL, no se van a poder propagar los cambios sobre datos que ya se encuentran dentro del sistema; ya que no se tiene la versión "cruda" de los datos.

Por otro lado, por más que la arquitectura Multi-Layered tiene aspectos similares a las arquitecturas basadas en zonas, decidimos descartarla debido a que no cuenta con una zona similar a una "Sandbox" para que los DST puedan interactuar con los datos en distintas etapas de procesamiento.

Con respecto a las arquitecturas basadas en zonas, elegimos descartar las presentadas en (Zhao y cols., 2021) y (Oukhouya y cols., 2021). Al igual que la Multi-Layered Architecture, estas arquitecturas no cuentan con una zona donde los DST puedan explorar y realizar análisis sobre los datos, sin interferir con otros procesamientos que se estén llevando a cabo sobre los mismos.

Por otro lado, se descarta la arquitectura de (Herden, 2020); ya que esta no posee un almacenamiento de datos procesados, sino que simplemente almacena el *profile* de los datos.

Por último, la arquitectura de (Giebler y cols., 2020), exige ciertos requerimientos de modelado en determinadas zonas, los cuales creemos que son muy restrictivos; ya que puede haber casos de uso donde sea más útil utilizar otro

tipo de modelado. Luego, en esos casos se insume tiempo en realizar un doble modelado de los datos, el primero por la exigencia impuesta por la arquitectura, y el segundo por el caso de uso que requiere de otro tipo de modelado .

A partir de este análisis es que decidimos que la arquitectura de DL a utilizar esté fuertemente basada en la *Zaloni Architecture* (Sharma, 2018), tomando algunas ideas de (Giebler y cols., 2020), arquitectura en zonas similar a la de (Sharma, 2018). A su vez, se van a incorporar algunos aspectos de las demás arquitecturas como la existencia de un repositorio de metadatos global (Multi-Layered architecture), la extracción de metadatos en la primera zona (Multi-Layered architecture), una zona con las mismas funcionalidades que el Archival Data Pond (Data Pond Architecture), y una zona similar a la Govern Zone de (Oukhouya y cols., 2021) y (Zhao y cols., 2021).

En la Figura 4.3 se puede observar un diagrama simplificado de la arquitectura de DL final y en la siguiente sección se realiza una descripción de las distintas zonas que la conforman.

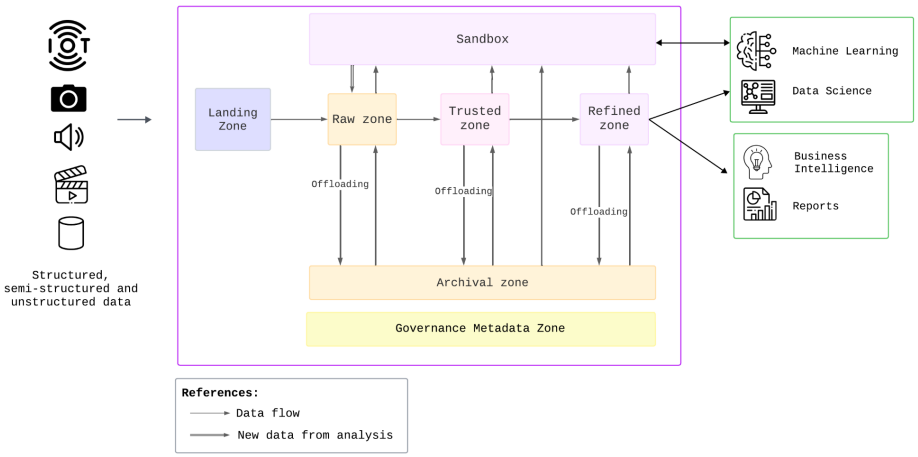

Figura 4.3: Arquitectura de DL elegida

4.2.2. Zonas del Data Lake

En esta subsección se describen las distintas zonas que conforman al DL de la arquitectura final (Ver Figura 4.3), indicando, cuando corresponde, las referencias en las que se basan las propuestas.

Landing Zone

Esta zona se encarga de extraer los datos de diversas fuentes y realizar controles de calidad y confidencialidad, en caso de ser necesario (Sharma, 2018).

A su vez, en caso que haya metadatos disponibles en las fuentes, estos se extraen y se almacenan en el repositorio de metadatos en la *Governance Metadata Zone* (Giebler y cols., 2020).

No hay persistencia de datos en esta zona, sino que aquellos datos que cumplen con los controles realizados se envían a la Raw Zone y luego son eliminados de esta zona. Si existen regulaciones de seguridad sobre los datos a ser almacenados en el sistema, las transformaciones y procesos pertinentes sobre los datos se deben realizan en esta zona (Ejemplo: masking o anonymizing) (Sharma, 2018).

Raw Zone

Esta zona almacena los datos que pasaron los controles de la Landing Zone (Sharma, 2018), manteniendo el formato y esquema que tenían en la fuente.

Archival Zone

La Archival Zone almacena datos fríos [1] (*cold data*) (Hlupić y cols., 2022) (B. Inmon, 2016), de cualquiera de los componentes de la arquitectura, excepto de la zona Sandbox y Landing. Estos datos se envían a esta zona a través de los procesos de Offloading.

Los datos almacenados en esta zona pueden volver a ser accedidos para análisis, por lo que se tiene un flujo de datos desde la Archival Zone hacia cada una de las zonas de almacenamiento y la Sandbox. En particular, cuando se requieren datos de la Archival Zone, estos son enviados a la última zona donde estaban almacenados (la zona que envío los datos a la Archival Zone a través del proceso de Offloading). Es importante notar que cuando los datos se envían a una nueva zona, estos se eliminan de la Archival Zone.

Para que los datos se retornen a la última zona donde estaban almacenados, se debe contar con metadatos que lleven un registro de la última zona en la que se encontraba el dato antes de ser enviado por el proceso de Offloading a la Archival Zone. A su vez, para identificar aquellos datos que no son frecuentemente accedidos, se deben tener metadatos asociados a la frecuencia de acceso de los mismos.

Por otro lado, la zona Sandbox puede acceder a datos de esta zona, solicitando una copia de los mismos. En este caso los datos no se eliminan de la zona

[1] Datos que no se usan o no se acceden frecuentemente

cuando se envían a la Sandbox.

Por último, en la teoría, no se manipulan ni se eliminan datos de esta zona. Sin embargo, en la práctica esto no es viable; ya que debido a las características de Big Data, la cantidad de datos almacenados crece rápidamente, por lo que se deben manipular y eliminar datos para gestionar el espacio de almacenamiento disponible en el sistema (Giebler y cols., 2020).

Trusted Zone

En la arquitectura presentada en (Giebler y cols., 2020) (Sección 3.2.3), se define la zona Harmonized. A partir de la descripción de esta zona, es claro que las funcionalidades de la misma son muy similares a las de la Trusted Zone propuesta en (Sharma, 2018) (Sección 3.2.3). Es por esto que tomamos funcionalidades de ambas propuestas y las incorporamos en la definición de nuestra Trusted Zone.

La Trusted Zone extrae datos "a demanda"de la Raw Zone, realizando una copia de los mismos. Esto es, a medida que se los necesita, se extraen datos de la zona anterior y estos no son eliminados de la Raw Zone, sino que se realiza una copia de los mismos (Sharma, 2018; Giebler y cols., 2020).

El principal propósito de esta zona es realizar modificaciones sobre los datos para cumplir con especificaciones de negocio, confidencialidad y calidad (Ejemplo de métodos a aplicar sobre los datos: data cleaning, data validation y data integration) (Sharma, 2018). Es importante remarcar que las transformaciones llevadas a cabo en esta zona son completamente independientes de casos de uso o herramientas que vayan a acceder a los datos (Giebler y cols., 2020).

Por otro lado, en esta zona no se exige que todos los datos sigan un mismo modelo. Los autores en (Giebler y cols., 2020) proponen que en la Harmonized Zone los datos estén integrados siguiendo un mismo enfoque de modelado, independientemente de su estructura (Por ejemplo, modelado multidimensional, modelado con data vault, entre otros). Esto no significa que todos los datos son parte de un mismo esquema, sino que pueden existir esquemas parciales con datos de determinadas fuentes o asociados a contextos particulares (Giebler y cols., 2020). No se va a exigir este requerimiento en la zona; ya que se estaría eliminando una de las características más importantes de los DL, la cual es permitir hacer análisis *on-the-fly* u *on-demand* (Ver Sección 2.3). Si impusiéramos la utilización de un mismo enfoque de modelado para todos los datos en esta zona (como sugiere (Giebler y cols., 2020)), se podría estar modelando datos siguiendo determinado esquema, que no fuera el requerimiento final para el acceso a los mismos. Esto implicaría invertir tiempo en realizar un doble modelado de los datos. En ese caso, se realizaría un primer procesamiento para seguir el enfoque de modelado de la Trusted Zone, y un segundo procesamiento para modelar los datos para su propósito final en la Refined Zone.

Por último, en esta zona se pueden realizar integraciones de datos, pero esto no implica que la integración involucre a todos los datos dentro de ella, sino que pueden ser integraciones parciales, que involucren algunos conjuntos de datos (Giebler y cols., 2020).

Refined Zone

La Refined Zone se encarga de modelar los datos para distintos casos de uso y herramientas (Giebler y cols., 2020). Además, es en esta zona donde los datos pueden ser accedidos por herramientas externas al sistema. Al igual que la Trusted Zone, esta zona extrae una copia de datos de la zona anterior (Trusted Zone), a demanda.

Por otro lado, el autor de (Sharma, 2018) propone que los datos en esta zona estén integrados siguiendo un formato común (por ejemplo, archivos .csv, .parquet, entre otros). No se exige este requerimiento; ya que esta zona es la responsable de modelar los datos para distintos casos de uso y/o herramientas, por lo que puede pasar que el formato elegido sea útil en algunos casos, y no en otros.

Sandbox

Esta zona proporciona un lugar para que los DST experimenten con los datos del DL (Sharma, 2018). Estos usuarios pueden acceder a copias de datos en cualquiera de las zonas del DL. A su vez, pueden realizar análisis y transformaciones de los datos, pudiendo modificar la sintaxis, el esquema y la granularidad de los mismos (Giebler y cols., 2020).

Por otro lado, no hay persistencia de los datos en esta zona, aunque sí se pueden almacenar transformaciones o modelos generados por los DST (Giebler y cols., 2020).

A su vez, se pueden enviar datos generados a partir del análisis y procesos ejecutados por los DST, a la Raw Zone. Esto se puede ver representado con una flecha doble entre ambas zonas (Ver Figura 4.2). Los datos no se envían a otras zonas; ya que esta zona no exige requisitos a nivel de limpieza, calidad o transformación de los datos. Si datos que sufrieron modificaciones en la zona Sandbox se enviaran a otra zona, que no es la Raw Zone, podrían estar modificados de tal forma que ya no cumplieran con los requerimientos de la zona a la cual se enviaron.

Governance Metadata Zone

Debido a las características de los datos a almacenar en nuestro sistema (grandes volúmenes de datos y diversidad de formatos), es necesario contar con

una gestión de metadatos para evitar que este se convierta en un data swamp. A su vez, los metadatos extraídos de las fuentes y los generados durante los procesos realizados sobre los datos, permiten verificar los orígenes de los datos y su historia de procesamiento, garantizando la confiabilidad de los mismos.

Por lo tanto, esta zona se encarga del almacenamiento de los metadatos que contribuyen a un correcto gobierno de datos.

Dentro de este componente se puede encontrar un repositorio de metadatos, el cual puede ser accedido por cualquiera de los procesos llevados a cabo dentro de la arquitectura (Hlupić y cols., 2022) (esto incluye los procesos ETLs de los DW, descritos en la sección 4.2.3), tanto para lectura como para escritura. Dentro de este repositorio se pueden encontrar los metadatos asociados a los datos y procesamientos llevados a cabo en la arquitectura (*General Metadata*), y los metadatos de calidad (*Data Quality Metadata*). También se podría distinguir un tercer tipo de metadatos, relacionados a aspectos de seguridad del sistema (Zhao y cols., 2021; Oukhouya y cols., 2021) (Ver Figura 4.4).

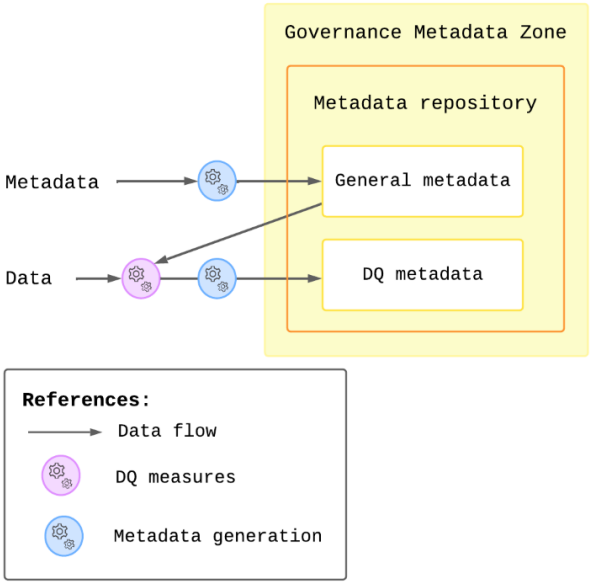

Figura 4.4: Generación de metadatos en la Governance Metadata Zone

Para poder almacenar los metadatos y permitir que estos sean consultados por diversos usuarios del sistema, es necesario definir un modelo de almacenamiento de los mismos. Luego, el procesamiento que se puede ver representado con un círculo azul en la Figura 4.4, hace referencia a la generación de los metadatos para conformar con el modelo adoptado.

En el caso de los metadatos de calidad, estos se generan a partir de la medición de la calidad, por lo que en la Figura 4.4 se puede ver representada con un círculo rosado la medición de calidad, y luego la generación de los metadatos correspondientes, que se almacenan en el repositorio de metadatos. También se puede ver cómo los procesos de medición de calidad tienen acceso a los metadatos de los datos y procesos.

Por último, creemos que el modelo de metadatos a utilizar en la arquitectura debería incluir los aspectos definidos al final de la Sección 3.6.

A este listado creemos que se deben incorporar metadatos asociados a los datos almacenados en un DW. Estos son metadatos acerca de la estructura del modelo multidimensional, permisos de acceso a los datos, procesos ETL aplicados, definiciones/descripciones de términos de negocio, entre otros.

Por lo tanto, un modelo de metadatos adecuado para nuestra arquitectura debería incorporar los siguientes aspectos (que enumeramos junto con referencias donde se proponen).

1. Información de cada dataset almacenado en la arquitectura, incluyendo su tipo (estructurado, no estructurado, semi-estructurado), información de la fuente de donde se obtuvo el dataset, fecha de extracción, entre otros aspectos (Ravat y Zhao, 2019b)(Megdiche y cols., 2021)(Eichler y cols., 2020)(Oukhouya y cols., 2021).

2. Relaciones entre datasets (Ravat y Zhao, 2019b)(Megdiche y cols., 2021)(Oukhouya y cols., 2021).

3. Granularidad de los datos (Eichler y cols., 2020) (Oukhouya y cols., 2021).

4. Vinculación de los datos con las respectivas zonas del DL que han atravesado (Eichler y cols., 2020).

5. Evolución de los datasets (Oukhouya y cols., 2021). Esto implica llevar un registro de los cambios que sufren los distintos datasets.

6. Procesos y Análisis llevados a cabo sobre los datos (Linaje de los datasets)(Megdiche y cols., 2021) (Zhao y cols., 2021).

7. Acceso a los datos y aplicaciones utilizadas para ello (Ravat y Zhao, 2019b)(Megdiche y cols., 2021) (Oukhouya y cols., 2021).

8. Aspectos de calidad de los datos (Megdiche y cols., 2021) (dimensiones, factores, métricas, medidas obtenidas, evaluaciones y procesos de mejora).

9. Aspectos de los análisis realizados sobre los datos por parte de DST o expertos de ML (implementación, evaluación y resultados) (Zhao y cols., 2021).

10. Metadatos asociados a datos almacenados en DW.

Para finalizar esta subsección, es importante remarcar que todas las zonas que contienen almacenamiento persistente de datos (*Raw, Trusted, Refined* y *Archival* Zone) deben tener la capacidad de almacenar distintos tipos de datos (estructurados, no-estructurados y semi-estructurados).

4.2.3. Incorporación de Data Warehouses en la arquitectura de Data Lake

Una de las características que debe cumplir una arquitectura de BD es dar soporte a las herramientas de BI. Sin embargo, como ya se mencionó en la Sección 3.2.3, la arquitectura de DL adoptada (basada en *Zaloni Architecture*) tiene la desventaja de no brindar soporte para estas herramientas.

Para resolver esta problemática se pueden integrar uno o varios DW con sus respectivos procesos ETL a la arquitectura de DL (Ver Figura 4.5).

Analizando las ventajas y desventajas de cada integración descrita en la Sección 3.4, la mejor forma de integrar estos sistemas es de forma paralela con los procesos de Onloading y Offloading correspondientes. Al realizar esta integración, el sistema logra brindar soporte a las herramientas pertinentes.

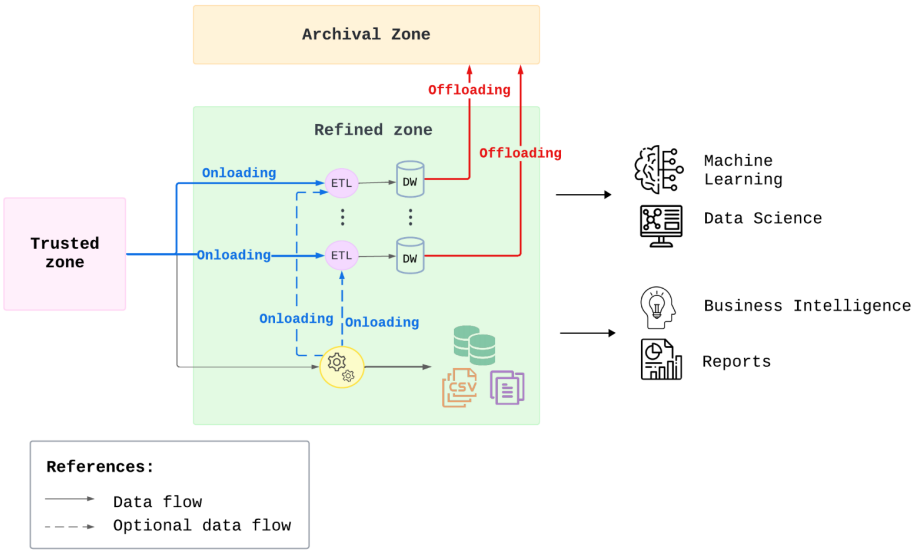

Figura 4.5: Flujo de datos entre la Refined Zone y los procesos ETL.

Los procesos de Onloading opcional de la Figura 4.5 evitan que los procesamientos comunes (aquellos procesamientos que sean de utilidad para datos a ser almacenados en los DW, así como para otros datos almacenados dentro de

la Refined Zone) se realicen varias veces. De esta manera, todo procesamiento común se realiza en el flujo principal de procesamiento de la Refined Zone (círculo amarillo en la Figura 4.5), y en paralelo los procesos ETL pueden realizar otros procesamientos específicos de los datos a ser almacenados en los DW; ya que estos también reciben como entrada datos de la Trusted Zone (proceso de Onloading mandatorio en Figura 4.5). El flujo que envía los datos resultantes de los procesamientos comunes está marcado con una flecha punteada; ya que no siempre va a haber procesamientos comunes entre el flujo principal de procesamiento de la zona y los procesos ETL.

Por otro lado, notar que los procesos ETL en la arquitectura dejan de ser los procesos ETL clásicos de los sistemas de DW; ya que hay algunas responsabilidades de estos procesos que se realizan en otras zonas del DL. En particular, en la Figura 4.6 se puede ver como la responsabilidad de Extracción (*Extract*) de los datos se asigna a la Landing Zone, las Transformaciones (*Transform*) se asignan a las zonas Trusted, Refined y procesos ETLs de los DW y la Carga (*Load*) de datos se asigna a los procesos ETL de los DW.

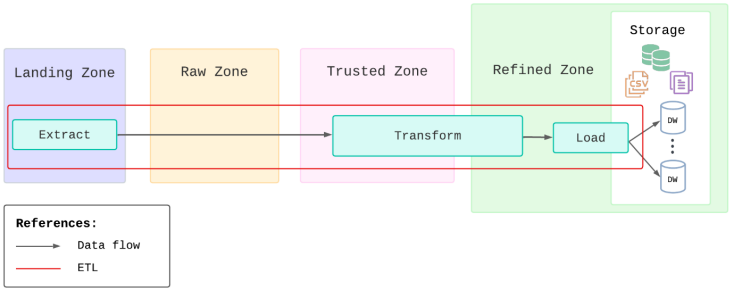

Figura 4.6: Procesos ETL a través de las zonas del DL

Esta forma de integrar estos dos sistemas presenta una ventaja para los DW; ya que la carga de procesamiento se puede dividir entre los procesos ETL y el flujo de procesamiento de la Refined Zone, permitiendo procesar datos en paralelo. Por último, el proceso de Offloading hacia la Archival Zone permite evitar tener un crecimiento desmedido del volumen de datos dentro de los diversos DW y mantener la performance de las consultas sobre los mismos, sin tener que eliminar estos datos del sistema; ya que se almacenan en la Archival Zone.

4.2.4. Procesamiento de stream y batch

Para cumplir con el requisito de brindar soporte para el procesamiento de stream y batch, se analizaron diversas arquitecturas, entre ellas, la arquitectura Kappa, Lambda y BRAID (Ver Sección 3.5). A partir de la discusión de las ventajas y desventajas de las distintas arquitecturas se decidió seleccionar la

arquitectura BRAID (Giebler y cols., 2018); ya que la misma permite realizar ambos tipos de procesamiento e intercambiar resultados entre las dos ramas de procesamiento.

En la Figura 4.7 se presenta la integración de esta arquitectura en la arquitectura propuesta. Notar que se eliminó la *Archival Zone* y los procesos de Offloading para simplificar el diagrama. Igualmente, en la Figura 4.9 se puede observar la arquitectura final, que incluye esta zona.

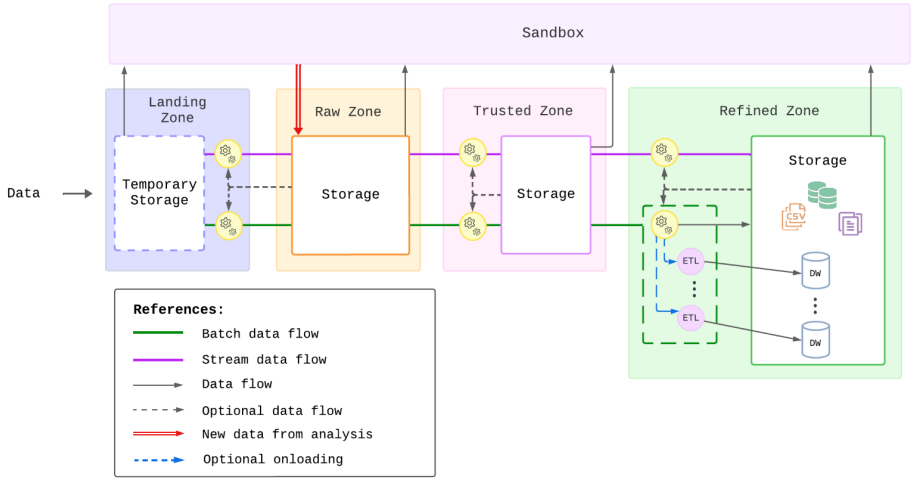

Figura 4.7: Integración de la arquitectura BRAID dentro de la arquitectura que integra un DL con DWs.

Dentro de la arquitectura basada en zonas elegida, las zonas Landing, Trusted y Refined tienen la responsabilidad de realizar un procesamiento de los datos, pudiendo ser este un procesamiento de stream o batch, generando resultados a partir del mismo. Para integrar la arquitectura BRAID dentro de las zonas del DL se tuvo que identificar los componentes *Shared Storage*, *Stream Processing*, *Batch Processing* y *Result Layer*, en cada zona.

En el caso de la Landing Zone, los datos que llegan al sistema se almacenan en un componente común de almacenamiento, denominado *Temporary Storage* (*Shared Storage* de la Arquitectura BRAID), y luego se envían a alguna de las dos ramas de procesamiento (*Stream Processing* y *Batch Processing* de la arquitectura BRAID). Notar que el componente de almacenamiento es temporal; ya que esta zona no almacena datos de forma permanente. Una vez que se realizan los procesamientos sobre los datos, estos se envían a la Raw Zone para ser almacenados. Por lo tanto, el componente de almacenamiento de la Raw Zone se corresponde con el *Result Layer* de la Arquitectura BRAID para los procesamientos realizados en la Landing Zone. En caso que se desee, se pueden obtener

resultados de procesamientos pasados almacenados en el *Storage* de la Raw Zone (Este flujo de datos se ve representado en la Figura 4.7 por una flecha punteada que sale del componente *Storage* de la Raw Zone hacia los procesamientos de la Landing Zone). Por otro lado, en el componente *Temporary Storage* se pueden encontrar los archivos de configuración de cada una de las ramas de procesamiento.

En el caso de las zonas Trusted y Refined, cada una de ellas recibe como entrada datos de la zona anterior, realiza un procesamiento sobre los mismos y almacena el resultado obtenido. El componente *Shared Storage*, de donde se obtienen los datos para procesar, se corresponde con el componente de almacenamiento de la zona anterior (*Storage*). Una vez que se obtienen los datos de la zona anterior, se envían a las ramas de procesamiento correspondiente y los resultados se almacenan en el componente *Storage* de cada zona, el cual se corresponde con el *Result Layer*. Este componente puede ser accedido por ambas ramas de procesamiento, en caso que se deseen utilizar resultados de procesamientos anteriores, y es quien almacena los archivos de configuración de las ramas de procesamiento de la zona.

Notar que en el caso de la Refined Zone, los procesos ETL también forman parte del procesamiento batch, por lo que estos acceden a datos de la zona anterior a través de los procesos de Onloading. Además, para los procesos ETL, el *Result Layer* es el DW correspondiente, el cual es parte del componente *Storage* de la zona.

Por otro lado, notar que la zona Sandbox no forma parte del flujo principal de procesamiento de los datos; ya que esta zona es para exploración por parte de los DST. Esto no significa que no se pueda realizar procesamiento de stream y batch, sino que los procesamientos que ellos realizan con los datos se realizan por fuera del flujo principal del sistema. Es por esto que desde todas las zonas se tienen flujos de datos desde los componentes de almacenamiento hacia la Sandbox.

Por último, notar que en la Figura 4.9 se incorporan los procesos de Offloading y carga de datos desde la Archival Zone hacia las demás zonas. En ambos casos, los procesos acceden al componente *Storage* de las distintas zonas; ya que este es el componente de almacenamiento de las mismas.

4.2.5. Usuarios del sistema

Los datos contenidos en las diferentes zonas cumplen diferentes propósitos y son de interés para distintos tipos de usuarios. Por estos dos motivos es de interés definir grupos de usuarios a los cuales se le asignan permisos de acceso a distintas zonas.

Los principales usuarios de nuestros sistema son los DST y los DA.

Para cumplir con los requisitos de los DST es que se incorpora la zona *Sandbox*; ya que dentro de esta zona estos usuarios pueden realizar análisis, transformaciones y modelos sobre los datos, sin interferir con otros procesamientos de los datos (Sección 4.2.2). Igualmente, estos usuarios también pueden acceder a datos en la *Refined Zone*.

Por otro lado, los DA requieren de un mayor refinamiento de los datos para poder realizar cálculos y visualizaciones de los mismos (B. Inmon y cols., 2021), por lo que estos usuarios acceden a los datos en la *Refined Zone*, donde los datos están preparados para el uso por aplicaciones y sistemas de negocio.

Además de los DST y los DA, creemos importante identificar tres nuevos usuarios en nuestro sistema.

En primer lugar, identificamos a los **System Administrators**. Estos usuarios se encargan del mantenimiento del sistema y tienen acceso a todos los componentes de la arquitectura (Hai y cols., 2021). Algunos ejemplos de las responsabilidades de estos usuarios son: registro de nuevos usuarios dentro del sistema, asignación de permisos a usuarios, registro de nuevas fuentes, administración de la infraestructura del sistema, entre otros.

Por otro lado, debido a la importancia de la calidad de los datos dentro de estos sistemas; ya que una baja calidad de los datos hace que el sistema y los datos se perciban como poco confiables por parte de los usuarios, además de que los análisis realizados con estos datos pueden generar resultados erróneos, creemos importante incorporar un usuario dedicado a la gestión de la calidad de los datos. Por lo tanto, en segundo lugar, identificamos a los **Data Quality Expert**, quienes se encargan de garantizar la correcta gestión de la calidad de los datos en el sistema. Algunas tareas de este usuario involucran la definición del modelo de calidad, la medición de la calidad de los datos y la evaluación de las medidas obtenidas para definir procesos de mejora. Debido a los disintos procesamientos que se llevan a cabo sobre los datos, este usuario debe poder acceder a los datos en distintas etapas de procesamiento.

En tercer y último lugar, identificamos a los **Data Engineers**, quienes definen e implementan los procesamientos y transformaciones de los datos en las zonas *Landing, Trusted* y *Refined*.

En la Tabla 4.1 se puede observar un resumen del control de acceso a las distintas zonas, y en la Figura 4.8 se pueden ver los distintos usuarios en los componentes de la arquitectura. (Giebler y cols., 2020)

Para concluir esta sección, se presenta la arquitectura final en la Figura 4.9, con todos los componentes descritos en las subsecciones anteriores.

Componente de la arquitectura	**Grupo de Usuarios con acceso**
Landing Zone	System administrator, Data Engineer, Data Quality Expert
Raw Zone	System administrator, Data Quality Expert
Archival Zone	System administrator
Trusted Zone	System administrator, Data Engineer, Data Quality Expert
Refined Zone	Todos
Sandbox	Data Scientists, System administrator
Governance Metadata Zone	Data Scientists, System administrator, Data Engineer y Data Quality Expert

Tabla 4.1: Zonas y Grupos de Usuarios

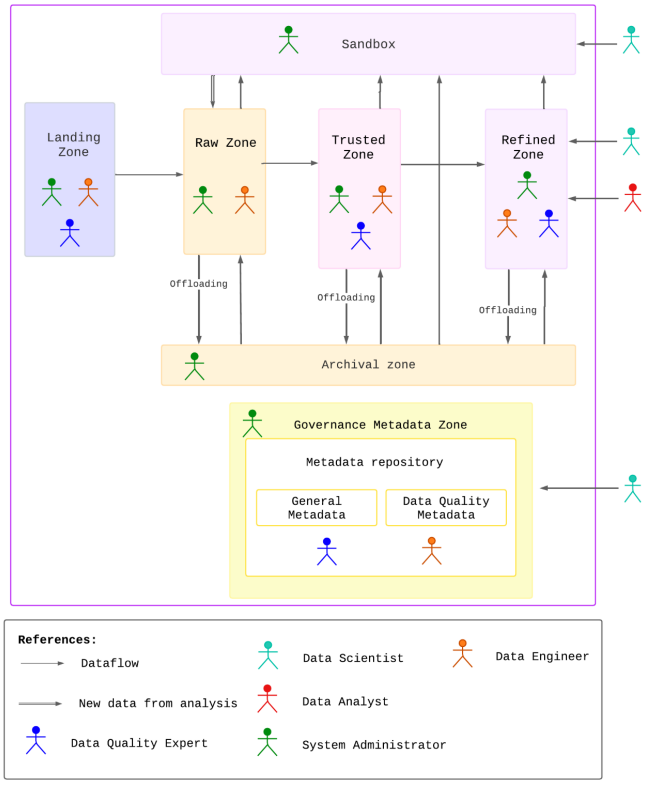

Figura 4.8: Usuarios dentro de la arquitectura

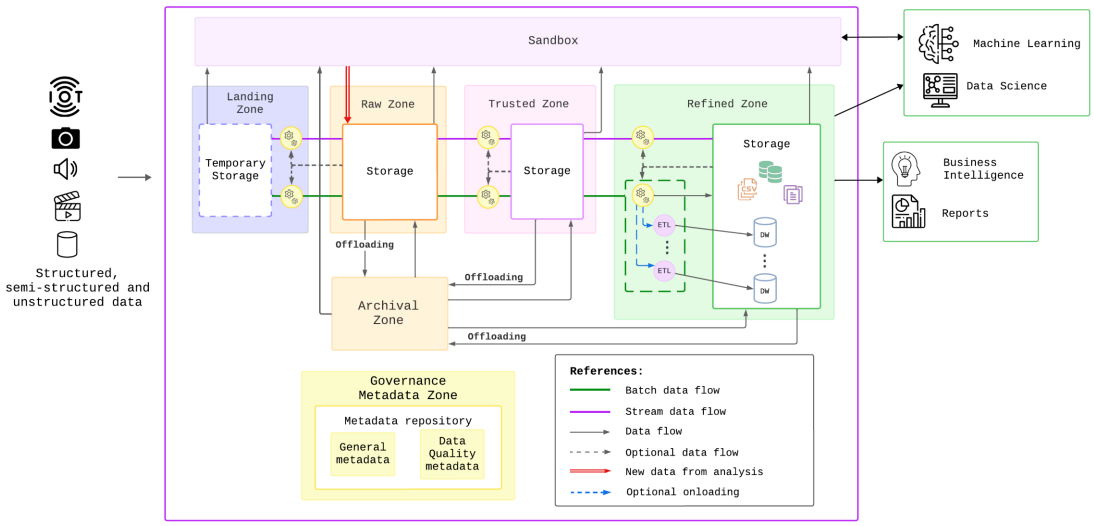

Figura 4.9: Arquitectura final.

4.3. Características de la arquitectura

En esta sección se busca verificar si la arquitectura propuesta cumple con los requerimientos especificados en la Sección 3.8 (Ver Tabla 4.2).

Por practicidad, a continuación se presentan nuevamente los requerimientos de arquitecturas de BD.

(C1): Ingestión y almacenamiento de datos de fuentes heterogéneas.

(C2): Almacenamiento de datos crudos.

(C3): Almacenamiento de grandes volúmenes de datos en sistemas de bajo costo.

(C4): Extracción, modelado y almacenamiento de metadatos (Metadata management).

(C5) Acceso a datos por parte de herramientas de BI.

(C6): Acceso a datos por parte de herramientas de AA.

(C7): Procesamiento y transformación de los datos (data integration, data cleaning, metadata enrichment).

(C8): Almacenamiento de datos procesados.

(C9): Gobierno, calidad y seguridad de datos dentro de la arquitectura.

(C10): Procesamiento de datos de stream y batch.

(C11): Transacciones ACID.

(C12): Consultas con baja latencia (Indexación, caching).

	Two-Layer	Data Pond	Multi-layered	Zaloni Architecture	(Giebler y cols., 2020)	(Herden, 2020)	(Zhao y cols., 2021)	(Oukhouya y cols., 2021)	(Hai y cols., 2023) y (Hai y cols., 2021)	(Armbrust y cols., 2021)	(Orescanin y Hlupic, 2021)	(Mazumdar y cols., 2023)	Arquitectura propuesta
(C1)	X	X	X	X	X	X	X	X	X	X	X	X	X
(C2)	X		X	X	X	X	X	X	X	X	X	X	X
(C3)	X	X	X	X	X	X	X	X	X	X	X	X	X
(C4)			X				X	X	X	X		X	X
(C5)					X			X		X	X	X	X
(C6)	X	X	X	X	X	X	X	X	X	X	X	X	X
(C7)		X	X	X	X	X	X	X	X	X	X	X	X
(C8)		X	X	X	X		X	X	X	X	X		X
(C9)							X	X					X
(C10)		X					X						X
(C11)										X	parcial	X	parcial
(C12)										X	parcial	X	parcial

Tabla 4.2: Clasificación de la arquitectura propuesta según las características presentadas en la Tabla 3.5

Los requisitos **(C1)** y **(C2)** se satisfacen con la incorporación del DL a la arquitectura. En particular, la Landing Zone permite ingerir datos (Requisito **(C1)**) y metadatos de diversas fuentes (Requisito **(C5)**), y las demás zonas del DL permiten el almacenamiento de grandes volúmenes de datos con distintos formatos (Requisito **(C1)**). A su vez, la Raw Zone permite almacenar los datos con el mismo formato que tenían en la fuente (Requisito **(C2)**). Por último, por más que el requisito **(C3)** puede ser dependiente de la implementación de la arquitectura, se afirma que la arquitectura cumple este requisito; ya que una de las características de los DLs es proporcionar almacenamiento masivo de datos en sistemas de bajo costo.

Con respecto al requisito **(C7)** y **(C8)**, las zonas Trusted y Refined permiten realizar distintos tipos de procesamientos sobre los datos, entre ellos, integración, limpieza, agregación y normalización. A su vez, los datos procesados se almacenan en estas zonas.

Por otro lado, la incorporación de la arquitectura BRAID a la arquitectura permite cumplir el requisito de dar soporte al procesamiento de datos de stream y batch (Requisito **(C10)**).

Los requisitos **(C5)** y **(C6)** se cumplen gracias a las zonas Sandbox y Refined; ya que estas permiten el acceso y consulta de datos por parte de herramientas de BI y AA.

A su vez, los requisitos **(C4)** y **(C9)** se logran cumplir con la incorporación de la zona *Governance Metadata Zone*; ya que incorpora aspectos para un correcto gobierno de datos. Notar que en esta zona se sub-divide el repositorio de metadatos, para resaltar los metadatos de calidad y de los datos. Sin embargo, se puede hacer una tercera división en caso que se desee enfocar en aspectos de seguridad de los datos.

Por otro lado, notar que la arquitectura cumple parcialmente los requisitos **(C11)** y **(C12)**; ya que estas características solo se pueden garantizar para los datos almacenados en el DW.

Por último, queremos verificar si la arquitectura cumple con las características de un DLH. A partir de los criterios establecidos en la Tabla 3.5 y de las características de DLH establecidas en la Sección 3.3.2, un DLH cumple con los criterios **(C1)**, **(C2)**, **(C3)**, **(C4)**, **(C5)**, **(C6)**, **(C7)**, **(C8)**, **(C9)**, **(C11)** y **(C12)** de las arquitecturas de BD. Por lo tanto, no se puede denominar la arquitectura como DLH; ya que **(C11)** y **(C12)** se cumplen parcialmente.

4.4. Arquitectura de Data Lakehouse

A partir de lo mencionado en la sección anterior, es claro que la arquitectura propuesta cumple todos los requisitos definidos para un DLH, salvo los requisitos **(C11)** y **(C12)**, que los cumple parcialmente.

Para que la arquitectura logre cumplir con las características **(C11)** y **(C12)**, se deben extender las funcionalidades de los DW, como transacciones ACID, facilidad de evolución de esquemas, indexado y caching, a través de las zonas que almacenan datos en el flujo principal de procesamiento del DL (*Raw*, *Trusted* y *Refined*). En la industria existen diversas tecnologías que buscan llevar estas propiedades sobre datos almacenados en DLs (Ver Sección 3.7.2). El aspecto en común de estas tecnologías es que todas definen formatos de tabla sobre formatos de archivos optimizados para fines analíticos (Ver Sección 3.7.1). Sin embargo, no basta con utilizar estos formatos de tabla, también es necesario contar con motores de consulta que permitan consultar los datos, garantizando buena performance para el acceso a los mismos.

En la Figura 4.10 se presenta la modificación de la arquitectura para que cumpla las características **(C11)** y **(C12)**.

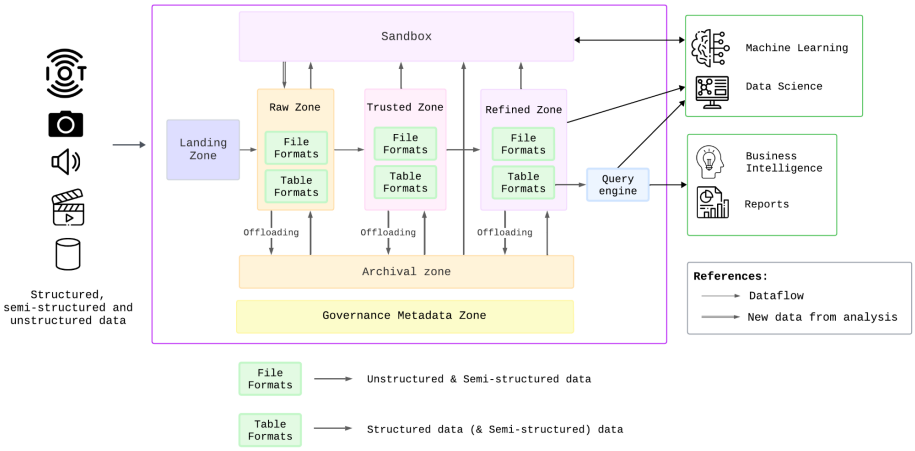

Figura 4.10: Arquitectura de DLH.

Por lo tanto, se propone utilizar formatos de tabla para almacenar aquellos datos que vayan a ser consumidos por herramientas de BI, y motores de consulta que se puedan integrar con estos formatos. Los demás datos, no-estructurados y semi-estructurados, se siguen almacenando utilizando formatos abiertos de archivos, en las distintas zonas del DL. Notar que también se podrían usar estos formatos de tabla para almacenar datos semi-estructurados, cuyo caso de uso final se pueda ver beneficiado por las propiedades de estos formatos.

Por último, notar que a partir de estas modificaciones, la nueva arquitectura cumple todas las características de DLH.

Capítulo 5

Gobierno de datos en la arquitectura de Big Data propuesta

En este capítulo se presentan distintos aspectos para poder llevar a cabo un gobierno de datos en la arquitectura de BD propuesta. Para ello, en la primera sección se puede ver la incorporación de algunas tareas del proceso de gestión de la calidad, descrito en la Sección 2.5, en la arquitectura, así como un modelo de metadatos de calidad para la misma. Por otro lado, en la segunda sección se presenta un modelo de metadatos de datos y procesos definido para la arquitectura, mientras que en la tercera sección se presenta la integración del modelo de metadatos de calidad con el de datos y procesos. Por último, en la cuarta sección se describe el diseño de la base de metadatos del gobierno de datos.

5.1. Gestión de calidad de datos en la arquitectura

Para incorporar la GCD en la arquitectura se tomó como base el proceso de Gestión de Calidad de Datos en Sistemas de Información presentado en la Sección 2.5. Luego, se identificaron qué etapas se realizan dentro de la arquitectura. En particular, algunas de estas etapas se pueden agrupar en los conjuntos, *Medición y Evaluación* y *Mejora* (Ver Figura 5.1).

Notar que las etapas "Análisis de procesos de negocio involucrados", "Data profiling", "Modelo de calidad" y "Estimación de la calidad" se realizan por fuera del sistema.

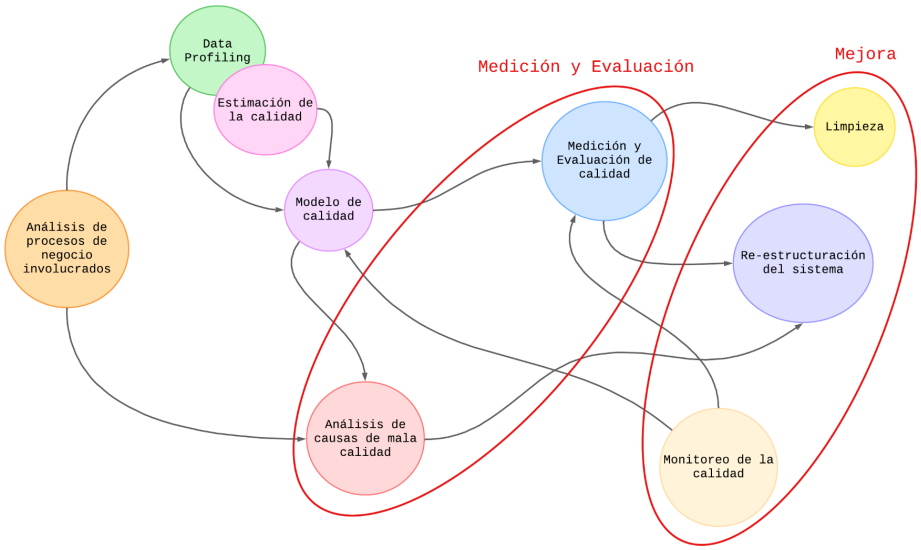

Figura 5.1: Proceso de Gestión de calidad de Datos en Sistemas de Información con etapas agrupadas.

5.1.1. Integración del proceso en la arquitectura

En la Figura 5.2 se presenta la integración del proceso de GCD dentro de la arquitectura propuesta, donde se eliminaron algunos flujos de datos para mejorar la comprensión del diagrama. Este proceso busca medir la calidad de los datos, además de llevar a cabo procesos de mejora para mejorar la calidad de los datos.

La medición de calidad se realiza en los procesamientos de cada zona (círculo rosado), a partir de la cual se obtienen las medidas (flecha punteada azul) y se generan los metadatos de calidad (cículo azul), que después se envían a la Governance Metadata Zone (flecha rosada) para ser almacenados. Por último, las mejoras de los procesos se representan con un recuadro rojo sobre los distintos procesamientos en las zonas. Estas mejoras se derivan de la medición de calidad.

Notar que se pueden realizar mediciones de calidad antes de cada procesamiento realizado sobre los datos; ya que que las mediciones de calidad permiten inferir procesos de corrección sobre los mismos, pero también se ejecutan nuevas mediciones luego de estos procesos, para identificar si estas transformaciones logran mejorar la calidad. Por otro lado, también es importante medir la calidad de los datos luego de realizadas distintas transformaciones; ya que estas pueden introducir nuevos errores de calidad en los datos.

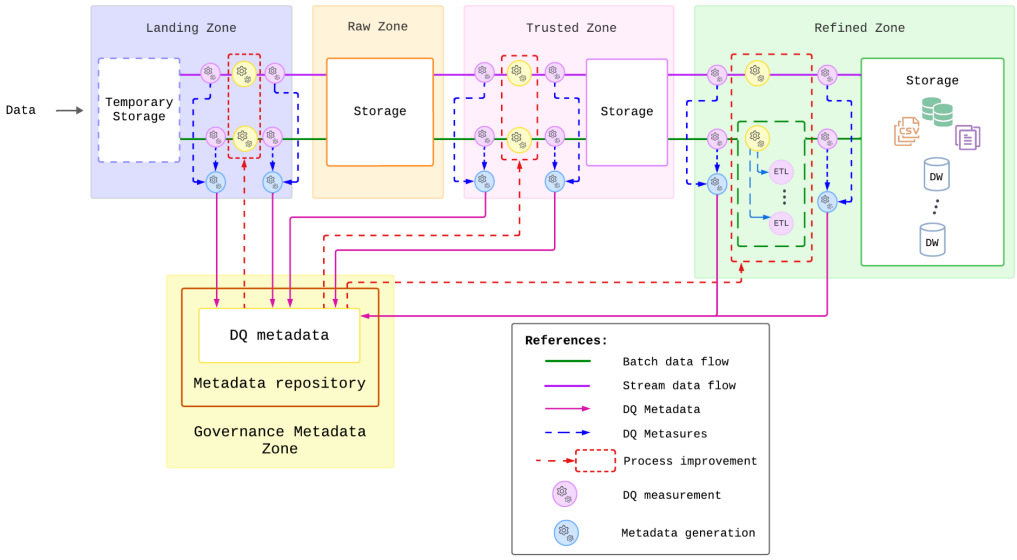

Figura 5.2: Proceso de gestión de calidad en la arquitectura propuesta.

5.1.2. Modelo de metadatos de calidad

Para llevar a cabo la GCD, se deben llevar a cabo distintas etapas definidas en la Sección 2.5. Dentro de la etapa *Medición y Evaluación de calidad* se debe definir el modelo de metadatos de calidad, donde se almacenan las medidas de calidad tomadas sobre los datos e información asociada a la medida (por ejemplo, métrica, factor, entre otros).

El modelo de metadatos de calidad (Ver Figura 5.3) en el cual nos vamos a basar está orientado al almacenamiento de metadatos de calidad para datasets estructurados. Sin embargo, debido a que la arquitectura debe almacenar distintas estructuras de datos (estructurados, semi-estructurados y no-estructurados), se debió adaptar el mismo para cumplir con este requisito. Por otro lado, se simplificó el modelo para considerar un solo método de medición para cada métrica. En caso que se tengan más, se debe adaptar el modelo, agregando una nueva entidad *Method* asociada a *Metric* con una relación N a 1, y la entidad *Measure* se asocia a *Method* con una relación N a 1.

RNEs para modelo de metadatos de calidad de la Figura 5.3:

1. La granularidad de la métrica debe corresponderse con la granularidad de

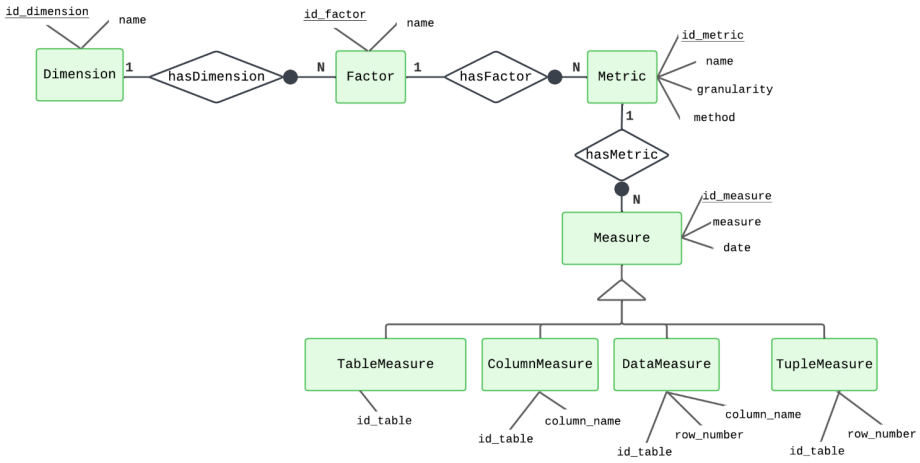

Figura 5.3: Modelo de metadatos de calidad

la medida.

$(\forall m \in \text{Metric})(\forall tm \in \text{TableMeasure})$
$(< m, tm > \in \text{hasMetric} \rightarrow \text{granularity}(m) = \text{Table})$
$(\forall m \in \text{Metric})(\forall dm \in \text{DataMeasure})$
$(< m, dm > \in \text{hasMetric} \rightarrow \text{granularity}(m) = \text{Data})$
$(\forall m \in \text{Metric})(\forall cm \in \text{ColumnMeasure})$
$(< m, cm > \in \text{hasMetric} \rightarrow \text{granularity}(m) = \text{Column})$
$(\forall m \in \text{Metric})(\forall tm \in \text{TupleMeasure})$
$(< m, tm > \in \text{hasMetric} \rightarrow \text{granularity}(m) = \text{Tuple})$

2. La categorización Measure es total y disjunta.

$$\text{Measure} = \text{TableMeasure} \cup \text{DataMeasure} \cup \text{ColumnMeasure} \cup \text{TupleMeasure}$$
$$\text{TableMeasure} \cap \text{DataMeasure} \cap \text{ColumnMeasure} \cap \text{TupleMeasure} = \emptyset$$

El modelo de metadatos de calidad modela las relaciones entre las dimensiones, factores y métricas de calidad, así como las medidas de calidad para datasets estructurados (tablas con columnas). En la tabla 5.1 se describen las distintas entidades y relaciones del modelo.

Entidad	**Descripción**	**Atributos**
Dimension	Dimensión de calidad.	**id_dimension:** Identificador **name:** Nombre

Factor	Factor de calidad. Está asociado a una dimensión de calidad a través de la relación "hasDimension". Una dimensión puede tener varios factores asociados.	**id_factor:** Identificador **name:** Nombre
Metric	Métrica de calidad. Está asociada a un factor de calidad a través de la relación "hasFactor". Un factor puede tener varias métricas asociadas.	**id_metric:** Identificador **name:** Nombre **granularity:** Granularidad **method:** Método.
Measure	Medida de calidad. Está asociada a una métrica de calidad a través de la relación "hasMetric". Una Métrica puede tener varias medidas asociadas.	**id_measure:** Identificador **measure:** Medida **date:** Fecha
Table Measure	Medida con granularidad tabla.	**id_table**: Identificador de la tabla.
Column Measure	Medida con granularidad columna.	**column_name:** Nombre de la columna en la tabla. **id_table**: Identificador de la tabla a la que pertenece la columna.
Data Measure	Medida con granularidad celda.	**id_table:** Identificador de la tabla donde se encuentra el dato. **column_name:** Nombre de la columna a la que pertenece el dato **row_number:** Número de fila donde se encuentra el dato.
Tuple Measure	Medida con granularidad tupla.	**id_table:** Identificador de la tabla donde se encuentra la tupla. **row_number:** Número de fila.

Tabla 5.1: Entidades del modelo de metadatos de calidad

5.2. Modelo de metadatos de datos y procesos

Debido a la diversidad de estructuras de datos en la arquitectura, se pueden tomar medidas de calidad sobre distintos componentes de las estructuras de los datos. A su vez, como se mencionó anteriormente, las medidas tomadas sobre los datos pueden generar distintos procesos de corrección y transformación de los datos para mejorar la calidad de los mismos. Por lo tanto, esta sección presenta una primera aproximación de un modelo de metadatos que refleje las distintas estructuras de datos y los procesos llevados a cabo sobre estos.

Como se mencionó en el capítulo anterior, los metadatos del sistema se almacenan en la Governance Metadata Zone y en la Sección 4.2.2 se definieron algunos aspectos que debería incorporar un modelo de metadatos para la arquitectura propuesta. Por lo tanto, en el modelo propuesto también se incorporan algunos de estos aspectos.

Para modelar los metadatos de los datasets y procesos se toma como base el modelo de metadatos presentado en (Megdiche y cols., 2021) que se puede ver en la Figura 5.4. Se elige este modelo; ya que es el único modelo, de los analizados en la Sección 3.6, que incorpora el modelado de metadatos asociados a procesos llevados a cabo sobre los datos.

Este modelo se simplificó para considerar solamente aspectos de los datasets y procesos (Ver Figura 5.5). Para cumplir con los metadatos establecidos en la Sección 4.2.2, se incorpora la entidad "Zone", para reflejar el almacenamiento de datasets y la ejecución de los procesos en las distintas zonas. Notar que no se hace referencia al almacenamiento de datasets en DW, debido a que estos se encuentran dentro de la Refined Zone. Por lo tanto, los metadatos de un DW van a estar asociados a la instancia de "Zone" con *name = "refined-zone"*. Además, se modela el acceso a datasets por parte de usuarios a través de aplicaciones (Ravat y Zhao, 2019b; Megdiche y cols., 2021). Notar que no se modelan los metadatos asociados a la evolución de los datasets, y tampoco se modelan los metadatos asociados a análisis realizados por parte de DST o expertos de ML, establecidos en la Sección 4.2.2; ya que no es el foco de este trabajo.

Se debió modelar la granularidad de las entidades y atributos asociados a los datasets estructurados y semi-estructurados. Notar que esto es un aspecto que los trabajos (Oukhouya y cols., 2021) y (Oukhouya y cols., 2021) argumentan que hace falta en el modelo de la Figura 5.4. Por lo tanto, se considera un dataset estructurado a una tabla compuesta por un conjunto de columnas. Por otro lado, se considera un dataset semi-estructurado a una colección (*Collection*) de documentos (*Documents*), los cuales poseen un conjunto de campos (*Fields*). Los campos pueden ser de tipo atómico (por ejemplo, numérico, de texto, entre otros), un documento o un array (Etcheverry, s.f.; Cristalli, Serra, y Marotta, 2018). Por último, se considera un dataset no-estructurado a un conjunto de elementos.

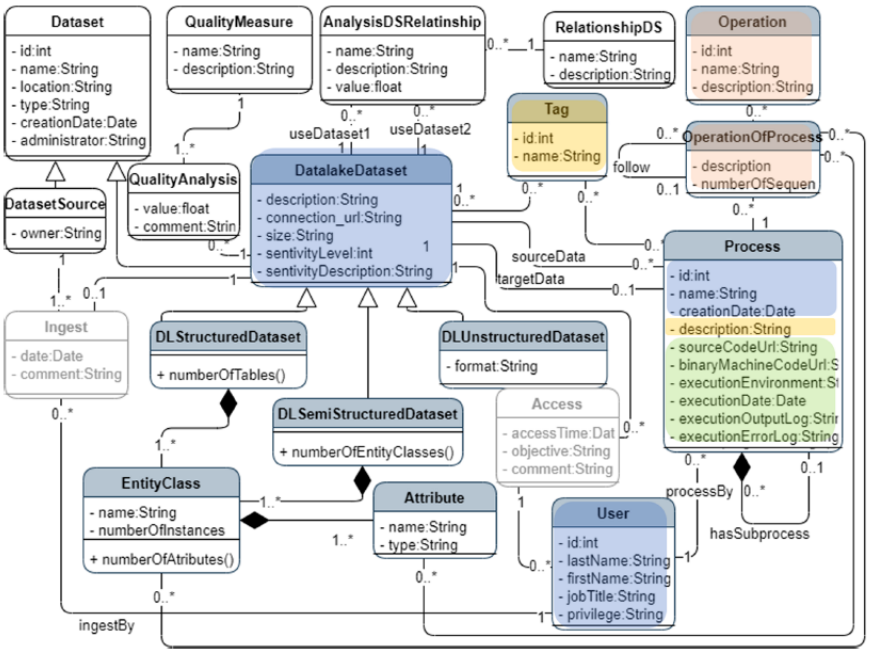

Figura 5.4: Modelo de metadatos presentado en (Megdiche y cols., 2021)

En la tabla 5.2 se describen las distintas entidades del modelo de la Figura 5.5.

Entidad	**Descripción**	**Atributos**
Process	Proceso llevado a cabo en la arquitectura. Puede tener varios datasets como fuente (relación "source") y resultado (relación "target"), pero cada dataset resultado está asociado a un único proceso. A su vez, un proceso se ejecuta en una zona particular de la arquitectura (relación "executedIn").	**id_process:** Identificador **name:** Nombre **description:** Descripción **date:** Fecha de ejecución.
Zone	Zona de la arquitectura.	**id_zone:** Identificador **name:** Nombre

Dataset	Dataset. Puede estar almacenado en una única zona (relación *storedIn*). A su vez, se agrega la relación *lastStorageZone*, para los procesos de Offloading de la Archival Zone. También se pueden representar relaciones entre datasets a través de la relación "related", donde se guarda la descripción de la relación en el atributo **description**.	**id_dataset**: Identificador **name**: Nombre **description**: Descripción **ingestion_date**: Fecha de ingestión **ingestion_info:** Información de la fuente **last_update:** Última fecha de actualización **type:** Tipo de dato almacenado (Por ejemplo: imágenes, videos, archivo excel, archivo csv, entre otros)
Table	Tabla. Puede estar asociado a varias Columnas (relación "belongsToTable") y cada Columna está asociado a una única tabla.	
Column	Columna de una tabla.	**id_column:** Identificador. **column_name:** Nombre **column_type:** Tipo de dato
Collection	Colección. Puede estar asociado a varios Documentos (relación "belongsToCollection") y cada Documento está asociado a como máximo una Colección.	
Unstructured	Dataset no-estructurado. Puede estar asociado a un conjunto de Elementos (relación "belongsToUnstructured") y cada Elemento está asociado a un único dataset no-estructurado	
Element	Elemento de un dataset no-estructurado	**id_element**: Identificador **format**: Formato de archivo (Ejemplo: .png, .jpeg, .mp4, etc.)
Document	Documento. Puede estar asociado a varios Campos (relación "belongsToDoc").	**id_document:** Identificador.

Field	Campo. Cada campo está asociado a un único Documento (relación "belongsToDoc"). Si el campo tiene tipo *document* va a estar a asociado a un documento, y si es de tipo *array* puede estar asociado a más de un documento, en ambos casos a través de la relación "belongsToField".	**id_field:** Identificador. **name**: Nombre. **type**: Tipo de dato.
User	Usuario del sistema.	**id_user:** Identificador. **name:** Nombre. **role:** Rol del usuario.
Application	Aplicación con la cual un usuario accede a datos en la *Refined Zone*. Esto se representa con la relación "access", donde se almacenan las distintas fechas de acceso (atributo multivaluado *date*).	**id_application:** Identificador **name:** Nombre **description:** Descripción.

Tabla 5.2: Entidades del modelo de metadatos

RNEs del modelo de la Figura 5.5:

1. Los procesos ejecutados en las zonas *Trusted* o *Refined* almacenan el resultado en dichas zonas.

$$(\forall p \in \text{Process})(\forall z \in \text{Zone})(\forall d \in \text{Dataset})$$
$$(< p, z > \in \text{executedIn} \wedge \text{name(z)} = \text{'trusted-zone'} \wedge$$
$$< p, d > \in \text{target} \rightarrow < d, z > \text{storedIn})$$

$$(\forall p \in \text{Process})(\forall z \in \text{Zone})(\forall d \in \text{Dataset})$$
$$(< p, z > \in \text{executedIn} \wedge \text{name(z)} = \text{'refined-zone'} \wedge$$
$$< p, d > \in \text{target} \rightarrow < d, z > \text{storedIn})$$

2. El atributo *role* de la entidad *User* puede tomar valores en el conjunto {system_admin, data_engineer, data_scientist, data_analyst, data_quality_expert}

$$(\forall u \in \text{User})(\text{role}(u) \in \{system\_admin, data\_engineer, data\_scientist,$$
$$data\_analyst, data\_quality\_expert\})$$

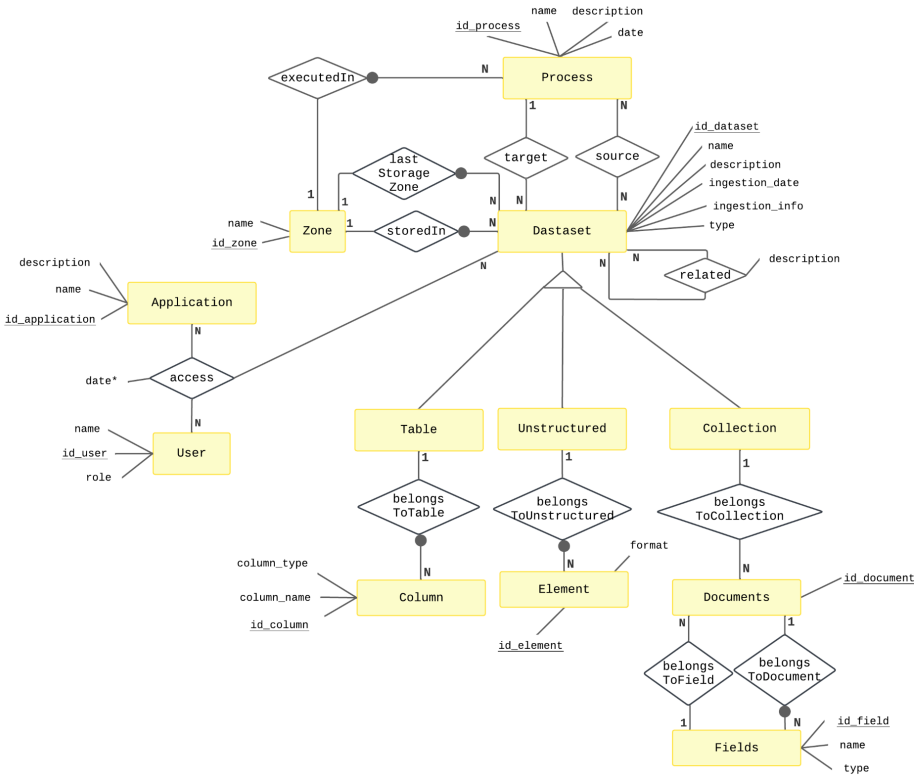

Figura 5.5: Modelo de metadatos simplificado

3. El atributo *type* de la entidad *Fields* puede tomar valores en el conjunto {string, number, boolean, document, array, null}.

$$(\forall f \in \text{Fields})(\text{type}(f) \in \{string, number, boolean, document, array, null\})$$

4. Solo los *Fields* con *type* $\in$ {document, array} pueden asociarse a un *Document* a través de la relación *belongsToField*.

$$(\forall f \in \text{Fields})(\forall d \in \text{Document})(< \text{d, f} > \in \text{belongsToField} \rightarrow \text{type(f)} \in \{\text{array, document}\})$$

5. Los *Application* y *Users* solo se pueden asociar a *Datasets*, a través de la relación "access", almacenados en la Refined Zone.

$$(\forall a \in \text{Application})(\forall u \in \text{User})(\forall d \in \text{Dataset})(\forall z \in \text{Zone})$$
$$(< a, u, d > \in \text{access} \wedge < d, z > \in \text{storedIn} \rightarrow \text{name(z)} = \text{'refined-zone'})$$

6. La categorización *Dataset* es total y disjunta.

$$\text{Dataset} = \text{Table} \cup \text{Collection} \cup \text{Unstructured}$$
$$\text{Table} \cap \text{Collection} \cap \text{Unstructured} = \emptyset$$

5.3. Modelo de metadatos de gobierno de la arquitectura

En esta sección se presenta la integración del modelo de metadatos de calidad con el modelo de metadatos de datos y procesos, para formar el modelo completo de gobierno de datos.

Para llevar a cabo esta integración, se debió adaptar el modelo de la Figura 5.3 para incluir el almacenamiento de medidas tomadas sobre datasets semi-estructurados y no-estructurados. Debido a que este modelo representa las distintas estructuras de los datasets, no es necesario tener la categorización "Measure"; ya que basta con mantener esta entidad y tener relaciones para los distintos tipos de medidas que se pueden tomar, asociados al componente de la estructura del dataset correspondiente.

En el caso de los datasets estructurados, las medidas que se pueden tomar son a nivel de dataset (tabla) (relación *DatasetMeasure*), columna (relación *ColumnMeasure*), tupla (relación *TupleMeasure*) o dato (relación *DataMeasure*). Para los datasets no-estructurados se pueden tomar medidas a nivel del dataset (relación *DatasetMeasure*) o a nivel de los elementos (relación *ElementMeasure*). Para definir las medidas que se pueden tomar sobre un dataset semi-estructurado, se tuvo en cuenta lo definido en el trabajo (Cristalli y cols., 2018). Los autores de este trabajo indican que se pueden tomar medidas a nivel de la colección (relación *DatasetMeasure*), documento (relación *DocMeasure*), campos en todos los documentos (relación *FieldMeasure*), valor de un campo (relación *FieldValueMeasure*) y elemento de un array (relación *ArrayValueMeasure*).

En la Tabla 5.3 se puede ver un resumen de las relaciones asociadas a la medición de calidad.

Por otro lado, se agregó la relación "derives", entre un *Process* y una *Measure*, la cual permite representar cuando una medida de calidad genera un proceso de mejora. También se agregó la relación "measuredIn", entre una "Measure" y una "Zone", la cual permite representar en qué zona se tomó la medida de calidad. Notar que en los casos donde se toman medidas en la Landing Zone, la zona en la cual se toma la medida difiere de la zona donde se va a almacenar el dato (Raw Zone).

Para finalizar, en la Figura 5.6 se presenta la unificación de los 2 modelos de las Figuras 5.3 y 5.5, con las modificaciones mencionadas anteriormente.

Relación	Descripción	Atributos
DatasetMeasure	Medida tomada sobre un dataset.	No tiene.
DataMeasure	Medida tomada sobre un dato de un dataset estructurado.	**row_number:** Número de fila en la cual se encuentra el dato en la tabla.
ColumnMeasure	Medida tomada sobre una columna de un dataset estructurado.	No tiene.
TupleMeasure	Medida tomada sobre una tupla (fila) de un dataset estructurado.	**row_number:** Número de fila en la tabla.
DocMeasure	Medida tomada sobre un documento.	No tiene.
FieldValueMeasure	Medida tomada sobre el valor atómico de un campo.	No tiene.
FieldMeasure	Medida tomada sobre un campo en todos los documentos.	No tiene.
ArrayValueMeasure	Medida tomada sobre un valor de un array.	**value_index:** Índice del dato en el array.
ElementMeasure	Medida tomada sobre un elemento de un dataset no-estructurado	No tiene.

Tabla 5.3: Relaciones de tipos de medidas.

RNEs para modelo de metadatos de la Figura 5.6:

1. Los procesos ejecutados en las zonas *Trusted* o *Refined* almacenan el resultado en dichas zonas.

$$(\forall p \in \text{Process})(\forall z \in \text{Zone})(\forall d \in \text{Dataset})$$
$$(< p, z > \in \text{executedIn} \wedge \text{name(z)} = \text{'trusted-zone'} \wedge$$
$$< p, d > \in \text{target} \rightarrow < d, z > \text{storedIn})$$

$$(\forall p \in \text{Process})(\forall z \in \text{Zone})(\forall d \in \text{Dataset})$$
$$(< p, z > \in \text{executedIn} \wedge \text{name(z)} = \text{'refined-zone'} \wedge$$
$$< p, d > \in \text{target} \rightarrow < d, z > \text{storedIn})$$

2. El atributo *role* de la entidad *User* puede tomar valores en el conjunto

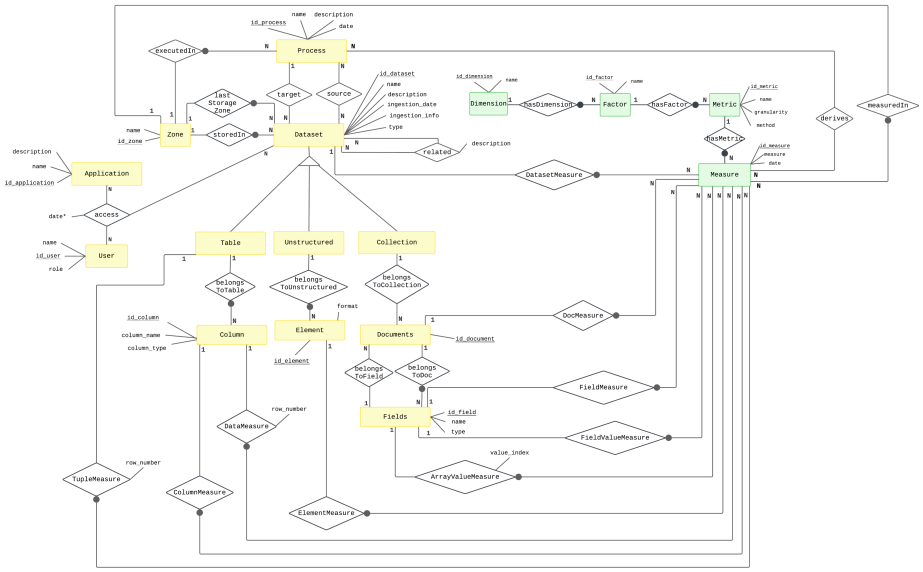

Figura 5.6: Modelo entidad-relación de metadatos.

{system_admin, data_engineer, data_scientist, data_analyst, data_quality_expert}

$$(\forall u \in \text{User})(\text{role}(u) \in \{system\_admin, data\_engineer, data\_scientist, data\_analyst, data\_quality\_expert\})$$

3. El atributo *type* de la entidad *Fields* puede tomar valores en el conjunto {string, number, boolean, document, array, null}.

$$(\forall f \in \text{Fields})(\text{type}(f) \in \{string, number, boolean, document, array, null\})$$

4. Solo los *Fields* con *type* $\in$ {document, array} pueden asociarse a un *Document* a través de la relación *belongsToField*.

$$(\forall f \in \text{Fields})(\forall d \in \text{Document})(< \text{d, f} > \in \text{belongsToField} \rightarrow \text{type(f)} \in \{\text{array, document}\})$$

5. Todo *Field* asociado a una *Measure* a través de la relación "ArayValue-Measure" tiene que tener *type* $\in$ {array}.

$$(\forall f \in \text{Fields})(\forall m \in \text{Measure})(< \text{m, f} > \in \text{ArrayValueMeasure} \rightarrow \text{type(f)} \in \{\text{array}\})$$

6. Los *Application* y *Users* solo se pueden asociar a *Datasets*, a través de la relación "access", almacenados en la Refined Zone.

$$(\forall a \in \text{Application})(\forall u \in \text{User})(\forall d \in \text{Dataset})(\forall z \in \text{Zone})$$
$$(< a, u, d > \in \text{access} \wedge < d, z > \in \text{storedIn} \rightarrow \text{name(z)} = \text{'refined-zone'})$$

7. La categorización *Dataset* es total y disjunta.

$$\text{Dataset} = \text{Table} \cup \text{Collection} \cup \text{Unstructured}$$
$$\text{Table} \cap \text{Collection} \cap \text{Unstructured} = \emptyset$$

5.4. Diseño de la base de metadatos de gobierno de datos

Se opta por modelar los metadatos siguiendo el modelo de datos de grafo. Esta decisión se toma porque estos brindan mayor flexibilidad de esquemas; ya que permiten agregar propiedades, etiquetas y relaciones fácilmente sin modificar otros datos, mientras que en una base de datos relacional esto implicaría modificar esquemas, afectando datos ya almacenados. También proporcionan capacidades de consultas más avanzadas, como pattern matching, camino más corto y consultas de nodos vecinos, entre otros (Etcheverry, s.f.).

Por lo tanto, se hace el pasaje del modelo entidad-relación de la Figura 5.6 a un modelo de grafos.

Para realizar el pasaje, se realizan las siguientes tareas (Etcheverry, s.f.).

1. Se crea un nodo para cada entidad.
2. Las relaciones binarias se llevan a relaciones (aristas) entre nodos.
3. Los atributos de las entidades se definen como propiedades en los nodos.
4. Los atributos de las relaciones se definen como propiedades en las aristas.
5. Para el caso de las relaciones asociadas a 3 entidades o más, se crea un nodo intermedio para la relación, además de los nodos asociados a las entidades de la relación.

Se tomó como convención que todas las relaciones N a 1 se representen como relaciones entre nodos, donde la dirección de la arista apunta a la entidad del lado 1.

En la Figura 5.7 se puede ver el modelo de metadatos y en la Tabla 5.4 se pueden ver las propiedades de cada nodo, mientras que en la Tabla 5.5 se pueden ver las propiedades de las aristas.

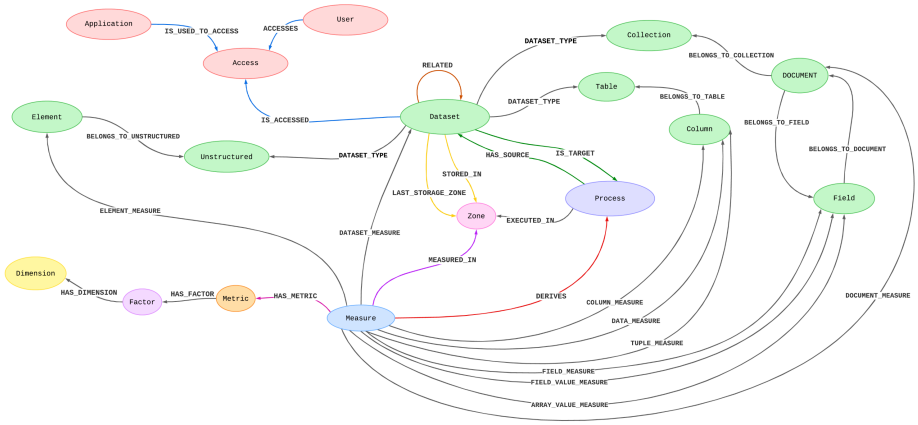

Figura 5.7: Modelo de metadatos final (sin unificar nodos).

Etiqueta del nodo	**Propiedades**
Dimension	id, name
Factor	id, name
Metric	id, name, granularity, method
Zone	id, name
Measure	id, measure, date
Dataset	id, name, description, ingestion_date, ingestion_info, type, last_update
Table	id
Column	id, name, type
Unstructured	id
Element	id, format
Collection	id
Document	id
Field	id, name, type
Process	id, name, description, date
Application	id, name, description
User	id, name, role
Access	date

Tabla 5.4: Propiedades de nodos del modelo de metadatos sin unificar nodos.

Considerando que los modelos de grafo de tipo *property graphs* permiten asignar varias etiquetas a un mismo nodo (Howard, 2024), para evitar tener nodos con 1 atributo (categorización *Dataset*), se unifican los nodos padres con los nodos hijos de la categorización. En las Figuras 5.8, 5.9 y 5.10 se muestra el modelo de metadatos para datasets no-estructurados, estructurados y semi-estructurados, respectivamente. Debido a la eliminación del nodo "Dataset", las

Etiqueta de la arista	Propiedades
RELATED	description
ARRAY_VALUE_MEASURE	value_index
TUPLE_MEASURE	row_number
DATA_MEASURE	row_number

Tabla 5.5: Propiedades de aristas del modelo de metadatos sin unificar nodos.

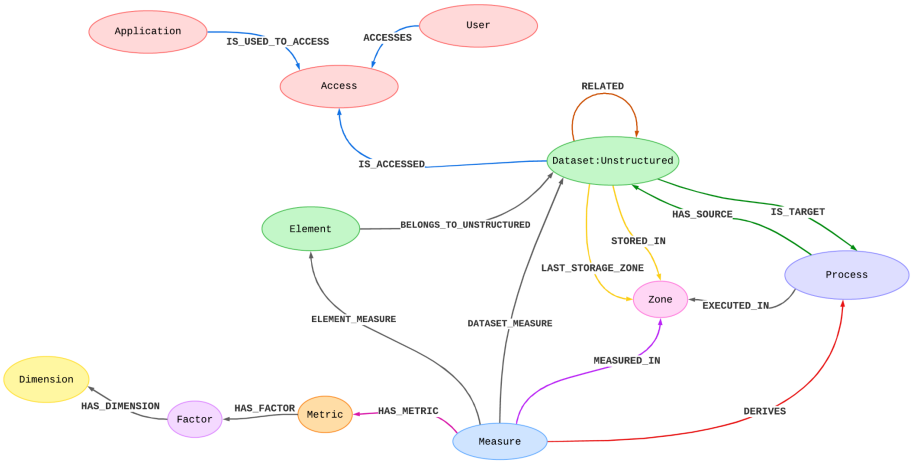

Figura 5.8: Modelo de metadatos para dataset no-estructurado.

propiedades de este nodo se incorporan en los nodos asociados a las entidades hijas (Ver Tabla 5.6). Por último, las relaciones del nodo padre ("Dataset") se asocian a los nodos asociados a las entidades hijas.

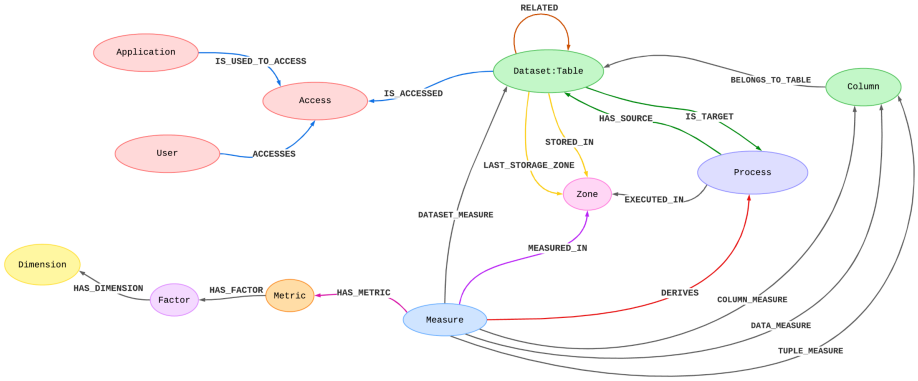

Figura 5.9: Modelo de metadatos para dataset estructurado.

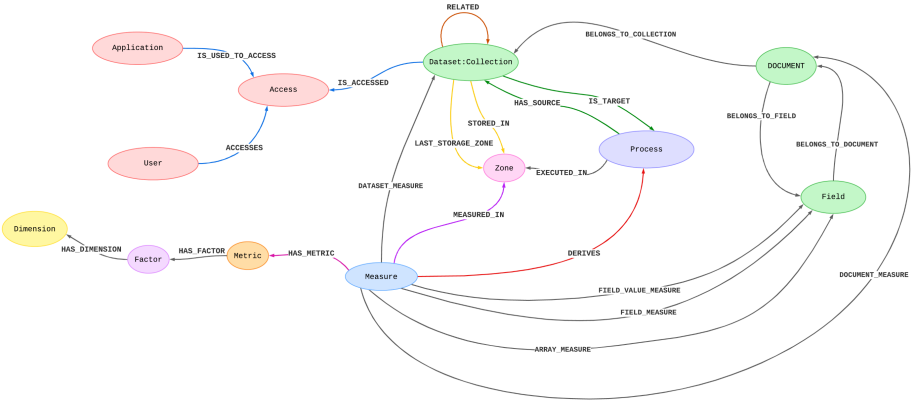

Figura 5.10: Modelo de metadatos para dataset semi-estructurado.

Etiqueta del nodo	**Propiedades**
Dimension	id, name
Factor	id, name
Metric	id, name, granularity, method
Zone	id, name
Measure	id, measure, date
Dataset:Table	id, name, description, ingestion_date, ingestion_info, type
Column	id, name, type
Dataset:Unstructured	id, name, description, ingestion_date, ingestion_info, type
Element	id, format
Dataset:Collection	id, name, description, ingestion_date, ingestion_info, type
Document	id
Field	id, name, type
Process	id, name, description, date
Application	id, name, description
User	id, name, role
Access	date

Tabla 5.6: Propiedades de nodos del modelo de metadatos final.

Capítulo 6

Experimentación

En este capítulo se describe la implementación de un prototipo de la arquitectura propuesta. Se decidió desarrollar un flujo de datos transversal a toda la arquitectura, pero se limitó el alcance considerando solamente el procesamiento batch de los datos y solo el flujo principal de los datos, sin los procesos de Offloading hacia la *Archival Zone*. En la Figura 6.1 se puede observar la arquitectura a implementar.

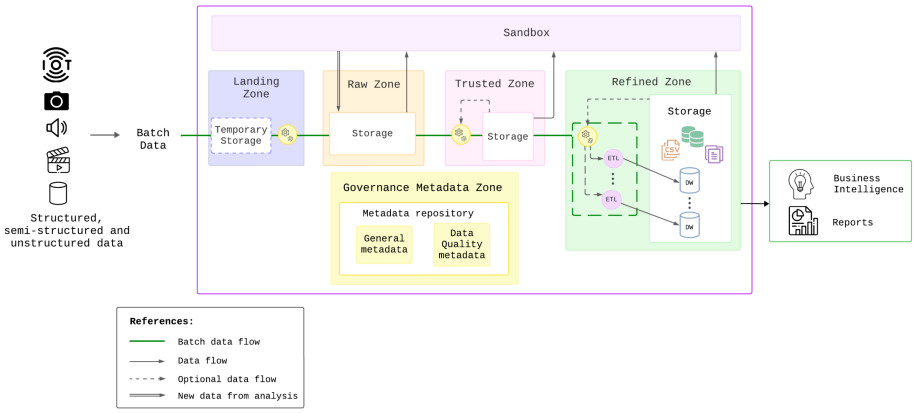

Figura 6.1: Arquitectura a implementar

Este capítulo se organiza en 6 secciones. La primera sección describe las tecnologías y los datos utilizados para la implementación de la arquitectura. La segunda sección describe un caso de uso ficticio, a partir del cual se identifican los requerimientos de análisis de los datos. Por otro lado, la tercer sección describe el modelo multidimensional de los datos para cumplir con los requisitos de análisis establecidos en la segunda sección. Luego, la cuarta, quinta y sexta sección describen la implementación de la arquitectura, describiendo los procesamientos

de los datos, la gestión de calidad y la visualización final de los mismos.

6.1. Tecnologías y datasets

En esta sección se describen las tecnologías y conjuntos de datos a utilizar.

A continuación se describen brevemente las tecnologías, y en la Figura 6.2 se puede observar cómo se distribuyen estas en los distintos componentes de la arquitectura.

1. Para el almacenamiento de datos en las zonas *Landing*, *Raw*, *Trusted* y *Refined* se eligió utilizar el sistema de archivos **Hadoop Distributed File System** (en adelante, HDFS) (Ver Anexo D.2). Este sistema de almacenamiento es muy utilizado en la comunidad académica, como por ejemplo en los trabajos (Giebler y cols., 2020, s.f.), pero también es utilizado por diversas plataformas de proveedores.

2. Para el procesamiento de los datos se optó por usar **Apache Spark**. En particular, se utilizó la librería **PySpark** del lenguaje Python. Este motor de procesamiento permite realizar procesamientos sobre grandes volúmenes de datos, tanto de stream como batch, y también cuenta con librerías para consultar datos con SQL (Spark SQL) y para trabajar con algoritmos de aprendizaje automático (MLlib) (Ver Anexo D.3).

3. Para el almacenamiento del DW se utilizó una base de datos relacional **PostgreSQL**.

4. Para el almacenamiento de metadatos, en la Governance Metadata Zone, se utilizó una base de datos de grafos en **Neo4j**.

5. Para implementar la zona Sandbox, se utilizó **Jupyter Notebook**, junto con **PySpark** para permitir el acceso a datos de las distintas zonas.

Una vez definido el stack de tecnologías a utilizar, se definió el conjunto de datos sobre el cuál se iba a trabajar. Para ello, se buscaron datasets en el Catálogo Nacional de Datos Abiertos de Uruguay. Para la selección de datasets se definieron como requisitos que estos tengan distintos formatos de almacenamiento, que el volumen de datos sea relativamente grande y que exista alguna oportunidad de integración de datos entre los datasets. A partir de la búsqueda se obtuvieron datasets correspondientes al acceso a plataformas educativas del Centro de innovación educativa con tecnologías digitales de Uruguay (en adelante, CEIBAL) por parte de estudiantes y datasets con información de los centros educativos de la Administración Nacional de Educación Pública de Uruguay (en adelante, ANEP).

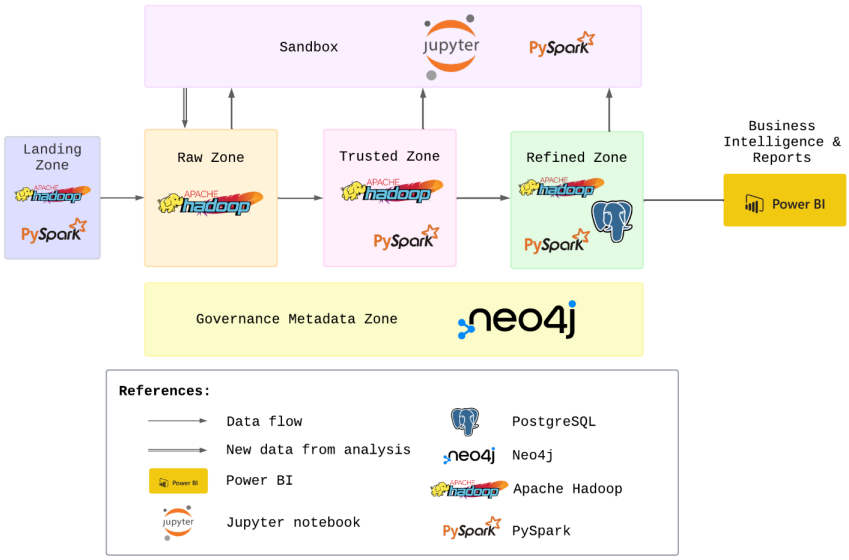

Figura 6.2: Tecnologías en los componentes de la arquitectura

En la tabla 6.1 se puede ver un resumen de la información de los datasets hallados.

Dataset	**Formato**	**Tamaño**	**Año**
datos_estudiantes_año con año $\in \{2019, 2020, 2021, 2022\}$	*.csv*	$\approx$ 70 MB	2019 a 2022
CEIP - Centros educativos de Educación Inicial y Primaria	*.xlsx*	393 KB	2022
CES - Centros educativos de Educación Secundaria	*.xlsx*	412 KB	2022

Tabla 6.1: Datasets del Catálogo de Datos Abiertos

En el caso de los datasets de accesos a plataformas por parte de estudiantes, todos tienen los mismos atributos debido a que almacenan los mismos datos pero para distintos años, por lo que se puede integrar esta información en un mismo dataset.

Por otro lado, para el caso de los datasets de centros educativos, se pudieron identificar atributos en común que permiten integrar esta información en un mismo dataset.

En adelante, se hace referencia a los datasets de accesos a plataformas por parte de estudiantes como *datos_estudiantes_año*, y a los datasets de centros

educativos de inicial y primaria, y secundaria como *CEIP* y *CES*, respectivamente.

6.2. Caso de uso

Una vez seleccionados los datasets y las tecnologías a utilizar, se definió un caso de uso, planteado en un escenario ficticio, donde se establecen los requerimientos de análisis de los datos, como se describe a continuación.

La ANEP es el organismo estatal encargado de la gestión del Sistema Educativo Público en los niveles de educación Inicial, Primaria, Media, Técnico–tecnológica (Media y Terciaria) y Formación en Educación en todo el territorio uruguayo. A su vez, en el año 2007, a partir de un decreto presidencial se crea el Plan CEIBAL. Este centro llevó a cabo la instalación de conectividad a internet en las escuelas y la distribución de dispositivos a los estudiantes de educación pública. Por otro lado, a lo largo de los años, esta institución ha construido un ecosistema de plataformas educativas para brindar apoyo dentro de las aulas educativas (*Qué es Ceibal - Ceibal*, s.f.).

CEIBAL le ha proporcionado a ANEP datos de los accesos a las plataformas educativas por parte de estudiantes. A partir de estos datos, la institución quiere generar reportes anuales con distintas métricas de la interacción de los estudiantes con estas plataformas (Por ejemplo, cantidad de accesos, cantidad de actividades realizadas, cantidad de comentarios, entre otros.), para poder identificar si es necesario optimizar los recursos de las mismas según la demanda. Por otro lado, también buscan analizar estas métricas según la ubicación geográfica, el sexo de los estudiantes y contextos socio-económicos. Esto permitiría identificar zonas donde la cantidad de accesos es menor, que según la ubicación geográfica y el contexto socio-económico, se puede deber a la falta o problemas de conectividad, permitiendo mejorar estos aspectos. Por último, también interesa analizar las métricas según los subsistemas de ANEP. Esto último puede ayudar a identificar en qué subsistema son más utilizadas las plataformas e incorporar actividades para grados educativos más altos o bajos, según sea necesario.

Por otro lado, se cuenta con datos de los distintos centros educativos de ANEP. Este organismo estatal desea analizar la cantidad de centros según la distribución geográfica de los mismos. Esto permite identificar áreas con limitado acceso a estas instalaciones y así mejorar el acceso a las mismas. También interesa analizar esta métrica según los distintos turnos de los centros y el subsistema de ANEP.

Por otro lado, se quieren desarrollar sistemas de recomendación de materiales educativos, según las preferencias y características de los estudiantes. A su vez, en base a las preferencias y características, se pueden generar y/o modificar

los materiales educativos.

Por último, los docentes se podrían beneficiar de la implementación de modelos predictivos que permitan predecir el desempeño académico de los estudiantes y el riesgo de abandono de la educación, en base a diversos factores, como el compromiso con las plataformas educativas, el contexto socio-económico y la ubicación geográfica.

A partir de este escenario, se pueden identificar los siguientes requisitos los cuales debe cumplir el sistema:

1. Almacenamiento y procesamiento de grandes volúmenes de datos: Debido a la existencia de datos históricos, el sistema debe contar con la capacidad de almacenamiento de los mismos, pudiendo realizar una carga inicial de los datos ya existentes.
2. Generación de Reportes: El consejo directivo requiere la generación de reportes que les permitan ver la cantidad de accesos a las plataformas educativas por parte de estudiantes, así como diversas métricas de los centros educativos.
3. Los reportes generados deben permitir realizar análisis de métricas según distintos factores como contexto socio-económico, ubicación geográfica, subsistema de ANEP, entre otros.
4. Generación de modelos predictivos y sistemas de recomendación.

El primer requerimiento se cumple gracias a la existencia del DL y la incorporación de la arquitectura BRAID en la arquitectura propuesta; ya que se puede realizar un procesamiento batch de los datos, facilitando la carga de datos históricos, y se permite el almacenamiento de grandes volúmenes de datos. A su vez, al poder integrar un DW dentro de la arquitectura, se pueden satisfacer el segundo y tercer requerimiento; ya que los mismos implican la utilización de herramientas de BI para visualización de los datos y la existencia de un modelo multidimensional de los datos para realizar análisis según distintos factores. Por último, el cuarto requerimiento se logra cumplir gracias a las zonas Sandbox y Refined, las cuales pueden ser accedidas por herramientas de AA.

Debido al alcance del prototipo a implementar, el último requerimiento no se implementó, pero sí se implementan los accesos a los datos hacia las distintas zonas desde la zona Sandbox.

6.3. Modelo multidimensional para requerimientos de BI

Para permitir realizar un análisis de los datos de tipo BI utilizando un DW como espacio de almacenamiento, se sigue un enfoque ROLAP. Luego, para im-

plementar este enfoque se utiliza una base de datos relacional y un modelado de los datos siguiendo un esquema estrella (Ver Secciones 2.1.2 y 2.1.3).

En las siguientes 2 secciones se describe el diseño conceptual y lógico del modelo multidimensional de los datos a almacenar en el DW.

6.3.1. Diseño conceptual

En esta sección se presenta el diseño conceptual del modelo multidimensional utilizando el modelo CMDM (Ver Sección 2.1.2).

Acceso a plataformas

A partir del análisis de los campos en los datasets *datos_estudiantes_año* y los requisitos definidos en el caso de uso, se definieron las siguientes dimensiones.

Dimensión "Tiempo"
Para analizar los datos en el tiempo, se crea la dimensión "Tiempo". Esta dimensión tiene una única jerarquía con un solo nivel llamado "Año", debido a que solo se tiene el año como dato temporal en los datasets (Ver Figura 6.3).

Dimensión "Sexo"
Para analizar los datos según el sexo de los estudiantes, se crea la dimensión "Sexo". Esta dimensión tiene una única jerarquía con un solo nivel llamado "Sexo", que puede tomar los valores "Femenino", "Masculino" y "Sin Dato"(Ver Figura 6.3).

Dimensión "Zona"
Para analizar los datos según el contexto socio-económico se crea la dimensión "Zona". Esta dimensión está formada por una única jerarquía con 2 niveles, "Contexto" y "Zona". El nivel "Zona"toma los valores "Rural", "Urbana", "Sin Dato" y "Rural-Urbana-SinDato". Este último valor se incorpora para eliminar las relaciones N a N entre los 2 niveles, y así exigir que existan relaciones 1 a N desde el nivel superior (Zona) al nivel inferior (Contexto). A su vez, el nivel "Contexto" toma los valores "quintil rural 1", "quintil rural 2", "quintil rural 3", "quintil rural 4", "quintil rural 5", "quintil urbano 1", "quintil urbano 2", "quintil urbano 3", "quintil urbano 4", "quintil urbano 5" y "sin dato"(Ver Figura 6.3).

Dimensión "Departamento"
Para analizar los datos según la ubicación geográfica de los estudiantes, se crea la dimensión "Departamento". Esta dimensión está formada por una única jerarquía con un único nivel llamado "Departamento"que toma los valores de los 19 departamentos de Uruguay (Ver Figura 6.4).

Figura 6.3: Dimensiones Tiempo, Sexo y Zona

Dimensión "Subsistema Estudiantes"
Para analizar los datos según el subsistema de ANEP se crea la dimensión "Subsistema Estudiantes". Esta dimensión está formada por una única jerarquía con 3 niveles, "Grado", "Ciclo" y "Subsistema". El nivel "Grado" toma valores numéricos y valores de tipo texto. A su vez, para los grados que tienen asociado más de un ciclo, se le concatena el nombre del ciclo, para exigir que exista una relación 1 a N desde el nivel superior (Ciclo) al nivel inferior (Grado). El nivel "Ciclo" toma los valores "Inicial", "Primaria", "Ciclo Básico", "Bachillerato", "Educación Media", "Formación", "Articulación" y "Otros". Nuevamente, existen valores del nivel "Ciclo" que están asociados a más de un subsistema, por lo que se concatena el nombre del subsistema a estos valores, para exigir relaciones 1 a N entre el nivel "Subsistema" y "Ciclo". El nivel "Subsistema" toma los valores "DGEIP", "DGETP", "CFE" y "DGES" (Ver Figura 6.4).

Relación "Acceso a plataformas"
Para poder analizar distintas métricas según las distintas dimensiones se define la relación "Acceso a plataformas". Las métricas a analizar son "Cantidad de días de ingeso a CREA", "Cantidad de entregas CREA", "Cantidad de comentarios CREA", "Cantidad de acciones totales CREA", "Cantidad de días de ingreso a MATIFIC", "Cantidad de episodios fin MATIFIC", "Cantidad de ingresos PAM", "Cantidad de actividades finalizadas PAM", "Cantidad de días de ingreso a BIBLIOTECA" y "Cantidad de préstamos en BIBLIOTECA"(Ver Figura 6.5).

Centros educativos

A continuación se describen las dimensiones y relaciones definidas para analizar los datos de centros educativos.

Dimensión "Subsistema"

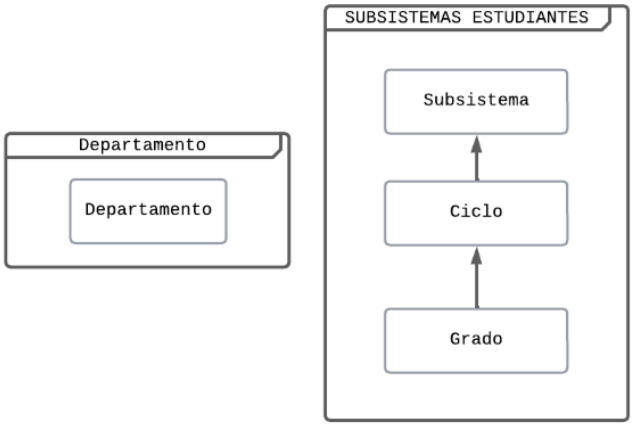

Figura 6.4: Dimensiones Departamento y Subsistema Estudiantes

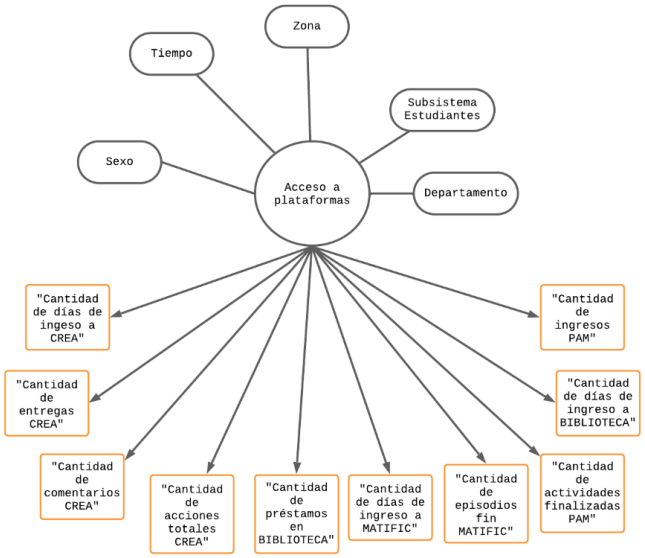

Figura 6.5: Relación “Acceso a plataformas”

Para analizar los datos según los distintos subsistemas de ANEP se crea la dimensión “Subsistema”. Esta dimensión contiene una única jerarquía, con un solo nivel “Subsistema” que toma los valores “DGEIP” y “DGES” (Ver Figura 6.6).

Dimensión “Ubicación”

Para analizar los datos según la ubicación geográfica de los centros, tanto localidad como departamento, se crea la dimensión “Ubicación”. Esta dimensión

contiene una única jerarquía con 2 niveles, “Localidad” y “Departamento”. Para aquellas localidades con el mismo nombre, correspondientes a distintos departamentos, se concatena el nombre del departamento a la localidad, para exigir que haya relaciones 1 a N desde el nivel superior (Departamento) al nivel inferior (Localidad) (Ver Figura 6.6).

Dimensión “Horarios”
Para analizar los datos según los horarios de los centros educativos, se crea la dimensión “Horarios”. Esta dimensión contiene una única jerarquía con 2 niveles, “Horario” y “Tipo”. El nivel “Tipo” toma los valores “Diurno”, “Nocturno”, “Completo”, “Especial”, “Rural” y “Sin Dato” (Ver Figura 6.6).

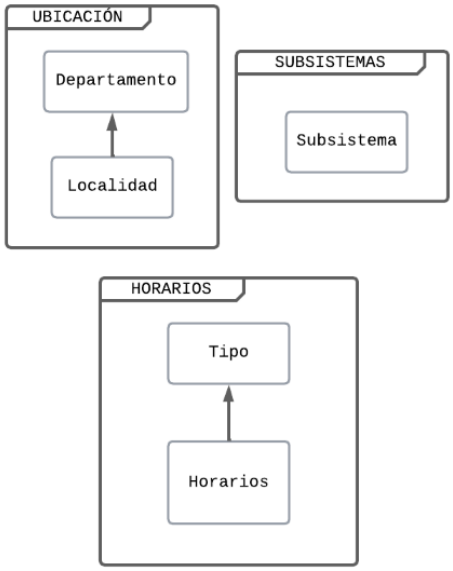

Figura 6.6: Dimensiones “Ubicación”, “Subsistema” y “Horarios”

Relación “Centros educativos”
Para poder analizar la cantidad de centros según las dimensiones “Ubicación”, “Subsistema” y “Horarios”, se crea la relación “Centros educativos” con la métrica “Cantidad de centros” (Ver Figura 6.7).

6.3.2. Diseño lógico

A continuación se presenta el diseño lógico para las relaciones “Acceso plataformas” (Ver Figura 6.8) y “Centros educativos” (Ver Figura 6.9) utilizando un esquema estrella.

Se crearon tablas de dimensión para cada dimensión en la relación, donde

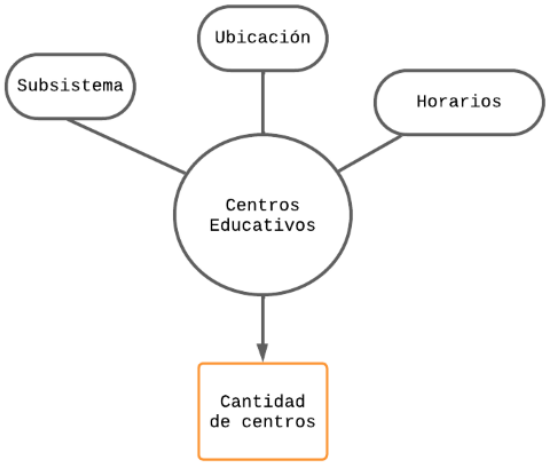

Figura 6.7: Relación “Centros educativos”

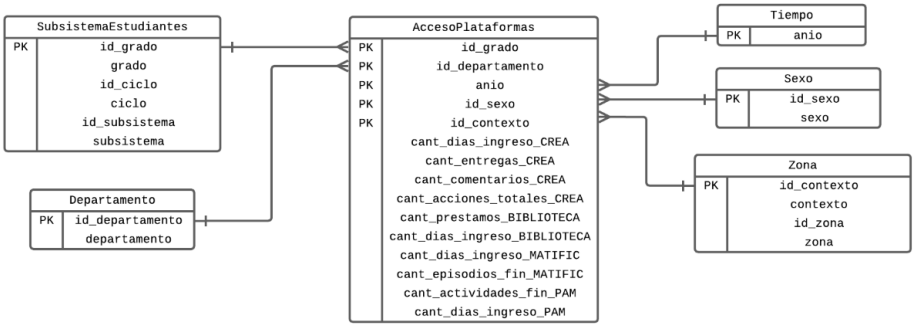

Figura 6.8: Diseño lógico de la relación “Acceso plataformas”

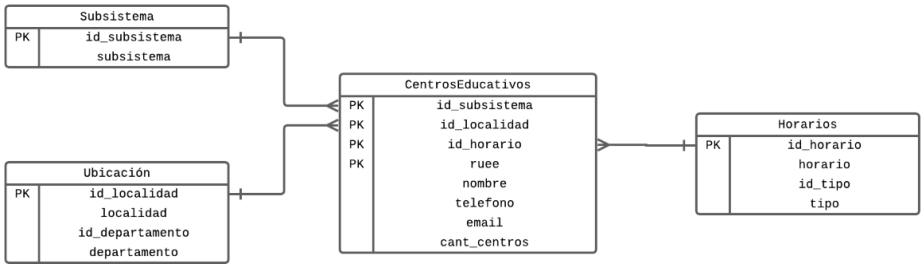

Figura 6.9: Diseño lógico de la relación “Centros educativos”

las jerarquías se encuentran desnormalizadas dentro de estas tablas. A su vez, se incorporan identificadores para cada nivel de la jerarquía, tomando como clave primaria de la tabla el indicador del nivel más bajo de la jerarquía. Por

otro lado, se crean tablas de hechos, las cuales se relacionan con las tablas de dimensión con relaciones N a 1. A su vez, estas tablas contienen las columnas asociadas a las medidas de la relación y su clave primaria es la concatenación de las claves primarias de las tablas de dimensión.

6.4. Implementación de la arquitectura

Para comenzar con la implementación de los procesamientos sobre los datos se tuvo que definir cómo se iban a almacenar los datos en HDFS. Debido a que los datasets no superan los 71 MB se decidió crear un directorio para cada zona que contiene almacenamiento de datos. Por lo tanto, en HDFS se crearon directorios para las zonas Landing, Raw, Trusted y Refined. En el caso de la Landing Zone, el directorio asociado se corresponde con el componente *Temporary Storage*, mientras que para las demás zonas se corresponde con el componente *Storage* (Ver Sección 4.2.4).

En caso de trabajar con volúmenes de datos más grandes, en vez de tener un directorio para cada zona, se podrían tener sistemas de archivos para cada zona.

Una vez que se crearon los espacios de almacenamiento para cada zona, se implementaron los flujos de procesamiento de los datos (en adelante, *pipelines*). En la subsección 6.4.1 se define el pipeline que realiza la carga inicial de los datos, mientras que la subsección (6.4.2) describe la implementación de la zona Sandbox.

6.4.1. Pipeline de carga inicial

Este pipeline se encarga de realizar la carga inicial de datos al sistema, desde su extracción de la fuente hasta su consumo por herramientas de BI. A continuación se menciona el procesamiento que se realiza en cada zona.

Como consideración general, para trabajar con los datos de cada dataset, en cualquiera de las zonas, se cargan los datos en un DataFrame de PySpark (Ver Anexo D.3).

Landing Zone y Raw Zone

El pipeline de la Landing Zone obtiene datos del directorio denominado *landing-zone*. Luego, se realizan algunos filtrados sobre los datos, como por ejemplo, la eliminación de columnas que no serán utilizadas a futuro. También se realizan renombres de atributos, para homogeneizar nombres o utilizar nombres más descriptivos para cada columna.

Una vez realizados estos filtrados y homogeneización de campos, se realiza la extracción de metadatos de cada dataset. Estos metadatos se almacenan en el repositorio de metadatos de la Governance Metadata Zone en Neo4j.

Es importante notar que no se desarrolló todo el modelo de metadatos definido en el capítulo 5. Se limitó el alcance para considerar todas las entidades, salvo aquellas que refieren al acceso a datos por parte de usuarios y aplicaciones, la relación *derives* entre una *Measure* y un *Process*, la relación *related* entre 2 datasets y la relación *lastStorageZone*. A su vez, debido al tipo de datos utilizados (estructurados), no se desarrolló el guardado de datos de datasets no-estructurados y semi-estructurados (Ver Figura 6.10 donde se marca en rojo lo no implementado).

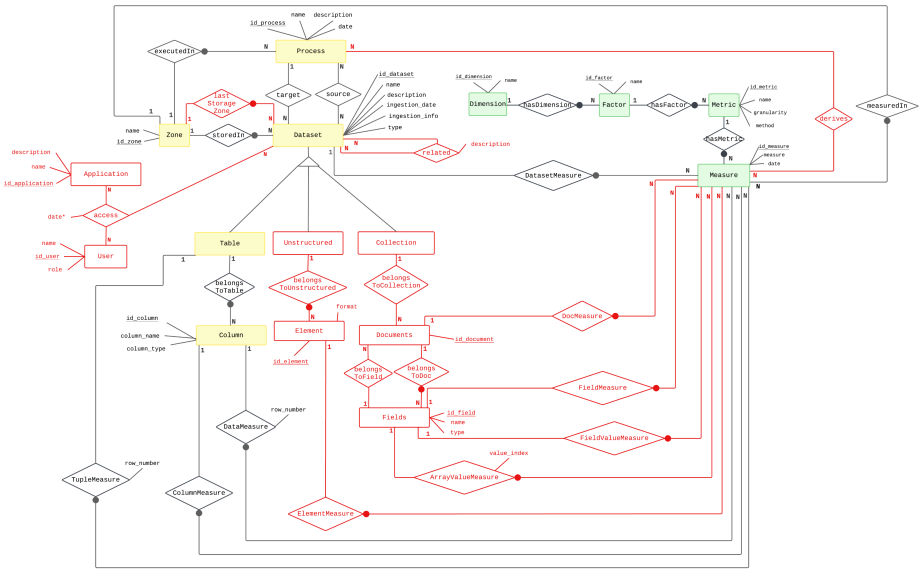

Figura 6.10: Modelo de Metadatos desarrollado.

El almacenamiento de metadatos de cada dataset se realiza mediante la ejecución de consultas CYPHER. En la Consulta 6.1 se presenta el almacenamiento de metadatos para un dataset estructurado.

Listing 6.1: Consulta CYPHER para almacenar metadatos de dataset estructurado

```
CREATE(dataset:Dataset:Table
                {name: 'ds_name',
                description: 'ds_description',
                type: 'ds_type',
                ingestion_date: 'ingestion_date',
                ingestion_data: 'ingestion_data'})
MATCH (zone:Zone {name: 'zone_name'})
WITH dataset, zone
CREATE (dataset)-[:STORED_IN]->(zone)
```

```
WITH dataset
CREATE(column:Column
                    {name: 'column_name',
                    type: 'column_type'})
CREATE (dataset)<-[:BELONGS_TO_TABLE]-(column)
```

En caso que el dataset tenga más de una columna, se debe ejecutar la creación de esta entidad tantas veces como columnas tenga el dataset. En la Figura 6.11 se pueden ver los metadatos asociados al dataset "CEIP".

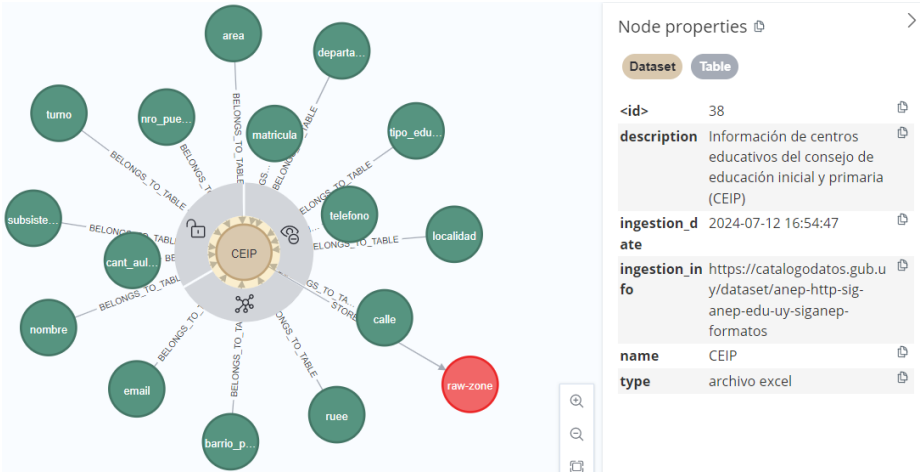

Figura 6.11: Ejemplo de metadatos del dataset *CEIP*

El siguiente procesamiento a realizar es la medición de la calidad de los datasets, generando metadatos de calidad que también se almacenan en la Governance Metadata Zone. En la Figura 6.12 se pueden observar los metadatos de calidad almacenados para el dataset *CEIP*, en la Raw Zone.

Una vez finalizados estos procesamientos, los datasets se eliminan del directorio *landing-zone* y se almacenan en el directorio *raw-zone*, correspondiente al espacio de almacenamiento de la Raw Zone.

En la Raw Zone no se realizan procesamientos, sino que simplemente se almcenan los datos en el mismo formato en el que se encontraban en la fuente (Por ejemplo, si los datos se encontraban en un archivo *.csv* en la fuente, en la Raw Zone se almacenan en un archivo *.csv*).

Trusted Zone

Esta zona obtiene datos del directorio *raw-zone* y realiza transformaciones sobre los mismos. La mayoría de las transformaciones realizadas son del tipo

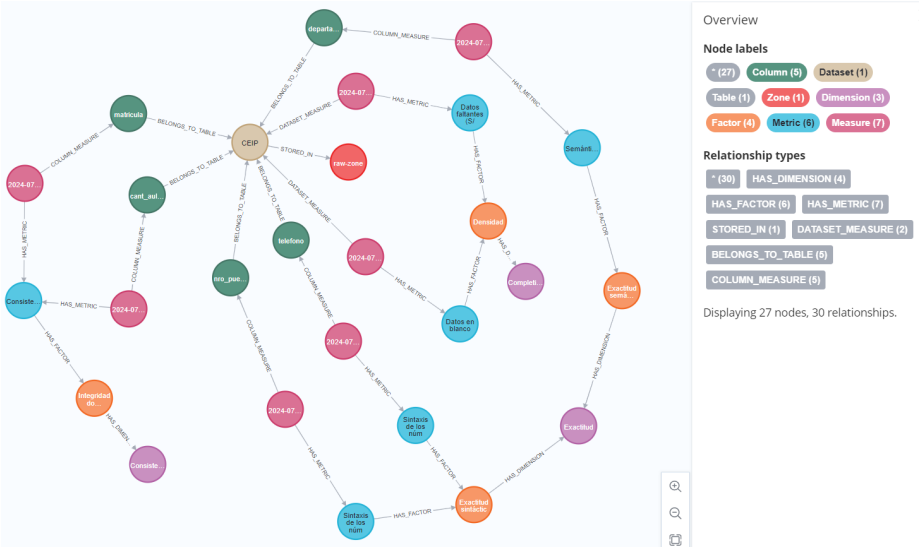

Figura 6.12: Ejemplo de metadatos de calidad del dataset *CEIP*

data cleaning[1], por ejemplo, se homogenizaron los distintos términos utilizados para identificar datos faltantes ("Sin Dato", "NA", entre otros). En el Anexo E.1.1 se describen en detalle las transformaciones realizadas en esta zona.

Una vez realizadas estas transformaciones, se realizan nuevas mediciones de calidad sobre los datasets transformados y se almacenan los metadatos de los mismos en el repositorio de metadatos. Luego, se almacenan los nuevos datasets en el directorio *trusted-zone*. A su vez, se almacenan metadatos asociados al linaje de los nuevos datasets, generados a partir del procesamiento aplicado sobre datasets en la zona anterior. En el Anexo E.1.3 se muestra un ejemplo de la consulta CYPHER que guarda los metadatos de una transformación realizada sobre un conjunto de datos.

Por otro lado, en el caso de los datasets de *datos_estudiantes_año*, debido a que se quiere realizar un análisis histórico de los datos, se debe contar con un dataset integrado que contenga los accesos a las plataformas a lo largo de los años. Entonces en esta zona se realiza la integración de los datasets *datos_estudiantes_año* almacenados en el directorio *trusted-zone*. Debido a que se genera un nuevo dataset, se almacenan los metadatos y el linaje del mismo. En adelante, cada vez que se haga referencia a este dataset integrado se lo nombra como *acceso_plataformas*.

[1]Proceso de corregir o eliminar datos incorrectos, corruptos, con formatos incorrectos, duplicados o incompletos dentro de un conjunto de datos (*Data Cleaning: Definition, Benefits, And How-To | Tableau*, s.f.)

En el caso de los datasets de centros educativos, también se debe realizar una integración de los mismos para obtener un único dataset con información de centros de inicial, primaria y secundaria. Por último, se almacenan los metadatos y el linaje del nuevo dataset integrado en la Governance Metadata Zone. En adelante, cada vez que se haga referencia a este dataset integrado se lo nombra como *centros_educativos*.

En la Figura 6.13 se muestran los metadatos asociados al linaje del dataset *centros_educativos*.

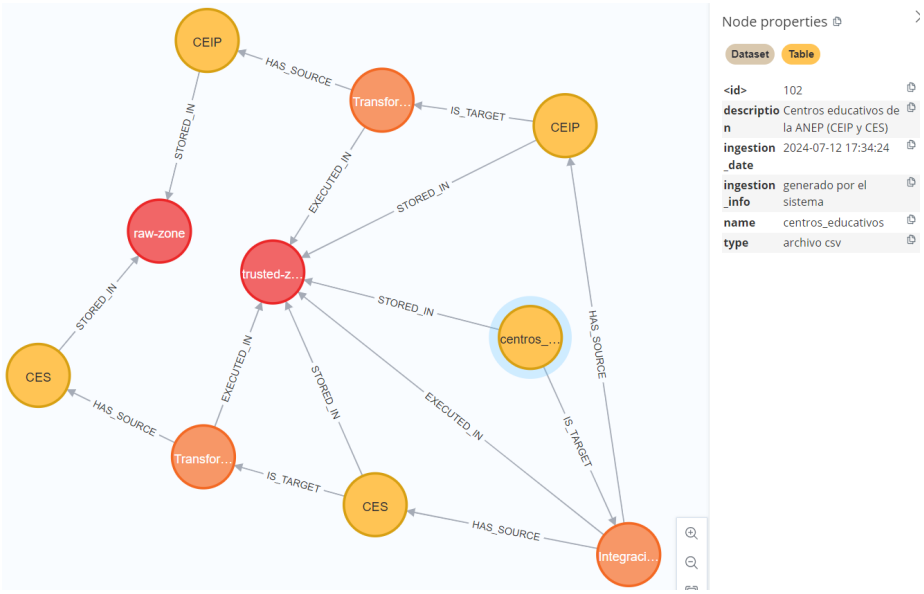

Figura 6.13: Ejemplo de metadatos del linaje del dataset *centros_educativos*

Refined Zone

Esta zona obtiene datos almacenados en el directorio *trusted-zone* y genera las tablas de dimensión y hechos que se almacenan en la base de datos relacional en PostgreSQL. Los datos que se utilizan como fuente para estas transformaciones son los datasets *centros_educativos* y *acceso_plataformas*, extraídos del directorio *trusted-zone*.

Durante la creación de las tablas de dimensión se tuvieron que corregir algunos problemas en los datos, como relaciones N a N entre niveles de las jerarquías. Por lo tanto, se tomaron los datasets *centros_educativos* y *acceso_plataformas*, se cargaron en DataFrames de PySpark y se realizaron las correcciones de los datos sobre estas estructuras. Luego, estos DataFrames transformados son los

que se utilizan para crear las tablas de dimensión Sexo, Ubicación, Zona, SubsistemaEstudiantes, Tiempo, Departamento, Horarios y Subsistema, y las tablas de hechos CentrosEducativos y AccesoPlataformas. En el Anexo E.1.2 se detalla la creación de estas tablas y los problemas encontrados durante la creación de las mismas.

Por último, se guardan metadatos de las diversas tablas, así como el linaje de las mismas (Ver Figura 6.14).

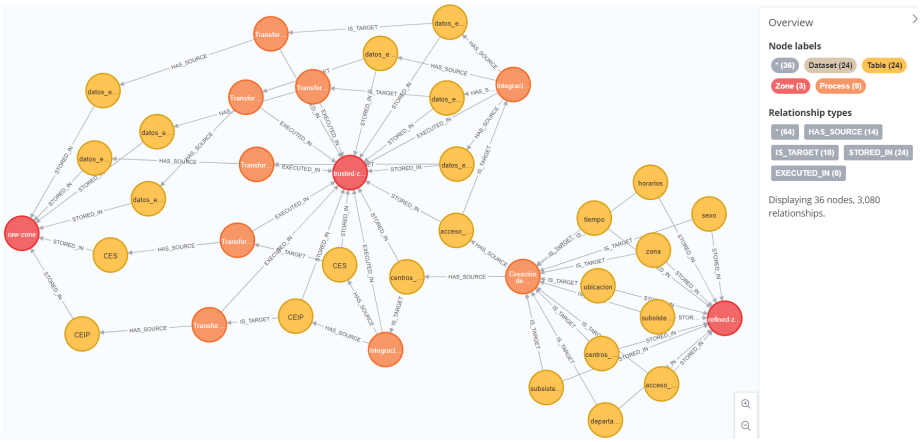

Figura 6.14: Linaje del DW

Para finalizar, en la Figura 6.15 se puede ver un resumen de cómo los datasets atraviesan los distintos componentes de la arquitectura.

6.4.2. Sandbox

Para implementar la zona Sandbox se generó un Jupyter Notebook. Este notebook permite consultar datos de la Governance Metadata Zone, leer datos de las zonas Raw, Trusted y Refined (incluye acceso al DW y al directorio *refined-zone*), y guardar datos en la Raw Zone.

En la Figura 6.16 se puede ver la lógica para consultar la Governance Metadata Zone. Allí se consulta la base de datos en Neo4j utilizando consultas CYPHER y se pueden visualizar los resultados de la consulta en formato texto o en un grafo.

Para leer datos de las distintas zonas de la arquitectura se usa PySpark, con los conectores correspondientes para acceder a *HDFS* o *postgreSQL* (Figura 6.17 y 6.18). En el caso de los datos almacenados en la base de datos *postgreSQL*, se pueden leer datos a través de una consulta SQL o se lee una tabla completa;

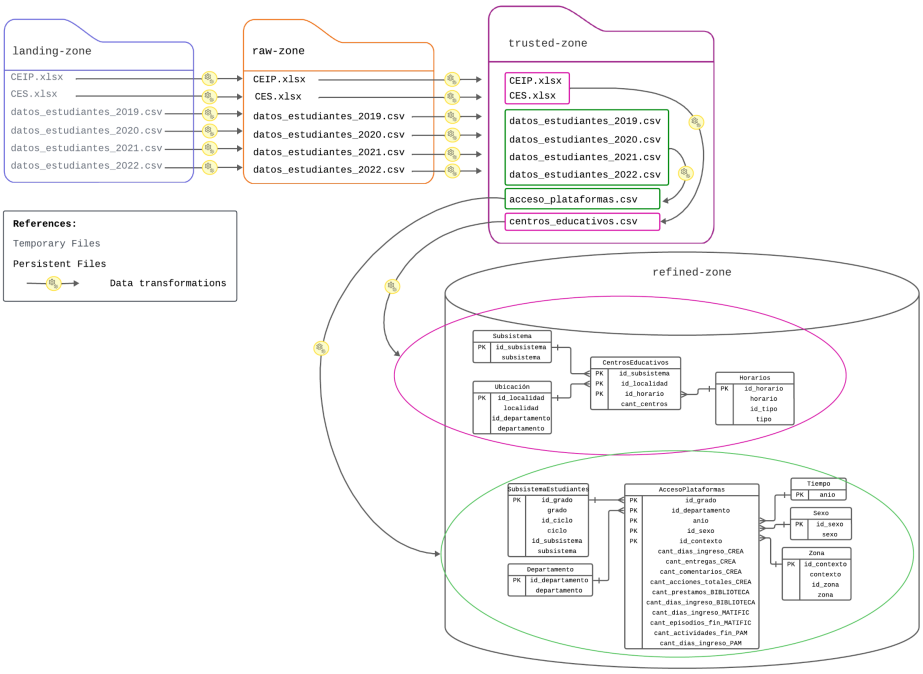

Figura 6.15: Flujo de datos durante la carga del sistema

mientras que en el caso de *HDFS* se leen archivos completos. Por otro lado, en el caso de la escritura de datos en la Raw Zone, simplemente se utiliza el conector de *HDFS* (Ver Figura 6.19).

Por último, notar que para el caso de la lectura/escritura de datos solo se tiene la opción de almacenar archivos con extensión *.csv* y *.xlsx*, debido a que solo se trabajaron con este tipo de archivos. Sin embargo, en caso que se quieran almacenar otros tipos de archivos, se debe utilizar el conector adecuado.

6.5. Gestión de calidad de datos

En esta sección se describen las tareas llevadas a cabo para la incorporación de la GCD en la arquitectura.

En particular, se redujo el proceso de GCD presentado en la Sección 2.5 a las etapas presentadas en la Figura 6.20.

Se eliminaron las etapas “Análisis de procesos de negocio involucrados” y “Re-estructuración del sistema”; ya que no se está trabajando con un sistema de información ya existente. Por otro lado, se eliminó la etapa “Monitoreo de la calidad”; ya que el prototipo de la arquitectura es estático, por lo que no

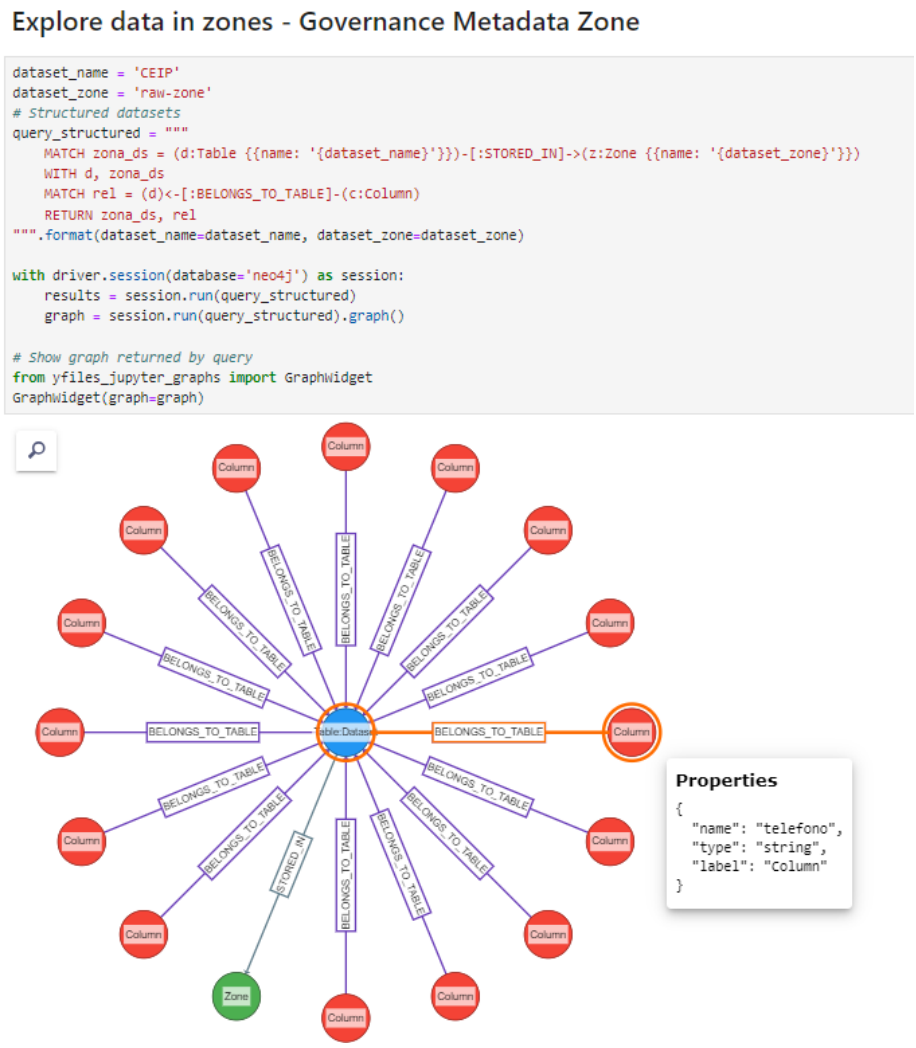

Figura 6.16: Visualización de grafo resultado de la consulta a la Governance Metadata Zone en zona Sandbox

se tienen nuevos datos ingresando al sistema. A su vez, se eliminó la etapa de "Análisis de causas de mala calidad"; ya que los errores de calidad hallados provienen de las fuentes. Notar que en el caso de la etapa "Medición y Evaluación de la calidad" se elimina la Evaluación; ya que no se cuenta con requisitos de calidad por parte de usuarios para evaluar las medidas obtenidas. Por último, se eliminó la etapa de "Estimación de la calidad"; ya que no se cuenta con un sistema existente donde es necesario estimar la calidad para, a su vez, estimar costos de posibles re-estructuraciones del sistema, limpieza, entre otros.

Read data in Zones (Raw, Trusted, Refined & Archival)

```
# Para conectarse a datos en HDFS
from pyspark.sql import SparkSession
spark = SparkSession.builder\
        .appName('Sandbox')\
        .config("spark.hadoop.fs.defaultFS", HDFS_BASE_DIRECTORY)\
        .config("spark.jars.packages", "com.crealytics:spark-excel_2.12-3.5.0_0.20.3") \
        .getOrCreate()

file_name = ''

# Use variables TRUSTED_ZONE, REFINED_ZONE & ARCHIVAL_ZONE to access data in other zones

# Read csv file from Raw Zone
df = spark.read.option('delimiter', ',')\
                .csv(f"{HDFS_BASE_DIRECTORY}/{RAW_ZONE}/{file_name}.csv", header=True)

# Read excel file from Raw Zone
df = spark.read.format("com.crealytics.spark.excel")\
                .option("header", "true")\
                .load(f"{HDFS_BASE_DIRECTORY}/{RAW_ZONE}/{file_name}.xlsx")

spark.stop()
```

Figura 6.17: Lectura de datos de las zonas Raw, Trusted y Refined en la zona Sandbox.

Read data from Datawarehouse ANEP

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Sandbox') \
                            .config("spark.hadoop.fs.defaultFS", HDFS_BASE_DIRECTORY)\
                            .config("spark.jars.packages", "com.crealytics:spark-excel_2.12-3.5.0_0.20.3") \
                            .getOrCreate()

# Read table
table_name = ''
df = spark.read.jdbc(url=jdbc_url, table=table_name, properties=jdbc_properties)

# SQL query
query = ""
df = spark.read.jdbc(url=jdbc_url, table=query, properties=connection_properties)

spark.stop()
```

Figura 6.18: Lectura de datos en DW en la zona Sandbox.

Cabe destacar que la descripción de la etapa de "Limpieza" se encuentra en la Sección 6.4 y en el Anexo E de este capítulo; ya que los procesos de limpieza están embebidos en las transformaciones realizadas en la *Trusted Zone*.

La primera etapa llevada a cabo fue el Data Profiling de los datos, donde se analizó su estructura y se identificaron problemas de calidad existentes, y la frecuencia de los mismos. A partir de los problemas de calidad identificados en esta etapa se definió el modelo de calidad de datos. Una vez definido el modelo de calidad, se implementaron los métodos de medición y se llevaron a cabo las mediciones de calidad en la Landing Zone. A partir de las medidas obtenidas se definieron transformaciones sobre los datos, en la Trusted Zone, para corregir

Save data to Raw Zone

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Sandbox') \
                            .config("spark.hadoop.fs.defaultFS", HDFS_BASE_DIRECTORY)\
                            .config("spark.jars.packages", "com.crealytics:spark-excel_2.12-3.5.0_0.20.3") \
                            .getOrCreate()

df = None
data_format = '' # excel, parquet o csv
new_file_name = ''

saveDataToHDFS(spark, df, data_format, new_file_name)

spark.stop()
```

Figura 6.19: Escritura de datos en la Raw Zone desde la zona Sandbox.

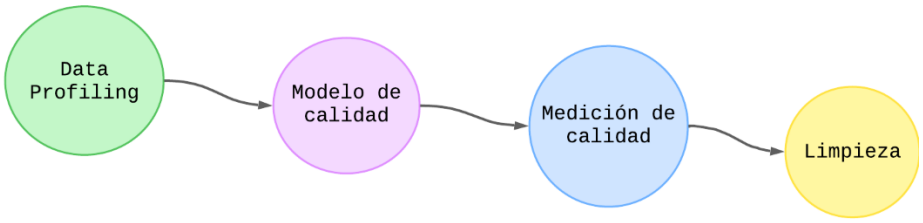

Figura 6.20: Proceso de Gestión de Calidad implementado en la arquitectura.

problemas de calidad y se volvieron a tomar medidas sobre los datos corregidos.

En el Anexo E.2 se describen en detalle cada una de las etapas realizadas.

6.6. Dashboards

En esta sección se presenta un ejemplo de dashboard generado con la herramienta Power BI de Microsoft.

Se generaron 2 dashboards, uno para cada relación multidimensional definida en la Sección 6.3.1.

En la Figura 6.21 se presenta el reporte para la relación "Centros Educativos". Este reporte está compuesto por 1 página, la cual muestra la cantidad de centros por Departamento (Ver Tabla en Figura 6.21), y por Departamento y Tipo de Horario (Ver Gráfico en Figura 6.21). A su vez, se puede realizar *drill-down* sobre la medida "Cantidad de Centros" (Ver Tabla en Figura 6.22), por lo que se puede ver la medida en el nivel Localidad, siguiente nivel en la jerarquía de la dimensión Ubicación.

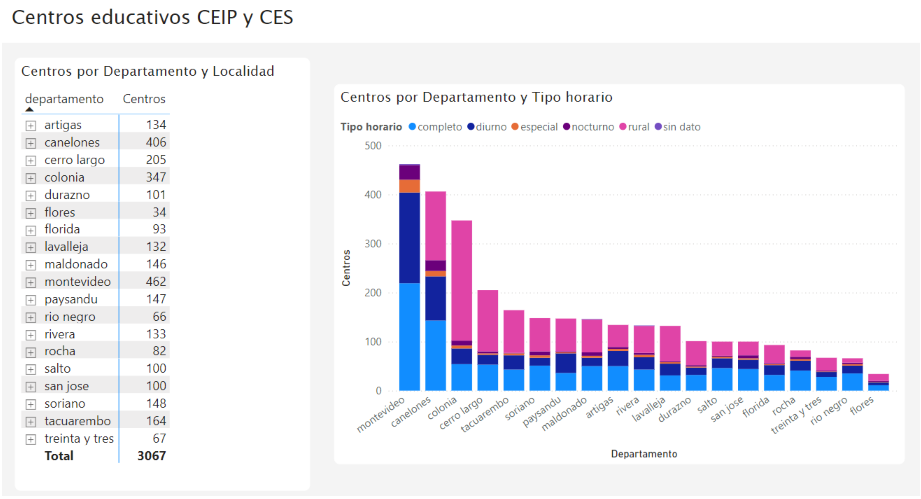

Figura 6.21: Dashboard “Centros educativos”.

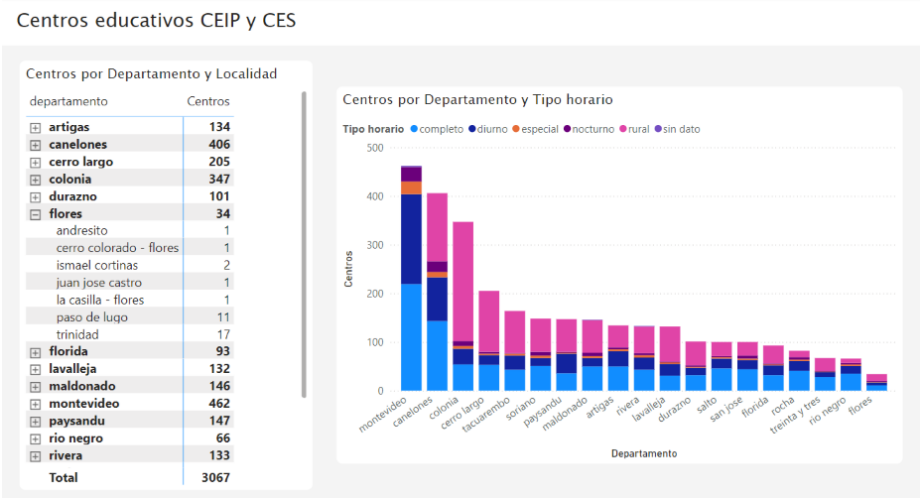

Figura 6.22: Dashboard “Centros educativos” con *drill-down* sobre la dimensión Ubicación para el Departamento “Flores”.

En el Anexo E.3 se describe el dashboard creado para la relación “Acceso a plataformas”.

6.7. Conclusión

Se desarrolló un prototipo de la arquitectura propuesta utilizando Apache Spark, HDFS, PostgreSQL y Neo4j, donde se excluyó el procesamiento de stream y todo lo relacionado a la Archival Zone.

La implementación de este prototipo permitió demostrar la viabilidad de la arquitectura propuesta, instanciada en un caso de uso real; ya que se logró almacenar y transformar grandes volúmenes de datos heterogéneos, logrando cumplir con los requerimientos de análisis de datos establecidos en el caso de uso; dado que estos pudieron ser accedidos por una herramienta de BI.

Sin embargo, la integración manual de las tecnologías utilizadas aumentó el tiempo de desarrollo. Este tiempo podría reducirse significativamente utilizando alguna de las plataformas descritas en el Anexo A.2; ya que algunas de estas plataformas proporcionan la integración de estas tecnologías, agilizando los procesos de configuración y gestión.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones de este proyecto y se exponen recomendaciones para futuros trabajos de investigación.

7.1. Conclusiones

Este proyecto tenía 3 objetivos principales. El primer objetivo planteaba la definición de una arquitectura de BD genérica. Para cumplir con este objetivo, se realizó un estudio del estado del arte de las diversas arquitecturas de BD existentes, tanto teóricas como de proveedores. A partir de este estudio, se desarrolló una propuesta teórica de una arquitectura de BD, la cual considera aspectos de diversas arquitecturas estudiadas, permitiendo almacenar grandes volúmenes de datos heterogéneos, realizar procesamientos (stream y batch) sobre los mismos, también incluye aspectos para permitir llevar a cabo un correcto gobierno de los datos y puede ser accedida por herramientas de BI y AA, que son las principales cargas de trabajo en BD. A su vez, luego de la definición de la propuesta teórica, se establecieron algunas modificaciones sobre la misma para que esta pueda ser desarrollada con tecnologías de proveedores, y así mejorar algunos aspectos de la misma.

El segundo objetivo del trabajo buscaba integrar un proceso de GCD en la arquitectura propuesta, con un sub-objetivo que planteaba la definición de un modelo de metadatos que incorpore metadatos de calidad. Se logró integrar el proceso de GCD dentro de la arquitectura, identificando en qué componentes actúan distintas etapas del mismo. También se logró definir una primera aproximación del modelo de metadatos, el cual busca modelar distintos aspectos de los datos almacenados, así como de los procesos llevados a cabo dentro de la arquitectura, vinculando aspectos de calidad.

El último objetivo planteaba la implementación de un prototipo de la arquitectura propuesta, incluyendo la gestión de calidad dentro de la misma. Para cumplir con este objetivo se definió un prototipo de la arquitectura, restringiendo el flujo de procesamiento de datos y el modelo de metadatos definido, debido al alcance de este proyecto. Luego, se definió un caso de uso con datos obtenidos del Catálogo Abierto de Datos de Uruguay, a partir del cual se implementaron diversas transformaciones sobre los datos, se midió la calidad de los mismos y se generaron visualizaciones de los datos con una herramienta de BI. La implementación de este prototipo permitió mostrar la viabilidad de la arquitectura propuesta, instanciada en un caso de uso real; ya que se logró almacenar y transformar grandes volúmenes de datos heterogéneos, utilizando tecnologías populares para casos de uso de BD.

Por último, la arquitectura propuesta fue presentada como parte del artículo (Cortés, Sanz, Etcheverry, y Marotta, 2024) en la conferencia "50th International Conference on Very Large Databases" del año 2024, donde fue aceptado y será presentado próximamente.

7.2. Trabajo a futuro

Debido a la complejidad de las arquitecturas de BD, las distintas cargas de trabajo que deben soportar, así como la gestión de los datos, existen muchas mejoras a realizar sobre la arquitectura propuesta, la gestión de calidad dentro de la misma y el prototipo desarrollado.

7.2.1. Arquitectura propuesta y Gestión de calidad

En la Sección 4.4 se propone una mejora a realizar sobre la arquitectura propuesta, que implica utilizar tecnologías de proveedores, por ejemplo *Delta tables* (Ver Anexo C.3) para el almacenamiento de datos estructurados (y semi-estructurados, opcionalmente), y *Apache Spark* para consultar datos. Por lo tanto, creemos que puede ser un buen aporte implementar esta segunda versión de la arquitectura, permitiendo verificar la factibilidad técnica y tecnológica de la misma.

Con respecto al modelo de metadatos de la arquitectura, durante su definición no se consideraron algunos aspectos definidos en la Sección 4.2.2, debido al alcance y foco del proyecto. Sin embargo, creemos que los metadatos asociados a la evolución de los datasets y a análisis realizados por parte de DST o expertos de ML, son importantes dentro de este tipo de arquitectura. Por lo tanto, se propone como trabajo a futuro extender el modelo de metadatos definido para incorporar estos aspectos. También creemos importante incorporar aspectos de Evaluación de la Calidad en el modelo de metadatos, como por ejemplo, requerimientos de calidad de los datos impuestos por usuarios del sistema, los cuales se asocien a las respectivas métricas de calidad que generan, y valoraciones rea-

lizadas sobre las medidas de calidad obtenidas, indicando si se logran cumplir los requerimientos o no. Por otro lado, cuando se definió el modelo de metadatos de datos y procesos, se definió un dataset semi-estructurado como una colección de documentos. Sin embargo, existen otros tipos de datos semi-estructurados como por ejemplo, datos que siguen un modelo de grafo. Por lo tanto, se propone como trabajo a futuro incluir las entidades correspondientes al modelo para considerar otros tipos de datasets semi-estructurados.

Por último, en este trabajo se hizo foco en los aspectos de Gobierno de Datos asociados a Gestión de Metadatos y Calidad de Datos. Sin embargo, para lograr tener una completa gobernanza de datos en el sistema, se deben incorporar aspectos de seguridad, tanto del sistema como de los datos. Por lo tanto, se propone como trabajo a futuro icorporar aspectos de seguridad al sistema.

7.2.2. Prototipo de la arquitectura

Durante la etapa de experimentación se limitó el alcance del prototipo de la arquitectura a desarrollar, considerando únicamente el procesamiento batch de datos, por lo que se propone como trabajo a futuro desarrollar el procesamiento para datos de stream. A su vez, los procesamientos de datos realizados en el prototipo solo tienen en cuenta la carga inicial de los datos, por lo que también se propone implementar *pipelines* que permitan incorporar nuevos datos al sistema y gestionar cambios sobre datos ya almacenados en el mismo.

Otra restricción impuesta sobre el prototipo de la arquitectura fue la eliminación de la Archival Zone; ya que no se trabajó con grandes cantidades de datos que tuvieran que enviarse a esta zona. Luego, se propone implementar los procesos y espacios de almacenamiento asociados a esta zona.

Con respecto al caso de uso definido, este solo considera datos estructurados, aspecto que también impacta en la gestión de calidad de datos; ya que solo se consideraron Dimensiones y Factores de calidad para datos estructurados. Por lo tanto, se propone como trabajo a futuro incluir datos no-estructurados y semi-estructruados en el sistema, de los cuales se puedan extraer sus metadatos, tomar distintas medidas de calidad y realizar procesos de mejora. Notar que estos nuevos datos exigirían la implementación de algunas entidades del modelo de metadatos que no fueron implementadas en el prototipo debido a los datos utilizados.

Por último, se debería desarrollar una interfaz que permita consultar el modelo de metadatos de forma más amigable para usuarios no técnicos; ya que de momento la única forma de consultar este modelo es teniendo conocimiento del modelo y del lenguaje CYPHER.

Por otra parte, por más que se logró implementar todos los aspectos de la zona Sandbox, se propone como trabajo a futuro ocultar aspectos de conexión

al sistema, los cuales se encuentran visibles en el Jupyter Notebook y pueden generar problemas a nivel de seguridad.

Referencias

Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad, y de la Información y el Conocimiento (AGESIC). (2012, agosto). *Información Geográfica – Modelo de Direcciones Geográficas del Uruguay.* Descargado de https://www.gub.uy/agencia-gobierno-electronico-sociedad-informacion-conocimiento/sites/agencia-gobierno-electronico-sociedad-informacion-conocimiento/files/documentos/publicaciones/modelo_de_direcciones_geograficas_del_uruguay_ed01_00.pdf

Apache Arrow. (s.f.). Descargado 2024-06-01, de https://arrow.apache.org/

Apache Hadoop. (s.f.). Descargado 2024-06-08, de https://hadoop.apache.org/

Apache Hadoop: What is it and how can you use it? (2019, abril). Descargado 2024-06-08, de https://www.databricks.com/glossary/hadoop

Apache Hudi. (s.f.). Descargado 2024-06-08, de https://hudi.apache.org/tech-specs/

Apache Iceberg - Apache Iceberg. (s.f.). Descargado 2024-06-29, de https://iceberg.apache.org/

Apache Iceberg | Build an open data lakehouse architecture. (s.f.). Descargado 2024-06-01, de https://www.starburst.io/data-glossary/apache-iceberg/

Apache Spark. (2019, abril). Descargado 2024-06-09, de https://www.databricks.com/glossary/what-is-apache-spark

Apache Spark™ - Unified Engine for large-scale data analytics. (s.f.). Descargado 2024-06-08, de https://spark.apache.org/

Armbrust, M., Ghodsi, A., Xin, R., y Zaharia, M. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics.

Background - apache orc. (s.f.). Descargado 2024-06-08, de https://orc.apache.org/docs/

Batini, C., y Scannapieco, M. (2016). *Data and information quality.* Springer.

Benjelloun, S., Aissi, M. E. M. E., Loukili, Y., Lakhrissi, Y., Ali, S. E. B., Chougrad, H., y Boushaki, A. E. (2020). Big data processing: Batch-based processing and stream-based processing. En *2020 fourth international conference on intelligent computing in data sciences (icds)* (p. 1-6). doi: 10.1109/ICDS50568.2020.9268684

Berk, M. (2022, agosto). *Demystifying the Parquet File Format.* Descargado 2024-06-08, de https://towardsdatascience.com/demystifying-the-parquet-file-format-13adb0206705

BigQuery enterprise data warehouse. (s.f.). Descargado 2024-06-01, de https://cloud.google.com/bigquery

Cloudera, ., Terms, I. A. r. r., Statement, C. |. P., Hadoop, D. P. |. U. . D. N. S. M. P. I. A., trademarks, a. o. s. p. n. a. t. o. t. A. S. F. F. a. c. l. o., y Here, C. (s.f.). *The Open Data Lakehouse.* Descargado 2024-06-01, de https://www.cloudera.com/content/dam/www/marketing/resources/whitepapers/the-open-data-lakehouse.pdf.landing.html

Cortés, C., Sanz, C., Etcheverry, L., y Marotta, A. (2024, Agosto). Data Quality Management for Responsible AI in Data Lakes. En *Aceptado para 50th international conference on very large databases.* Guangzhou, China.

Cristalli, E., Serra, F., y Marotta, A. (2018). Data Quality Evaluation in Document Oriented Data Stores. En C. Woo, J. Lu, Z. Li, T. W. Ling, G. Li, y M. L. Lee (Eds.), *Advances in Conceptual Modeling* (pp. 309–318). Cham: Springer International Publishing. doi: 10.1007/978-3-030-01391-2_35

Data Cleaning: Definition, Benefits, And How-To | Tableau. (s.f.). Descargado 2024-07-16, de https://www.tableau.com/learn/articles/what-is-data-cleaning

Definition of Advanced Analytics - Gartner Information Technology Glossary. (s.f.). Descargado 2024-06-07, de https://www.gartner.com/en/information-technology/glossary/advanced-analytics

delta/PROTOCOL.md at master · delta-io/delta. (s.f.). Descargado 2024-06-08, de https://github.com/delta-io/delta/blob/master/PROTOCOL.md

Eichler, R., Giebler, C., Gröger, C., Schwarz, H., y Mitschang, B. (2020). HANDLE - A Generic Metadata Model for Data Lakes. En M. Song, I.-Y. Song, G. Kotsis, A. M. Tjoa, y I. Khalil (Eds.), *Big Data Analytics and Knowledge Discovery* (Vol. 12393, pp. 73–88). Cham: Springer International Publishing. Descargado 2023-12-20, de http://link.springer.com/10.1007/978-3-030-59065-9_7 (Series Title: Lecture Notes in Computer Science) doi: 10.1007/978-3-030-59065-9_7

Elouataoui, W., El Alaoui, I., el Mendili, S., y Youssef, G. (2022, diciembre). An Advanced Big Data Quality Framework Based on Weighted Metrics. *Big Data and Cognitive Computing*, *13*. doi: 10.3390/bdcc6040153

Etcheverry, L. (s.f.). *Curso bases de datos no relacionales.* Descargado de https://eva.fing.edu.uy/course/view.php?id=947

Etcheverry, L., Marotta, A., Sanz, C., y Serra, F. (s.f.). *Curso calidad de datos e información.* Descargado de https://eva.fing.edu.uy/course/view.php?id=1073

Executing Data Quality Projects. (2021). Descargado de https://shop.elsevier.com/books/executing-data-quality-projects/mcgilvray/978-0-12-818015-0

File Format. (s.f.). Descargado 2024-06-05, de https://parquet.apache.org/docs/file-format/

Giebler, C., Groger, C., Hoos, E., Schwarz, H., y Mitschang, B. (2020, octubre). A Zone Reference Model for Enterprise-Grade Data Lake Management. En *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)* (pp. 57–66). Eindhoven, Netherlands: IEEE. Descargado 2023-12-06, de https://ieeexplore.ieee.org/document/9233155/ doi: 10.1109/EDOC49727.2020.00017

Giebler, C., Gröger, C., Hoos, E., Eichler, R., Schwarz, H., y Mitschang, B. (s.f.). The Data Lake Architecture Framework: A Foundation for Building a Comprehensive Data Lake Architecture.

Giebler, C., Gröger, C., Hoos, E., Schwarz, H., y Mitschang, B. (2019). Leveraging the Data Lake: Current State and Challenges. En C. Ordonez, I.-Y. Song, G. Anderst-Kotsis, A. M. Tjoa, y I. Khalil (Eds.), (Vol. 11708, pp. 179–188). Cham: Springer International Publishing. Descargado 2024-03-23, de https://link.springer.com/10.1007/978-3-030-27520-4_13 (Book Title: Big Data Analytics and Knowledge Discovery Series Title: Lecture Notes in Computer Science) doi: 10.1007/978-3-030-27520-4_13

Giebler, C., Stach, C., Schwarz, H., y Mitschang, B. (2018). BRAID - A Hybrid Processing Architecture for Big Data:. En *Proceedings of the 7th International Conference on Data Science, Technology and Applications* (pp. 294–301). Porto, Portugal: SCITEPRESS - Science and Technology Publications. Descargado 2023-10-23, de http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006861802940301 doi: 10.5220/0006861802940301

Hai, R., Koutras, C., Quix, C., y Jarke, M. (2023, diciembre). Data Lakes: A Survey of Functions and Systems. *IEEE Transactions on Knowledge and Data Engineering*, *35*(12), 12571–12590. Descargado 2024-03-23, de http://arxiv.org/abs/2106.09592 (arXiv:2106.09592 [cs]) doi: 10.1109/TKDE.2023.3270101

Hai, R., Quix, C., y Jarke, M. (2021). *Data lake concept and systems: a survey.*

Hansen, J. (s.f.). *Selling the Data Lakehouse | by Jeremiah Hansen | Snowflake.* Descargado 2024-03-13, de https://medium.com/snowflake/selling-the-data-lakehouse-a9f25f67c906

Harby, A. A., y Zulkernine, F. (2022, diciembre). From Data Warehouse to Lakehouse: A Comparative Review. En *2022 IEEE International Conference on Big Data (Big Data)* (pp. 389–395). Osaka, Japan: IEEE. Descargado 2023-12-13, de https://ieeexplore.ieee.org/document/10020719/ doi: 10.1109/BigData55660.2022.10020719

Hello from Apache Hudi | Apache Hudi. (s.f.). Descargado 2024-06-29, de https://hudi.apache.org/

Herden, O. (2020). Architectural Patterns for Integrating Data Lakes into Data Warehouse Architectures. En L. Bellatreche, V. Goyal, H. Fujita, A. Mondal, y P. K. Reddy (Eds.), *Big Data Analytics* (pp. 12–27). Cham: Springer International Publishing. doi: 10.1007/978-3-030-66665-1_2

Herman-Wu. (s.f.). *Secure a data lakehouse on Synapse - Azure Architecture Center.* Descargado 2024-05-30, de https://learn.microsoft.com/en-us/azure/architecture/example-scenario/analytics/secure-data-lakehouse-synapse

Hlupić, T., Oreščanin, D., Ružak, D., y Baranović, M. (2022, mayo). An Overview of Current Data Lake Architecture Models. En *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1082–1087). Descargado 2023-10-01, de https://ieeexplore.ieee.org/document/9803717 (ISSN: 2623-8764) doi: 10.23919/MIPRO55190.2022.9803717

Home. (s.f.). Descargado 2024-06-29, de https://delta.io

Howard, R. (2024, junio). *RDF vs. Property Graphs: Choosing the Right Approach for Implementing a Knowledge Graph.* Descargado 2024-07-13, de https://neo4j.com/blog/rdf-vs-property-graphs-knowledge-graphs/

Impala. (s.f.). Descargado 2024-06-01, de https://impala.apache.org/

Inmon, B. (2016). *Data lake architecture: Designing the data lake and avoiding the garbage dump.* Technics Publications.

Inmon, B., Levins, M., y Srivastava, R. (2021). *Building the Data Lakehouse.* Technics Publications.

Inmon, W. H. (2005). *Building the Data Warehouse, 4th Edition | Wiley.* Wiley Computer Publishing.

Introduction - Apache Iceberg. (s.f.). Descargado 2024-06-08, de https://iceberg.apache.org/docs/nightly/#user-experience

Key Concepts & Architecture | Snowflake Documentation. (s.f.). Descargado 2024-05-31, de https://docs.snowflake.com/en/user-guide/intro-key-concepts

Linstedt, D., y Olschimke, M. (2015). *Building a Scalable Data Warehouse with Data Vault 2.0.* (Pages: 661)

Marotta, A., y Serra, F. (s.f.). *Curso sistemas de información para el análisis de grandes volúmenes de datos.* Descargado de https://eva.fing.edu.uy/course/view.php?id=1234

Mazumdar, D., Hughes, J., y Onofre, J. B. (2023, octubre). *The Data Lakehouse: Data Warehousing and More.* Descargado 2024-03-27, de https://arxiv.org/abs/2310.08697v1

Megdiche, I., Ravat, F., y Zhao, Y. (2021). Metadata Management on Data Processing in Data Lakes. En T. Bureš y cols. (Eds.), *SOFSEM 2021: Theory and Practice of Computer Science* (pp. 553–562). Cham: Springer International Publishing. doi: 10.1007/978-3-030-67731-2_40

Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., y Arocena, P. C. (2019, agosto). Data lake management: challenges and opportunities. *Proc. VLDB Endow.*, *12*(12), 1986–1989. Descargado 2024-07-03, de https://doi.org/10.14778/3352063.3352116 doi: 10.14778/3352063.3352116

Nuevo Plan de Educación Básica Integrada | Administración Nacional de Educación Pública. (s.f.). Descargado 2024-06-21, de https://www.anep.edu.uy/15-d/nuevo-plan-educaci-n-b-sica-integrada

Orescanin, D., y Hlupic, T. (2021, septiembre). Data Lakehouse - a Novel Step in Analytics Architecture. En *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1242–1246). Opatija, Croatia: IEEE. Descargado 2023-12-06, de https://ieeexplore.ieee.org/document/9597091/ doi: 10.23919/MIPRO52101.2021.9597091

Oukhouya, L., El Haddadi, A., Er-raha, B., y Asri, H. (2021). A generic metadata management model for heterogeneous sources in a data warehouse. *E3S Web of Conferences*, *297*, 01069. Descargado 2023-12-17, de https://www.e3s-conferences.org/10.1051/e3sconf/202129701069 doi: 10.1051/e3sconf/202129701069

Quix, C., y Hai, R. (2019). Data Lake. En S. Sakr y A. Y. Zomaya (Eds.), *Encyclopedia of Big Data Technologies* (pp. 552–559). Cham: Springer International Publishing. Descargado 2023-10-29, de https://doi.org/10.1007/978-3-319-77525-8_7 doi: 10.1007/978-3-319-77525-8_7

Qué es Ceibal - Ceibal. (s.f.). Descargado de https://ceibal.edu.uy/institucional/que-es-ceibal/

Ramchand, S., y Mahmood, T. (2022, junio). Big data architectures for data lakes: A systematic literature review. En *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1141–1146). Descargado 2024-03-23, de https://ieeexplore.ieee.org/document/9842704 (ISSN: 0730-3157) doi: 10.1109/COMPSAC54236.2022.00179

Ravat, F., y Zhao, Y. (2019a). Data Lakes: Trends and Perspectives. En S. Hartmann, J. Küng, S. Chakravarthy, G. Anderst-Kotsis, A. M. Tjoa, y I. Khalil (Eds.), *Database and Expert Systems Applications* (pp. 304–313). Cham: Springer International Publishing. doi: 10.1007/978-3-030-27615-7_23

Ravat, F., y Zhao, Y. (2019b, septiembre). Metadata Management for Data Lakes. En *23rd East-European Conference on Advances in Databases and Information Systems (ADBIS 2019)* (Vol. ADBIS 2019 S, pp. 37–44). Bled, Slovenia. Descargado 2023-12-20, de https://hal.science/hal-02397467

Schneider, J., Gröger, C., Lutsch, A., Schwarz, H., y Mitschang, B. (2023). Assessing the Lakehouse: Analysis, Requirements and Definition. En (Vol. 1, pp. 44–46). (ISSN: 2184-4992) doi: 10.5220/0011840500003467

The scope of the lakehouse platform. (s.f.). Descargado de https://docs.databricks.com

Sharma, B. (2018). *Architecting Data Lakes* (2nd ed.). O'Reilly Media, Inc.

Spec - Apache Iceberg. (s.f.). Descargado 2024-06-08, de https://iceberg.apache.org/spec/#goals

Specification. (s.f.). Descargado 2024-06-05, de https://avro.apache.org/docs/1.11.1/specification/

The SQL Lakehouse Platform for Enterprise | Dremio. (s.f.). Descargado 2024-06-01, de https://www.dremio.com/reference-architecture/

Timeline | Apache Hudi. (s.f.). Descargado 2024-06-08, de https://hudi.apache.org/docs/next/timeline/

Tutorial: Load and transform data using Apache Spark DataFrames. (s.f.). Descargado 2024-06-17, de https://docs.databricks.com

Unifying data lakes and data warehouses across clouds with BigLake. (s.f.). Descargado 2024-06-01, de https://cloud.google.com/blog/products/data-analytics/unifying-data-lakes-and-data-warehouses-across-clouds-with-biglake

Use Cases | Apache Hudi. (s.f.). Descargado 2024-06-08, de https://hudi.apache.org/docs/use_cases/

What Is Apache Parquet? | Dremio. (s.f.). Descargado 2024-06-05, de https://www.dremio.com/resources/guides/intro-apache-parquet/

What is Data Governance? (s.f.). Descargado 2024-03-13, de https://cloud.google.com/learn/what-is-data-governance

What is Data Science? | IBM. (2021, septiembre). Descargado 2024-07-03, de https://www.ibm.com/topics/data-science

What is Predicate Pushdown? | Dremio. (s.f.). Descargado 2024-06-08, de https://www.dremio.com/wiki/predicate-pushdown/

Yessad, L., y Labiod, A. (2016, noviembre). Comparative study of data warehouses modeling approaches: Inmon, Kimball and Data Vault. En *2016 International Conference on System Reliability and Science (ICSRS)* (pp. 95–99). Paris, France: IEEE. Descargado 2023-10-26, de http://ieeexplore.ieee.org/document/7815845/ doi: 10.1109/ICSRS.2016.7815845

Zagan, E., y Danubianu, M. (2020, mayo). Data Lake Approaches: A Survey. En *2020 International Conference on Development and Application Systems (DAS)* (pp. 189–193). Descargado 2024-03-23, de https://ieeexplore.ieee.org/document/9108912 doi: 10.1109/DAS49615.2020.9108912

Zhao, Y., Megdiche, I., Ravat, F., y Dang, V.-N. (2021, julio). A Zone-Based Data Lake Architecture for IoT, Small and Big Data. En (pp. 94–102). doi: 10.1145/3472163.3472185

Zouari, F., Kabachi, N., Boukadi, K., y Ghedira Guegan, C. (2021, septiembre). Data Management in the Data Lake: A Systematic Mapping. En *Proceedings of the 25th International Database Engineering & Applications Symposium* (pp. 280–284). New York, NY, USA: Association for Computing Machinery. Descargado 2024-07-02, de https://doi.org/10.1145/3472163.3472173 doi: 10.1145/3472163.3472173

¿Qué es IaaS (infraestructura como servicio)? (s.f.). Descargado 2024-06-01, de https://cloud.google.com/learn/what-is-iaas?hl=es

¿Qué es una PaaS? (s.f.). Descargado 2024-06-01, de https://cloud.google.com/learn/what-is-paas?hl=es

Anexos

Anexo A

Data Lakehouse

En este capítulo se presentan algunos aspectos más en detalle del estado del arte realizado para el concepto de DLH. En la subsección A.1 se describen en profundidad algunos trabajos académicos investigados, mientras que en la subsección A.2 se presentan diversas arquitecturas de DLH de proveedores de cloud estudiadas.

A.1. Definiciones

En esta subsección se describen algunos aspectos en profundidad de los trabajos (Armbrust y cols., 2021), (Harby y Zulkernine, 2022) y (Mazumdar y cols., 2023) que buscan definir un concepto de "Data Lakehouse".

A.1.1. Definición de DLH presentada en (Armbrust y cols., 2021)

El trabajo (Armbrust y cols., 2021) realiza un análisis sobre las distintas arquitecturas de BD, identificando las desventajas que presentan y la necesidad de evolucionar a una nueva arquitectura. En particular, los autores plantean las principales desventajas de la arquitectura predominante en sistemas de BD, formada por un DL y un DW, integrados de forma secuencial (Ver Figura A.1). Algunos de los problemas que mencionan se describen a continuación.

1. Fiabilidad: Se deben realizar continuos procesos ETL para enviar datos entre ambos sistemas (DW y DL), y así mantener la consistencia. A su vez, se tiene el riesgo de que estos procesos pueden insertar errores en los datos.

2. Obsolescencia de los datos: Se considera que los datos dentro del DW están obsoletos, en comparación con los datos en el DL; ya que los datos más actuales que llegan al sistema, demoran días en cargarse al DW.

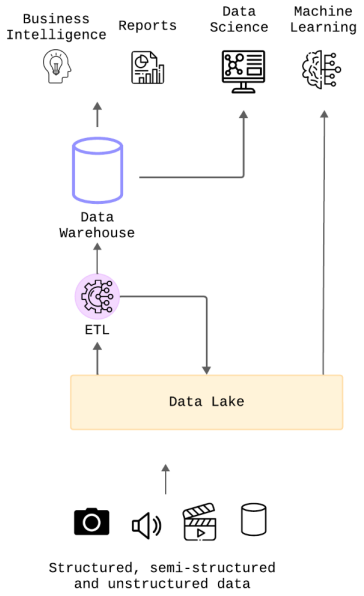

Figura A.1: Arquitectura predominante de BD según los autores de (Armbrust y cols., 2021)

3. Soporte para AA: La mayoría de las herramientas/lenguajes utilizados por esta disciplina no trabajan bien sobre sistemas de DW.

A su vez, estos autores mencionan que debido a la aparición de tecnologías que logran brindar soluciones para tener una buena performance SQL, soporte para herramientas de ML y DS, y una gestión de datos confiable dentro de un DL, la comunidad académica debe adoptar un nuevo sistema que contenga estos aspectos.

Luego, proponen una definición de DLH como "un sistema de gestión de datos, basado en sistemas de almacenamiento de bajo costo que permiten tener un acceso directo a los datos almacenados. A su vez, posee características de gestión de los DBMS tradicionales, como transacciones ACID, control de versiones de datos, auditoría, indexación, almacenamiento en caché y optimización de consultas". Por último, presentan una arquitectura de DLH (Ver Figura A.2) que toma como sistema de almacenamiento un DL y construyen sobre este, distintos componentes que permiten llevar a cabo la gestión de metadatos, dar soporte a herramientas de ML y DS, y brindar acceso a los datos, garantizando buena performance para las consultas SQL. Este trabajo es la primera aproximación a la plataforma Lakehouse de Databricks (*The scope of the lakehouse platform*, s.f.) (Ver Anexo A.2.2).

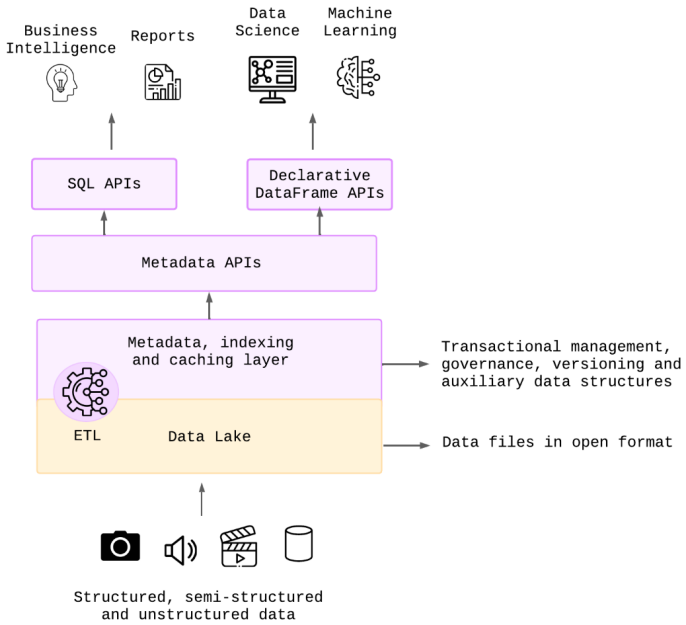

Figura A.2: Arquitectura propuesta en (Armbrust y cols., 2021)

A.1.2. Concepto de DLH presentado en (Harby y Zulkernine, 2022)

Los autores de (Harby y Zulkernine, 2022) no presentan una definición concreta del concepto DLH, pero proponen una arquitectura de DLH y presentan una serie de características que debería tener este sistema, descritas a continuación.

1. Acceso a dato para análisis, reportes y consultas.

2. Ingestion de datos incluyendo stream, real-time y batch.

3. Transformaciones de datos y extracción de metadatos.

4. Los sistemas de almacenamiento utilizados deberían ser escalables y brindar fácil acceso a los datos.

Para proponer la arquitectura de DLH, los autores investigan distintas herramientas que permitan cumplir con los requisitos establecidos anteriormente. Sin embargo, no existe una definición clara de cada componente de la arquitectura (Ver Figura A.3).

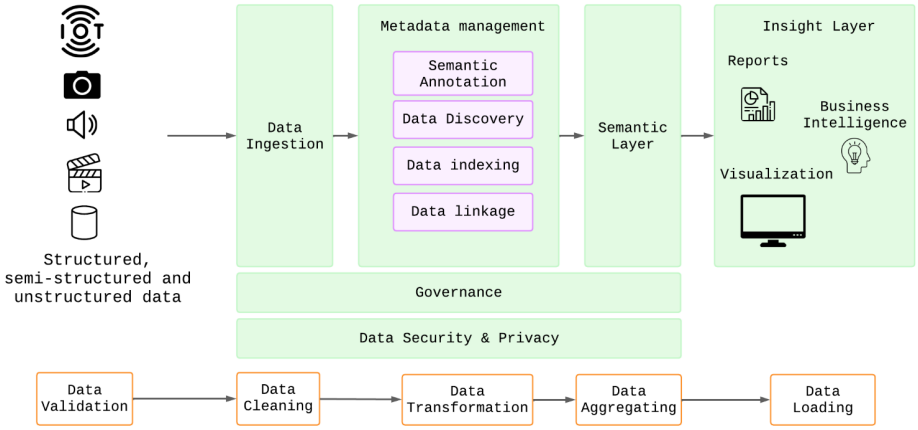

Figura A.3: Arquitectura propuesta por (Harby y Zulkernine, 2022)

A.1.3. Concepto de DLH presentado en (Mazumdar y cols., 2023)

Nuevamente, los autores de (Mazumdar y cols., 2023) no dan una definición concreta para el concepto DLH, pero proponen diversas características y capacidades que debe tener este tipo de sistema.

Características de DLH según los autores de (Mazumdar y cols., 2023).

1. Soporte de transacciones ACID.
2. Almacenamiento de datos en formatos abiertos.
3. Gestión de esquema.
4. Gobierno y calidad de datos.
5. Escalabilidad.

Capacidades de DLH según los autores de (Mazumdar y cols., 2023).

1. Gobernanza y seguridad de datos, debido a la gran cantidad de datos almacenada en estos sistemas y la posible existencia de datos sensibles.
2. Concurrencia de cargas de trabajo.
3. Consultas con baja latencia.
4. Consultas ad-hoc.

5. Gestión de distintas cargas de trabajo (BI, ML, DS).
6. Evolución de esquemas.
7. Transacciones ACID.
8. Separación del almacenamiento y el cómputo.
9. Modelado de datos
10. Calidad de datos
11. Procesos ETL/ELT

Por último, proponen una arquitectura de DLH (Ver Figura A.4) que luego implementan utilizando las tecnologías Apache Iceberg, Amazon S3 y Dremio Sonar.

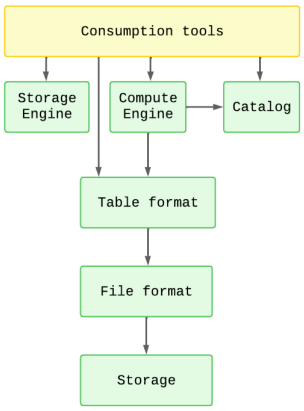

Figura A.4: Arquitectura propuesta por (Mazumdar y cols., 2023)

A continuación se describen los componentes de la arquitectura de DLH propuesta por los autores.

1. **Data storage:** Componente donde se almacenan los datos luego de la ingestion.
2. **Storage engine:** Componente que se encarga de la gestión de datos, incluyendo la compactación, reparticionamiento e indexación de los mismos.
3. **File formats:** Componente que almacena los datos crudos utilizando formatos abiertos.
4. **Table formats:** Componente que funciona como una capa de metadatos sobre los datos almacenados en la capa "File formats".
5. **Catalog:** Componente que permite la búsqueda y consulta de datos en el DLH.

6. **Compute engine:** Componente encargado de realizar procesamiento de datos.

A.2. Propuestas de proveedores

En esta subsección se describen arquitecturas de DLH de diversos proveedores de cloud.

A.2.1. Azure Synapse Analytics

Microsoft proporciona una plataforma de DLH (Ver Figura A.5) denominada *Azure Synapse Analytics*, cuyos componentes se describen a continuación.

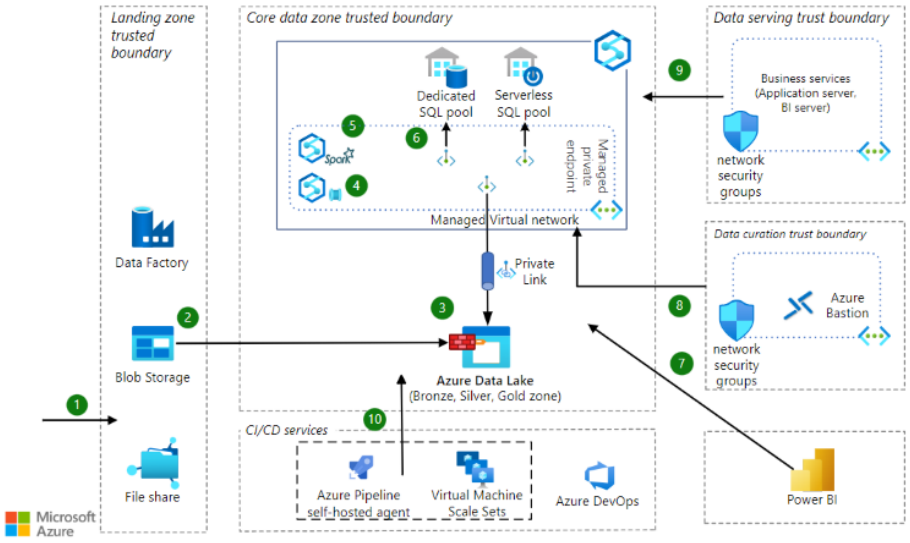

Figura A.5: Arquitectura de la plataforma Azure Synapse Analytics tomada de (Herman-Wu, s.f.)

1. **Landing Zone:** Permite realizar la ingesta de datos. Los datos ingeridos mediante procesos batch se almacenan en *Azure Blob Storage* o como archivo en *Azure Files*. En el caso de los datos de stream, estos se almacenan en *Azure Blob Storage*

2. **Data Zone:** Compuesto por un DL que almacena los datos extraídos de la *Landing Zone* en formato crudo (*raw*). El DL está compuesto por tres zonas, Bronze, Silver y Gold. La zona Bronze es donde están los datos crudos, en la zona Silver se almacenan datos sobre los cuales se aplicaron procesos de limpieza y en la zona Gold se almacenan datos agregados. En este componente se pueden realizar procesamientos de datos utilizando

Spark jobs (*Apache Spark*[1]) o *Notebooks*. Por último, en esta zona se tiene un *Serverless SQL pool* que permite definir esquemas de tablas sobre datos almacenados en el DL. Esta zona puede ser accedida por herramientas de BI, como *Power BI*. A su vez, mediante el uso de *Notebooks* y *Spark jobs*, los DST y expertos de ML pueden realizar análisis sobre los datos.

3. **CI/CD Services:** Permite llevar a cabo una integración y desarrollo continuo de la solución, con etapas de *build, test y deploy*.
4. **Data serving:** Permite el acceso de aplicaciones externas a los datos.
5. **Data curation:** Se encarga de la seguridad del sistema.

A.2.2. Databricks Lakehouse

La empresa Databricks brinda un DLH a través de la plataforma llamada *Databricks Data Intelligence Platform*. Para presentar esta plataforma, primero definen los principales componentes que debe poseer, hoy en día, una plataforma de datos e inteligencia artificial (Ver Figura A.6).

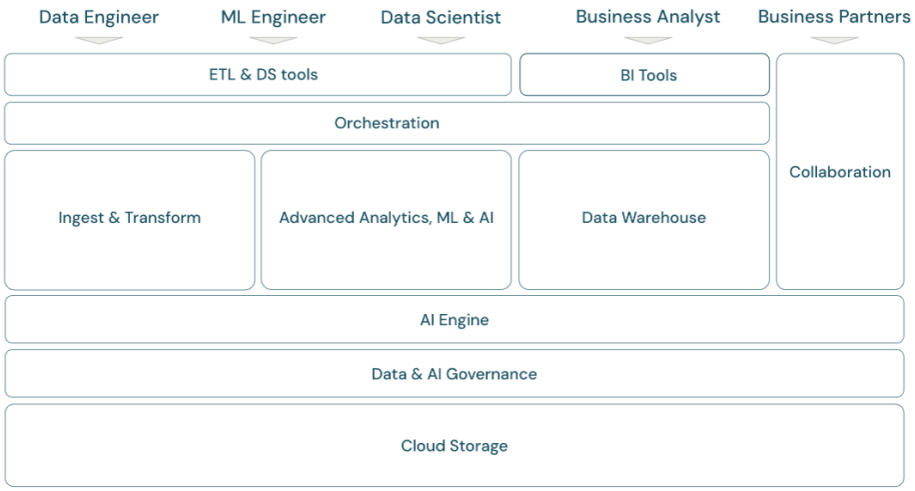

Figura A.6: Componentes de una plataforma de datos e inteligencia artificial según Databricks tomada de (*The scope of the lakehouse platform*, s.f.)

Luego, instancian los componentes con diversas herramientas, como se muestra en la Figura A.7.

A continuación se describen brevemente cada uno de los componentes de esta plataforma.

[1]Motor de procesamiento desarrollado por Apache Software Foundation (Ver Anexo D.1)

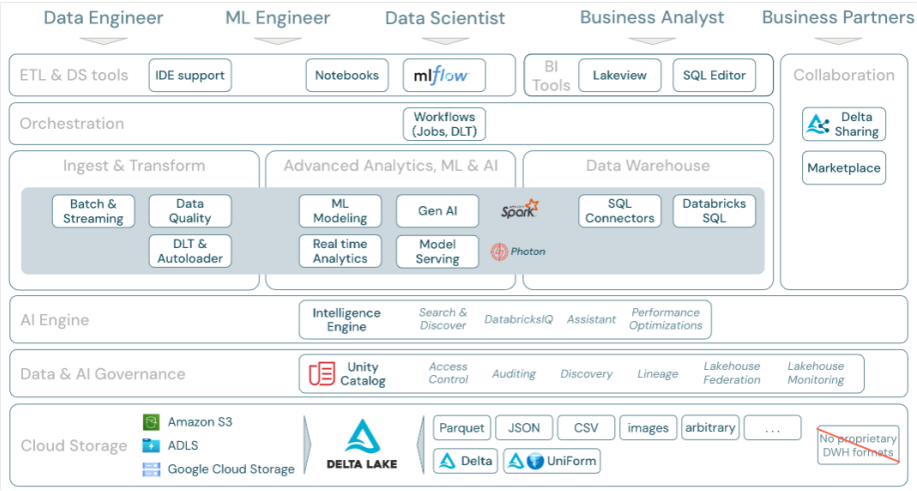

Figura A.7: Plataforma *Databricks Data Intelligence Platform* (*The scope of the lakehouse platform*, s.f.)

1. **Cloud Storage**: Componente de almacenamiento *on-cloud* que permite almacenar distintos tipos de datos.

2. **Data & AI Governance**: Este componente proporciona distintas funcionalidades para llevar a cabo el gobierno de datos de la plataforma.

3. **AI Engine:** Utilizan la herramienta *DatabricksIQ* que permite inferir aspectos semánticos de los datos en el DLH, a partir de la utilización de algoritmos de inteligencia artificial (en adelante, IA).

4. **Ingest & Transform:** Permite realizar procesamiento, de stream y batch, sobre datos en el **Cloud Storage**.

5. **Orchestration:** Permite orquestar los diversos procesamientos que se llevan a cabo en el ciclo de vida de los datos o IA.

6. **Advanced Analytics, ML & AI:** Utiliza el framework *Databricks Mosaic AI* que proporciona diversos algoritmos de IA.

7. **Data Warehouse:** Utiliza *Databricks SQL* para brindar las capacidades de un DW, sobre datos almacenados en el componente *Cloud Storage*.

8. **Herramientas para ETLs, DS y BI:** Proporciona acceso a distintas herramientas utilizadas por DST, Data Engineers y DA.

9. **Collaboration:** Permite compartir datos entre distintas organizaciones de forma segura.

A.2.3. Snowflake

En la Figura A.8 se presenta la plataforma *on-cloud Snowflake*. A continuación se describen los distintos componentes de la misma (*Key Concepts & Architecture | Snowflake Documentation*, s.f.).

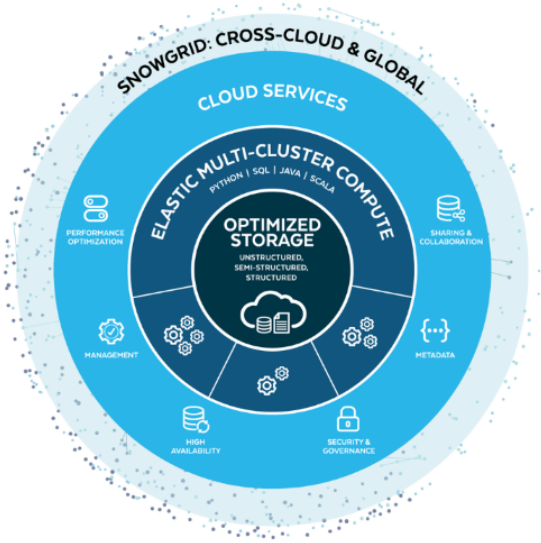

Figura A.8: Arquitectura de la Plataforma *Snowflake* tomada de (*Key Concepts & Architecture | Snowflake Documentation*, s.f.)

1. **Database storage:** Componente de almacenamiento, el cual puede ser accedido por cualquiera de los nodos de procesamiento de la arquitectura. Una vez que los datos se ingieren, estos se modelan para conformar con el formato columnar de la misma. La plataforma permite ingerir datos de stream (Se utiliza la herramienta *Snowpipe*) y batch. A su vez, la plataforma es quien gestiona aspectos de los datos como tamaño, estructura, compresión, organización, metadatos, entre otros. Por último, los datos almacenados solo pueden ser accedidos a través de consultas SQL.

2. **Query processing:** Este componente se encarga de la ejecución de consultas sobre los datos almacenados en el componente **Database storage**. Para ello, procesa consultas utilizando *virtual warehouses*, los cuales son clusters de cómputo que permiten realizar procesamiento masivo en paralelo (*massively parallel processing*, MPP). Los clusters no comparten recursos entre sí, por lo que no afectan la performance de los otros clusters.

3. **Cloud services:** Este componente presenta una colección de servicios que permiten coordinar las distintas actividades y procesos entre los distintos

componentes de la arquitectura. Algunos de los servicios gestionados en este componente son autenticación de usuarios, gestión de infraestructura, gestión de metadatos, y optimización y parseo de consultas.

Esta plataforma tiene la desventaja de que los datos solamente son accedidos mediante el lenguaje SQL, pero las herramientas de AA suelen utilizar otros lenguajes como Python, Scala, R, entre otros.

A.2.4. Lakehouse Dremio

En esta sección se presenta la solución de DLH *open-source* de la plataforma Dremio (Ver Figura A.9).

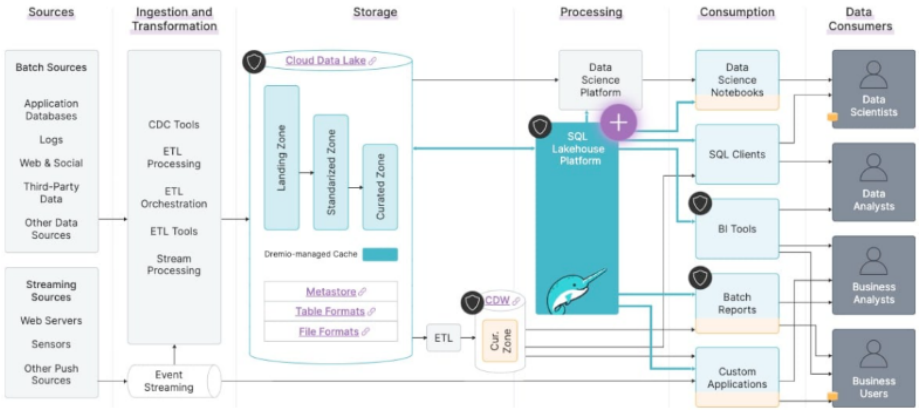

Figura A.9: Arquitectura de la Plataforma *Dremio* tomada de (*The SQL Lakehouse Platform for Enterprise | Dremio*, s.f.)

La solución de DLH propuesta por Dremio es un motor de consultas SQL sobre datos almacenados en un DL. Este motor permite realizar consultas SQL sobre datos almacenados en el DL, sin necesidad de mover los datos a una base de datos relacional como un DW. Para lograr esto, utilizan *Apache Iceberg* y *Apache Arrow* (Ver Anexo D.1).

La principal ventaja de este sistema es que evita replicar datos en ambos sistemas, DL y DW; ya que todo se tiene en un mismo sistema de almacenamiento (DL) y se accede con una misma interfaz. Por otro lado, tiene la desventaja que el acceso a datos es únicamente a partir del lenguaje SQL, pero las herramientas de AA tienden a utilizar otro tipo de lenguajes de programación, como Python, Scala, R, entre otros. Por lo tanto, este motor de consultas solo optimiza el acceso a datos para cargas de trabajo del estilo BI.

A.2.5. BigLake

En esta sección se describe la plataforma propuesta por Google Cloud Platform (en adelante, GCP) (Ver Figura A.10).

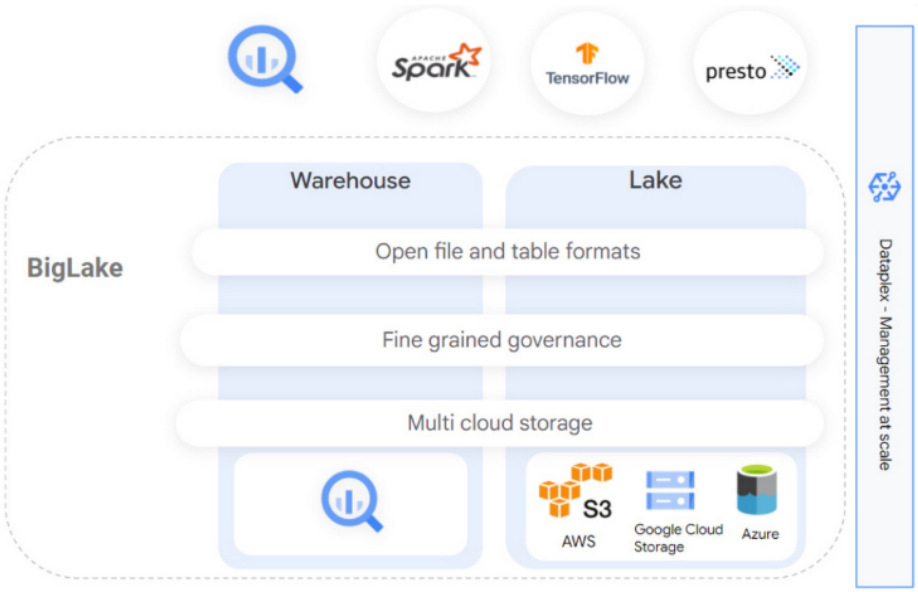

Figura A.10: Arquitectura de la Plataforma *BigLake* tomada de (*Unifying data lakes and data warehouses across clouds with BigLake*, s.f.)

BigLake es un motor de almacenamiento que permite unificar datos en DWs y DLs, de forma que los usuarios puedan analizar los datos, sin preocuparse por el formato o sistema de almacenamiento de los mismos que está por detrás. Para lograr esto, se extiende *BigQuery* para soportar consultas sobre DL y formatos abiertos, como Parquet y ORC (*Unifying data lakes and data warehouses across clouds with BigLake*, s.f.). *BigQuery*, también desarrollado por GCP, es un DW *serverless*[2], el cual está optimizado para manejar consultas SQL complejas sobre grandes volúmenes de datos. Por otro lado, con la extensión de *BigQuery*, también se pueden desarrollar modelos de ML sobre los datos almacenados con *BigQuery ML* (*BigQuery enterprise data warehouse*, s.f.).

A.2.6. Cloudera Data Platform

En esta sección se describe la plataforma *open-source* de *Cloudera* (Ver Figura A.11).

A continuación se describen los componentes de la plataforma (Cloudera y cols., s.f.).

[2]el proveedor de la nube administra y aproviciona los recursos

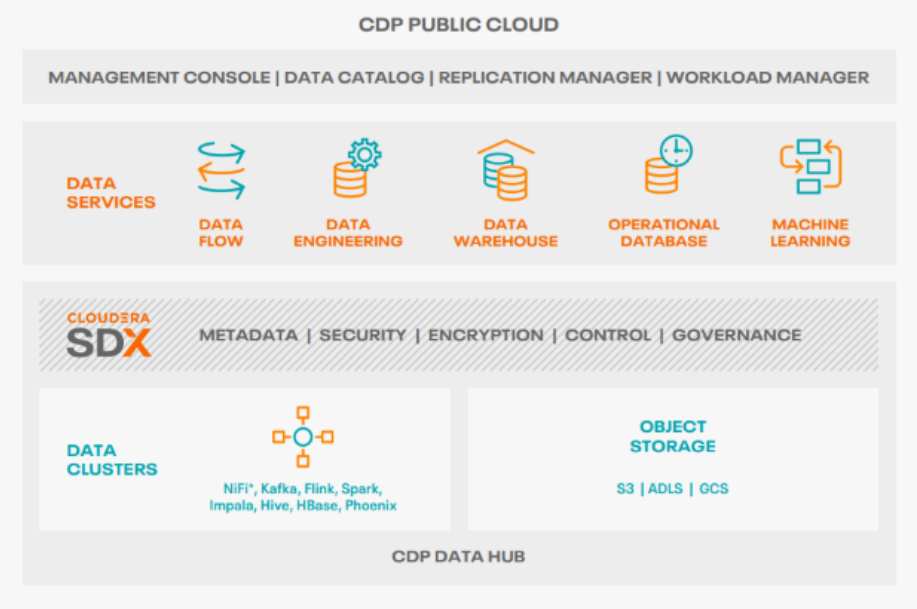

Figura A.11: Arquitectura de *Cloudera Data Platform* tomada de (Cloudera y cols., s.f.)

1. **Shared Data Experience (SDX):** Combina aspectos de seguridad, gobierno, linaje y gestión de datos a través de un catálogo de datos y metadatos.

2. **Data Hub:** Compuesto por un componente de almacenamiento (Object Storage) y un componente de procesamiento (Data Clusters). Permite desplegar clusters analíticos IaaS (Infraestructura como Servicios [3]) durante todo el ciclo de vida de los datos.

3. **Data Services:** Permite desplegar en la nube aplicaciones anlíticas en contenedores. Similar al Data Hub, pero en este caso se tienen clusters PaaS (Plataforma como Servcios[4]).

   a) **Cloudera Data Engineering (CDE):** Servicio en la nube para ingeniería de datos, el cual utiliza Apache Spark y Apache Airflow para desarrollar, orquestar y monitorear procesamientos de datos.

   b) **Cloudera Data Warehouse (CDW):** Utiliza Apache Iceberg, para el formato de tablas, y Apache Impala (Ver Anexo D.1) para el motor

[3]Servicio de infraestructura en la nube, bajo demanda, como almacenamiento, redes, cómputo y virtualización (*¿Qué es IaaS (infraestructura como servicio)?*, s.f.)

[4]Servicio de plataforma en la nube, que no solo ofrece infraestructura a demanda, sino que también ofrece servicios para crear, ejecutar y gestionar aplicaciones (servidores, sistemas operativos, middelware, entre otros) (*¿Qué es una PaaS?*, s.f.)

transaccional del DW.

   c) **Cloudera Machine Learning (CML):** Utiliza contenedores para llevar a cabo tareas de ingeniería de datos y ML. Da soporte a lenguajes de programación como R y Python.

4. **Data Catalog:** Catálogo de datos almacenados en la plataforma que provee funcionalidades como data profiling, linaje y seguridad de los datos.

5. **Management Console:** Componente que permite gestionar el sistema.

Anexo B

Consultas analíticas

Las consultas analíticas buscan obtener un conjunto de columnas de un dataset (*projection*), con alguna condición de filtro (*predicate*) (Ver Consulta B.1) (*What Is Apache Parquet? | Dremio*, s.f.).

Listing B.1: Consulta analítica general

```
SELECT ... -> projection
FROM ...
WHERE ... -> predicate
```

En el caso de los archivos orientados a columnas, los motores de búsqueda pueden utilizar los metadatos almacenados en estos formatos de archivo (Ver Sección 3.7.1) y la información en la consulta (*predicate* y *projection*) para omitir la lectura de columnas que no son parte del resultado final (*What Is Apache Parquet? | Dremio*, s.f.). A modo de ejemplo, si la consulta es de la forma "SELECT col1, col2 FROM dataset" donde "dataset" tiene 3 columnas *col1*, *col2* y *col3*, y se está considerando un archivo *Parquet*, al leer los datos en los *row groups* correspondientes, se puede evitar leer la información de los *data pages* en el *column chunk* de la columna "col3". Notar que en el caso de un almacenamiento orientado a filas, para poder retornar las columnas solicitadas se deben recorrer todas las filas y todas las columnas, incluyendo los datos de "col3".

Por otro lado, si se tiene una consulta con un predicado, se pueden utilizar los metadatos para filtrar *row groups*. Considerar ahora la siguiente consulta "SELECT col1, col2 FROM dataset WHERE col1 $<$ 5". Debido al almacenamiento de estadísticas de los valores en las columnas, como mínimos y máximos, se puede evitar leer los *row groups* que almacenan datos donde el valor mínimo es mayor a 5 (Berk, 2022) (Ver Figura B.1). En particular, este formato permite implementar la técnica *Predicate Pushdown*, la cual permite aplicar filtros sobre los datos en la capa de almacenamiento, durante la lectura de datos. Luego, en vez de leer todos los datos y luego aplicar los filtros, se aplican los filtros durante la lectura de datos, disminuyendo la cantidad de datos leídos de disco y el tiempo de ejecución de las consultas (*What is Predicate Pushdown? | Dremio*,

s.f.).

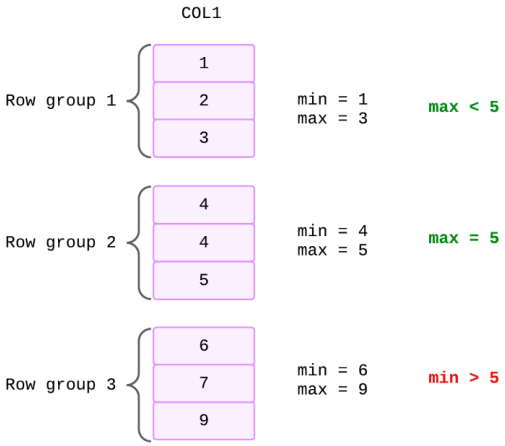

Figura B.1: Ejemplo de filtrado de *row groups*

Anexo C

Formatos de tabla

A continuación se describen los formatos de tabla *Apache Iceberg*. *Apache Hudi* y *Delta table*.

C.1. Apache Iceberg

Apache Iceberg define un formato de tabla sobre un conjunto de archivos de datos (*data files*) en formato *Parquet*, *Avro* u *ORC* (Ver Figura C.1). El estado de estas tablas se mantiene en archivos de metadatos (*metadata files*), los cuales almacenan información como el esquema de la tabla, configuraciones de particiones de los datos, snapshots del contenido de la tabla (Representa el estado de la tabla en un momento determinado en el tiempo), entre otros (*Spec - Apache Iceberg*, s.f.).

Los *manifest files* hacen un seguimiento de los *data files* en un *snapshot*. Estos archivos contienen una fila por cada *data file* en la tabla, los datos particionados y sus métricas. Los datos asociados a un *snapshot* se obtienen a partir de la unión de todos los *data files* en un *manifest file*. Los *manifest files* que componen un snapshot se almacenan en un *manifest list file*, los cuales también almacenan metadatos sobre estos archivos (*Spec - Apache Iceberg*, s.f.).

Por otro lado, este formato de tabla presenta un log transaccional para garantizar transacciones ACID sobre los datos. También posee soporte para evolución de esquemas para las operaciones *add*, *drop*, *update* y *rename*. Por último, también permite realizar *time-travel queries* (permiten consultar estados previos de la tabla) y *rollbacks* de la tabla (*Introduction - Apache Iceberg*, s.f.).

Para finalizar, es importante mencionar que este formato de tabla es soportado por los motores de procesamiento/consultas *Apache Spark*, *Trino*, *Apache Flink*, *Apache Hive*, entre otros (*Apache Iceberg - Apache Iceberg*, s.f.).

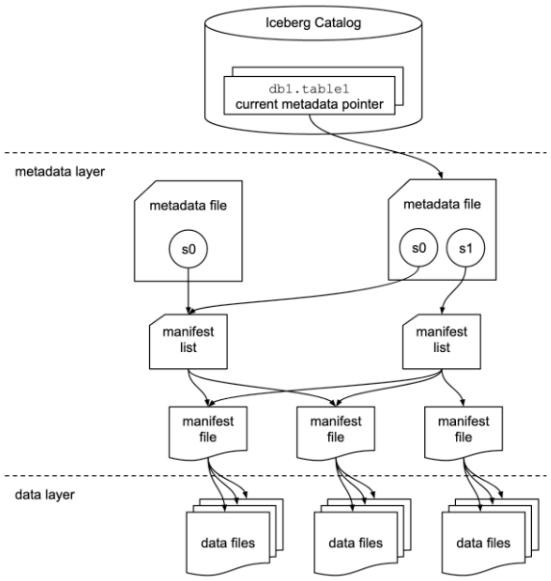

Figura C.1: Formato de tabla de Apache Iceberg tomado de (*Spec - Apache Iceberg*, s.f.).

C.2. Apache Hudi

Apache Hudi (Hadoop Upserts Deletes and Incrementals) provee un formato de tabla el cual utiliza los esquemas de *Avro* para almacenar, gestionar y evolucionar el esquema de las tablas (*Use Cases | Apache Hudi*, s.f.).

En particular, *Hudi* organiza los datos en una estructura de directorio, donde el nombre del directorio raíz (*base path*) se corresponde con el de la tabla *Hudi* (Ver Figura C.2). Debajo del directorio raíz se tiene un directorio de nombre *.hoodie*, el cual almacena el log transaccional y metadatos de la tabla. Por otro lado, *Hudi* provee 2 tipos de tablas, *Copy-on-Write* y *Merge-on-read*, cuyos datos se almacenan en distintos tipos de archivos. En el caso de la tabla *Copy-on-Write*, los datos se almacenan en *Base files*, que son archivos con formato columnar como *Parquet* o *ORC*. En el caso de *Merge-on-read*, los datos se almacenan en *Base files* y *Log files*. Estos último contienen *inserts*, *updates* y *deletes* realizados sobre *Base files*. Las tablas del tipo *Copy-on-write* suelen utilizarse para datasets con baja frecuencia de cambios, mientras que las tablas del tipo *Merge-on-read* se utilizan para datasets cuyos datos cambian frecuentemente (*Apache Hudi*, s.f.).

Sobre este formato de tablas se tienen garantizadas transacciones ACID; ya que se mantiene un log transaccional donde se almacenan las acciones realizadas sobre la tabla (*Use Cases | Apache Hudi*, s.f.). En particular, se almacena la acción realizada, un timestamp de cuándo se realizó la acción y el estado de la

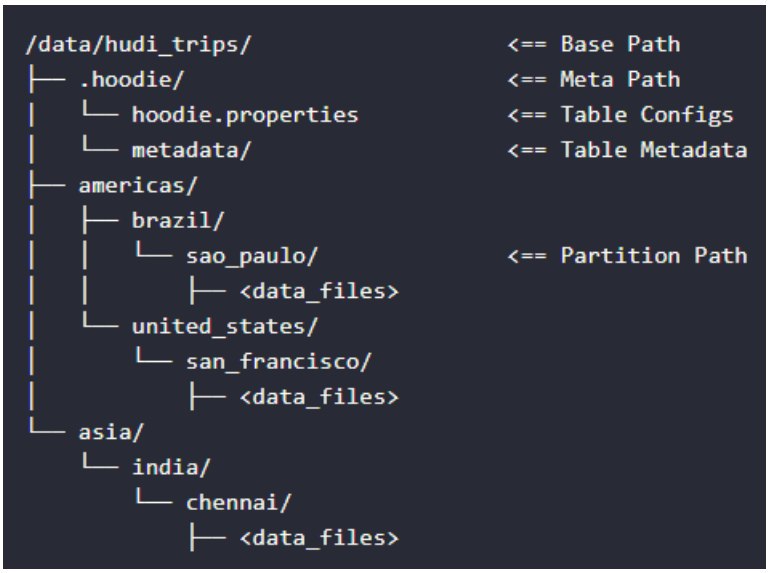

Figura C.2: Formato de tabla de Apache Hudi tomado de (*Apache Hudi*, s.f.).

tabla en ese momento (*Timeline | Apache Hudi*, s.f.).

Por otro lado, también permite realizar *Upserts* (*Update* e *Insert*) y *Deletes* sobre las tablas. Por último, a partir del log transaccional se pueden realizar *time-travel queries* (*Use Cases | Apache Hudi*, s.f.).

Para finalizar, es importante mencionar que este formato de tabla es soportado por los motores de procesamiento/consultas *Apache Spark*, *Apache Flink*, *Amazon Anthena*, entre otros (*Hello from Apache Hudi | Apache Hudi*, s.f.).

C.3. Delta table

Las *Delta tables* son formatos de tabla que definen un log transaccional sobre archivos *Parquet*.

Al igual que *Apache Hudi*, los datos se organizan en un directorio raíz, cuyo nombre es el nombre de la tabla (Ver Figura C.3). Dentro de este directorio se pueden encontrar los sub-directorios *_delta_log* y *_changed_data* (*delta/PROTOCOL.md at master · delta-io/delta*, s.f.).

El sub-directorio *_delta_log* almacena archivos JSON, los cuales registran las acciones realizadas en una transacción, y archivos *checkpoint*, los cuales almacenan una versión compacta del log transaccional hasta el momento (incluye todas las acciones realizadas sobre la versión actual de la tabla) (*delta/PROTOCOL.md*

at master · delta-io/delta, s.f.).

Por otro lado, el sub-directorio *_changed_data* almacena archivos *Parquet* que representan los cambios para la versión de la tabla a la que corresponden (*delta/PROTOCOL.md at master · delta-io/delta*, s.f.).

Dentro del directorio raíz se almacenan los archivos *Parquet* y el log transaccional lleva un registro de cuáles de estos archivos forman parte del estado actual de la tabla. Además de estos archivos, dentro del directorio raíz también se pueden encontrar archivos del tipo *deletion vectors*, los cuales llevan un registro de las filas que fueron eliminadas. Las filas se eliminan de forma lógica, luego, durante la ejecución de consultas se utilizan estos archivos para excluir las filas que fueron eliminadas (*delta/PROTOCOL.md at master · delta-io/delta*, s.f.).

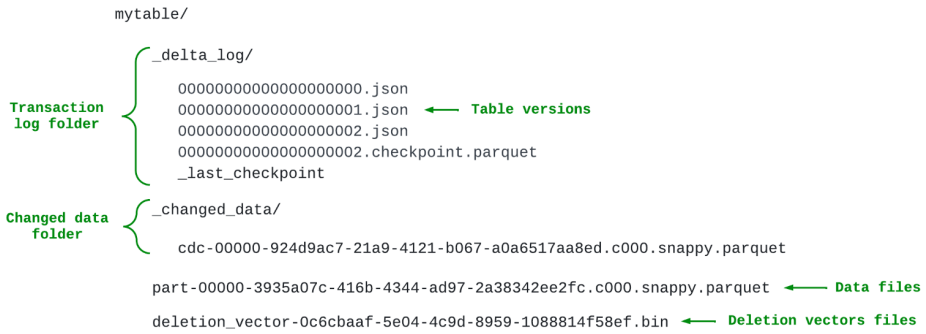

Figura C.3: Formato de tabla *Delta Table*

Este formato de tabla garantiza transacciones ACID, gracias al log transaccional, así como evolución de esquemas y *time-travel queries* (*delta/PROTOCOL.md at master · delta-io/delta*, s.f.).

Para finalizar, es importante mencionar que este formato de tabla es soportado por los motores de procesamiento/consultas *Apache Spark*, *Trino*, *Amazon Anthena*, *Google BigQuery*, entre otros (*Home*, s.f.).

Anexo D

Tecnologías

D.1. Apache Software Foundation

La Apache Software Foundation (ASF) es una organización sin fines de lucro que desarolla proyectos de software *open-source*. En particular, esta organización ha realizado grandes contribuciones a las tecnologías de BD, desarrollando herramientas para el almacenamiento, procesamiento y análisis de datos. A continuación se describen algunos de estas tecnologías, y en las Secciones D.2 y D.3 se profundiza en las tecnologías Apache Hadoop y Apache Spark, las cuales fueron utilizadas para el desarrollo de la arquitectura.

1. **Apache Hadoop:** Framework para realizar procesamiento distribuido de grandes conjuntos de datos en clusters de computadoras (*Apache Hadoop*, s.f.).
2. **Apache Impala:** Motor de consultas SQL para el procesamiento en paralelo de grandes volúmenes de datos, almacenados en un cluster de Apache Hadoop(*Impala*, s.f.)
3. **Apache Iceberg:** Es un formato abierto de tabla que permite proporcionar capacidades de DW a datos almacenados en un DL (*Apache Iceberg | Build an open data lakehouse architecture*, s.f.)
4. **Apache Arrow:** Plataforma de desarrollo que especifica un formato columnar en memoria, optimizado para operaciones analíticas en CPUs y GPUs (*Apache Arrow*, s.f.).
5. **Apache Spark:** Motor de procesamiento multi-lenguaje que permite ejecutar trabajos de ingeniería de datos, aprendizaje automático y ciencia de datos, en una única computadora o en un cluster de computadoras (*Apache Spark™ - Unified Engine for large-scale data analytics*, s.f.).
6. **Apache Hudi:** Plataforma de Data Lake que proporciona funcionalidades de bases de datos relacionales y DW sobre datos en un DL (*Use Cases | Apache Hudi*, s.f.).

D.2. Apache Hadoop

Apache Hadoop (en adelante, AH) es un framework que permite realizar procesamiento distribuido de grandes conjuntos de datos en clusters de computadoras.

Está formado por 3 componentes, Hadoop Distributed File System (en adelante, HDFS), Yarn y MapReduce. Para lograr el procesamiento de grandes datasets, AH utiliza clusters donde se procesan los datos en paralelo. Para ello, los datos se almacenan en HDFS, un sistema de archivos que permite almacenar grandes datasets a lo largo de nodos en un cluster. Luego, los datos se envían al componente MapReduce que realiza procesamiento en paralelo de los datos. Por último, Yarn es quien se encarga de la gestión de recursos dentro de un cluster, planificando y distribuyendo las tareas (*Apache Hadoop: What is it and how can you use it?*, 2019).

D.3. Apache Spark

Apache Spark es un motor de procesamiento para cargas de trabajo de BD. Está basado en el modelo MapReduce de Hadoop, y lo extiende para soportar otros tipos de procesamientos, como procesamiento de stream y consultas interactivas. Este motor se puede utilizar con diversos lenguajes como Python, Scala, Java y R (*Apache Spark*, 2019).

Está compuesto por 2 componentes, *Spark Core* y un conjunto de librerías/módulos para el uso de aprendizaje automático (MLlib), procesamiento de stream (Spark Streaming), manipulación de datos estructurados (Spark SQL) y procesamiento de grafos (GraphX). El componente *Spark Core* es el motor de procesamiento que permite realizar la ejecución de tareas, utilizando *Resilient Distributed Dataset (RDD)* como estructura de datos básica (*Apache Spark*, 2019). Los RDD son colecciones de datos que se logran distribuir a través de diversos nodos en un cluster, permitiendo realizar un procesamiento en paralelo de grandes datasets.

Para la implementación de las transformaciones de datos dentro de la arquitectura se utilizaron *DataFrames* de PySpark. Estas estructuras de datos organizan los datos en columnas que pueden ser de distintos tipos. A su vez, los *DataFrames* son abstracciones construidas sobre RDDs (*Tutorial: Load and transform data using Apache Spark DataFrames*, s.f.).

Anexo E

Experimentación

En este anexo se detallan algunos aspectos de la implementación del prototipo de la arquitectura propuesta.

E.1. Implementación de la arquitectura

En esta sección se describen en detalle las transformaciones llevadas a cabo en las zonas Trusted y Refined.

E.1.1. Trusted Zone

A continuación se describen las transformaciones llevadas a cabo en la Trusted Zone, describiendo primero las transformaciones comunes a todos los datasets, y luego las transformaciones específicas a cada uno de ellos.

Transformaciones comunes

1. Homogeneización de datos faltantes: Algunos datasets utilizan distintos términos para identificar datos faltantes (Por ejemplo, "S/N", "Sin Dato", "N/A", entre otros), por lo que se reemplazan los distintos términos por "Sin Dato". A su vez, en el caso de los campos númericos, se reemplaza "Sin Dato"por 0.

2. Se corrigen los valores de los campos **departamentos** y **localidades** utilizando diccionarios con los valores correctos para cada campo. Para realizar esta corrección se utiliza la distancia de *levenshtein* para obtener el departamento/localidad más similar al dato.

datos_estudiantes_año

1. Se homogeneiza el campo **grado**, eliminando las diversas representaciones de un mismo grado (Por ejemplo, 1°, 1^a y 1).

2. Se corrige la consistencia entre el campo **subsistema** y **ciclo**, siguiendo la lógica descrita en la Tabla E.1.

Condición	**Subsistema**
El valor en la columna "ciclo" contiene alguno de los textos "bachillerato" o "ciclo básico"	DGES
El valor en la columna "ciclo" contiene alguno de los textos "articulación" o "educación media"	DGETP
El valor en la columna "ciclo" contiene alguno de los textos "inicial" o "primaria"	DGEIP

Tabla E.1: Corrección de valores de la columna **subsistema** según los valores de la columna **ciclo**

CEIP y CES

1. Para los valores del campo **localidad** con el texto "zona rural (seleccionar paraje)" se reemplaza el valor en **localidad** por el valor en el campo **barrio_paraje**.

2. Se corrige el campo *nro_puerta*, tomando la primera secuencia de como máximo 5 dígitos en la celda. Se concatena el campo **calle** y **nro_puerta** con el delimitador " ", generando un nuevo campo **direccion**. Se eliminan las columnas **calle** y **nro_puerta**.

3. Para aquellos valores de la columna **teléfono** que contienen más de un teléfono, se selecciona uno de ellos y se elimina el resto. Para ello, primero se reemplazan los caracteres ";" y "/"por "-" en CEIP, y ".", "/" por "-" en CES. Luego, se selecciona el primer teléfono en este campo, haciendo un split según el caracter "-" y obteniendo el primer valor.

E.1.2. Refined Zone

En esta subsección se describe en detalle la creación de las tablas de dimensión y hechos.

1. Las tablas de dimensión Sexo, Ubicación, Zona, Subsistema y SubsistemaEstudiantes se crean al tomar las columnas correspondientes de los datasets, quedándose con los valores distintos de las mismas, y luego agregando columnas con identificadores para cada nivel de la jerarquía. Por ejemplo, para la dimensión Subsistema se toman los valores distintos de la columna "subsistema"del dataset *acceso_plataformas* y se agrega una

columna con identificadores ("id_subsistema"), para cada una de los valores de la columna "subsistema". Se define como clave primaria de la tabla la columna de identificadores del nivel más bajo de la jerarquía. Por otro lado, en el caso de la dimensión SubsistemaEstudiantes se hace un renombre de los valores del nivel *ciclo*, según los valores establecidos para este nivel en el modelo multidimensional (Ver Sección 6.3.1).

2. La dimensión Tiempo contiene una única columna que se crea tomando los valores distintos de la columna "anio"del dataset *acceso_plataformas*, pero no se agrega una columna con identificadores; ya que los propios años funcionan como identificadores de la tabla de dimensión.

3. La columna "departamento"de la dimensión Departamento se crea a partir de una lista de nombres correctos de departamentos de Uruguay. Luego, se incorpora una columna de identificadores "id_departamento.[a] la tabla de la dimensión, que también es la clave primaria de la tabla.

4. La tabla de hechos "Acceso a plataformas"se crea haciendo el join del dataset *acceso_plataformas* con las tablas Tiempo, Sexo, Subsistema y Zona, por la columna correspondiente al nivel más bajo de la jerarquía de cada tabla. Este join se hace para obtener las columnas de identificadores de estas columnas. Luego, se eliminan todas las columnas en la tabla resultante que no son identificadores de los niveles más bajos de las jerarquías de las tablas de dimensión o métricas.

5. La columna "horario"de la tabla de dimensión Horarios se obtiene a partir de los valores distintos de la columna "horario"del dataset *centros_educativos*. Luego, se crean los valores para la columna correspondiente al nivel "tipo" de la dimensión (Ver Tabla E.2 con la correspondencia de valores). Por último, se agregan columnas con identificadores para cada columna, "id_horario" e "id_tipo", donde "id_horario" es la clave primaria de la tabla.

6. La tabla de hechos "Centros educativos" se crea haciendo el join del dataset *centros_educativos* con las tablas Horarios, Ubicación y Subsistema, por la columna correspondiente al nivel más bajo de la jerarquía de cada tabla. Luego, se eliminan aquellas columnas que no son identificadores del nivel más bajo de alguna de las jerarquías de las tablas de dimensión. Por último, se agrega una columna correspondiente a la métrica "Cantidad de centros", denominada "cantidad_centros"que tiene el valor 1 para todas las filas.

E.1.3. Almacenamiento de metadatos

En esta subsección se presentan ejemplos de consultas utilizadas para el almacenamiento de metadatos.

Condición	Tipo
El valor en la columna “horario” contiene los strings “diurno”, “vespertino”, “matutino” o “diurno extendido”	“Diurno”
El valor en la columna “horario” contiene los strings “nocturno” o “nocturno extendido”	“Nocturno”
El valor en la columna “horario” contiene los strings “tiempo completo”, “doble turno”, “multi turno” o “extendido”	“Completo”
El valor en la columna “horario” contiene el string “Rural”	“Rural”
El valor en la columna “horario” contiene el string “Especial”	“Especial”
El valor en la columna “horario” contiene el string “Sin Dato”	“Sin Dato”

Tabla E.2: Correspondencia entre niveles Horario y Tipo de la dimensión Horarios.

Transformación realizada sobre un conjunto de datos

En la Consulta E.1 se muestra un ejemplo de la consulta CYPHER que guarda los metadatos de una transformación realizada sobre un conjunto de datos.

Listing E.1: Consulta CYPHER para almacenar metadatos del linaje de un dataset

```
MERGE (process:Process {name: 'process_name',
                        description: 'description',
                        date: 'date'})
    -[:EXECUTED_IN]->
(process_zone:Zone {name: 'zone'})
WITH process

MATCH (target:Dataset {name: 'name'})
        -[:STORED_IN]->
        (target_zone:Zone {name: 'zone'})
WITH process, target
MERGE (process)<-[:IS_TARGET]-(target)

MATCH (source:Dataset {name: 'name'})
        -[:STORED_IN]->
        (source_zone:Zone {name: 'zone'})
WITH process, source
MERGE (process)-[:HAS_SOURCE]->(source)
```

En caso que el proceso se genere a partir de varios dataset fuentes o tenga varios datasets como resultado, se deben asociar varios datasets a través de la relación *HAS_SOURCE* o *IS_TARGET*, a la instancia del proceso.

Linaje de un dataset

En la Consulta E.2 se muestra un ejemplo de la consulta CYPHER que permite obtener los metadatos asociados al linaje de un dataset.

Listing E.2: Consulta CYPHER para obtener el linaje de un dataset

```
MATCH (t:Dataset {name: 'dataset_name'})
          -[:IS_TARGET]->(p)
          -[:HAS_SOURCE]->(s)
CALL apoc.path.expandConfig(s, {
    relationshipFilter: 'HAS_SOURCE|IS_TARGET',
    minLevel: 1,
    maxLevel: 4
}) YIELD path
RETURN t, path
```

Esta consulta, primero obtiene el proceso que genera el dataset de nombre *dataset_name*, y luego utiliza un *procedure* que busca recursivamente relaciones del tipo *HAS_SOURCE* o *IS_TARGET* entre los nodos, empezando desde el nodo *s*. Notar que los nodos *s* son los nodos fuente del proceso que genera al dataset de nombre *dataset_name*. A su vez, este *procedure* recibe 2 parámetros como entrada, que hacen referencia a la cantidad mínima y máxima de niveles que se quiere obtener.

La cantidad de niveles en un grafo en Neo4j hace referencia al largo del camino entre nodos, comenzando desde un nodo inicial. Notar que un camino desde un nodo en la *Refined Zone* hacia un nodo en la *Raw Zone*, recorriendo las aristas *HAS_SOURCE* o *IS_TARGET*, tiene un largo mínimo de 6 aristas. Como el *procedure* busca recursivamente las relaciones *HAS_SOURCE* o *IS_TARGET* a partir del nodo s (*source* del proceso), se comienza 2 niveles más abajo, por lo que se busca un camino de largo máximo 4.

Por otro lado, si se tiene algún procesamiento realizado sobre un mismo dataset, donde el dataset es el *source* y *target*, se debería aumentar la cantidad máxima de niveles a más de 4. Notar que esto puede ocurrir en caso que se tomen datos ya almacenados en la zona y se ejecuten transformaciones sobre los mismos, sobreescribiendo el dataset original en la zona.

E.2. Gestión de calidad de datos

En esta sección se describen en detalle las tareas llevadas a cabo para la incorporación de la GCD en la arquitectura.

E.2.1. Data Profiling

La primera etapa del procesos de GCD llevada a cabo fue el Data Profiling de los datos, donde se analizó su estructura y se identificaron problemas existentes y la frecuencia de los mismos.

Esta etapa se realizó utilizando la herramienta Power BI de Microsoft. Se utilizó la funcionalidad "data profiling", la cual permite ver para cada columna la cantidad de datos en blanco, la distribución de valores en la columna, cantidad de valores distintos, cantidad de valores únicos, y en caso de ser una columna con valores numéricos se muestran estadísticas de los datos (valor promedio, mínimo, máximo, entre otros) (Ver Figura E.1).

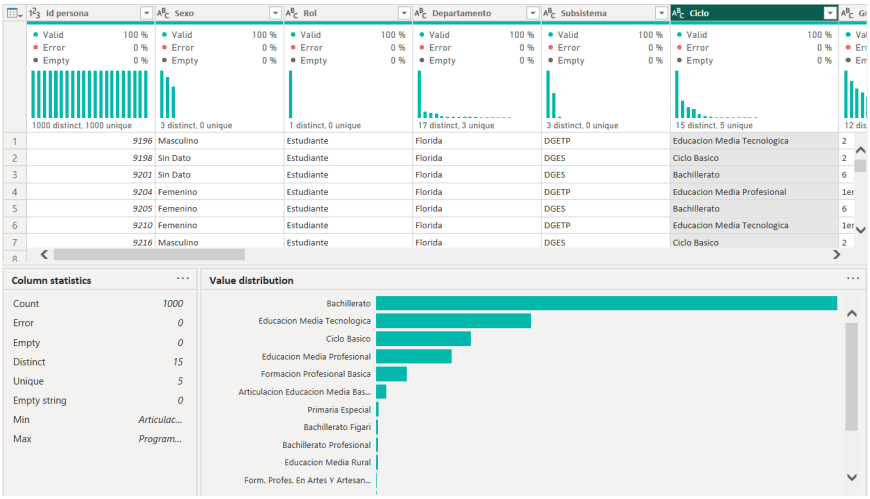

Figura E.1: Data Profiling en Power BI para el dataset *datos_estudiantes_2019*

En las siguientes secciones se presenta el data profiling para los distintos datasets.

Datasets *datos_estudiantes_año* con *año* $\in \{2019, 2020, 2021, 2022\}$

En esta sección se describe la información obtenida a partir del data profiling de los datasets *datos_estudiantes_año* con *año* $\in \{2019, 2020, 2021, 2022\}$.

Estructura de los datasets

A continuación se describe la estructura de los datasets correspondientes a accesos a plataformas en distintos años lectivos. En la Tabla E.3 se pueden ver las distintas columnas, con su descripción y tipo de datos. La descripción de los campos de estos datasets se obtuvo de los metadatos publicados junto a los datasets en el Catálogo Abierto de Datos.

Columna	**Descripción**	**Tipo de dato**
Id Persona	Identificador único para cada estudiante en cada año lectivo.	String
Sexo	Sexo de nacimiento del estudiante	String
Rol	Rol "Estudiante"	String
Departamento	Departamento donde está ubicado el centro educativo en que el estudiante se encuentra matriculado en cada año lectivo	String
Subsistema	Subsistema de ANEP al cual pertenece el estudiante en cada año lectivo	String
Ciclo	Ciclo educativo dentro de un subsistema de ANEP para cada año lectivo.	String
Grado	Grado educativo de los estudiantes dentro del ciclo.	String
Zona	Indica si una escuela pública se encuentra en un área rural o urbana.	String
Contexto	Registra la división del total de escuelas públicas en 5 grupos de igual cantidad de modo que el Quintil 1 agrupa al 20 % de las escuelas de contexto más vulnerable y el Quintil 5 al 20 % de las de contexto menos vulnerable. Esta clasificación se hace por separado para los conjuntos de escuelas urbanas y rurales.	String
Año lectivo	Año lectivo en el que fueron reportados los datos por algún subsistema educativo.	Integer
Cantidad de días ingreso a CREA	Cantidad de días diferentes en que el usuario registró al menos un evento de cualquier tipo en la plataforma CREA.	Integer
Cantidad de entregas de tareas en CREA	Total de tareas enviadas por un usuario en la plataforma CREA.	Integer

Cantidad de comentarios posteados en CREA	Total de comentarios posteados en la plataforma CREA.	Integer
Cantidad de acciones totales en CREA	Total de acciones realizadas por un usuario en la plataforma CREA.	Integer
Cantidad de días de ingreso a Matific	Cantidad de días diferentes en que el usuario registró al menos un evento de cualquier tipo en la plataforma Matific.	Integer
Cantidad de episodios finalizados en Matific	Corresponde al total de episodios finalizados por el estudiante en la plataforma Matifica.	Integer
Cantidad de días de ingreso a PAM	Cantidad de días diferentes en que el usuario registró al menos un evento de cualquier tipo en la plataforma PAM.	Integer
Cantidad de Actividades finalizadas en PAM	Corresponde a la cantidad de actividades finalizadas en la plataforma PAM.	Integer
Cantidad de días de ingreso a Biblioteca	Corresponde a la cantidad de días diferentes en que el usuario registró al menos un evento de cualquier tipo en la plataforma Biblioteca.	Integer
Cantidad de préstamos en Biblioteca	Corresponde al total de préstamos realizados en la plataforma Biblioteca.	Integer

Tabla E.3: Estructura de los datasets *datos_estudiantes_año* con *año* $\in \{2019, 2020, 2021, 2022\}$

Problemas de calidad:

1. Existen datos con el valor “Sin Dato” y “Sin Datos”.
2. La columna ”Grado”tiene diferentes representaciones para el mismo valor, como por ejemplo, “1” y “1ero.” o “4” y “4°”.
3. Hay columnas numéricas con el valor “NA”.

Dataset *CEIP*

En esta sección se describe la información obtenida a partir del data profiling del dataset *CEIP*.

Estructura del dataset:
A continuación se describe la estructura del dataset CEIP. En la Tabla E.4 se pueden ver las distintas columnas, con su descripción y tipo de datos. En este caso, en el Catálogo Abierto de Datos no se tenían disponibles metadatos con descripción de las columnas.

Columna	**Descripción**	**Tipo de dato**
OBJECTID	No se tiene descripción.	String
Rule	No se tiene descripción.	String
Subsistema	Subsistema "DGEIP"	String
Nombre	Nombre del centro educativo	String
NUmero	No se tiene descripción.	String
Area	Indica si una escuela se encuentra en un área rural o urbana.	String
Tipo_de_Educacion	Indica si el centro se corresponde con un centro de educación primaria o inicial.	String
Turno	Indica el tipo de turno (Por ejemplo: "Diurno", "Vespertino", entre otros).	String
Direccion	Indica la calle del centro.	String
NroPuerta	Indica el número de puerta del centro.	String
Telefono	Indica el teléfono del centro.	String
Mail	Indica el correo electrónico del centro.	String
Departamento	Indica el departamento del centro.	String
Localidad	Indica la localidad del centro.	String
Paraje_o_barrio	Indica el paraje en caso de que el centro se encuentre en un área rural, o el barrio en caso que el centro se encuentre en un área urbana.	String
Comparte	No tiene descripción.	String
Categoría	Indica el tipo del centro (Por ejemplo: "De práctica", "De tiempo completo", entre otros).	String
UNOfrece	No tiene descripción.	String
MatrIcula	Cantidad de estudiantes en el centro.	Integer
Alumnos_por_Docente	Cantidad de alumnos por docente.	Integer
Cant_Aulas	Cantidad de aulas en el centro	Integer
Grupos_por_nivel	Cantidad de grupos por nivel en el centro	Integer
Total_docentes	Cantidad de docentes en el centro	Integer
Foto	No tiene descripción	String.
Novedades	No tiene descripción	String.
TMGrupo_Einicial	No tiene descripción	String.
RUEE	Identificador único para cada centro	Integer.
PNOfrece2	No tiene descripción.	String
X	Latitud de la ubicación geográfica del centro.	Integer
Y	Longitud de la ubicación geográfica del centro.	Integer

rule1	No tiene descripción.	String

Tabla E.4: Estructura del dataset *CEIP*

Problemas de calidad:

1. Existen datos con el valor "S/N" y "s/n".
2. Las columnas "OBJECTID", "Rule", "Total_docentes", "Foto", "Novedades" y "TMGrupo_Einicial" tienen solo valores *null*.
3. Hay datos en blanco.
4. La columna "Telefono" utiliza distintas representaciones para los números de teléfonos y en algunos casos almacena más de un valor (Por ejemplo: *21234567*, *2-134-45-67* y *21234567;28910111*).
5. La columna "NroPuerta" tiene diferentes representaciones de números de puerta (Por ejemplo, 1234, 5678-fq, 9101-bis) y valores "S/N".

Dataset *CES*

En esta sección se describe la información obtenida a partir del data profiling del dataset *CES*.

Estructura del dataset:
A continuación se describe la estructura del dataset CES. En la Tabla E.5 se pueden ver las distintas columnas, con su descripción y tipo de datos. Nuevamente, en el Catálogo Abierto de Datos no se tenían disponibles metadatos con descripción de las columnas.

Columna	**Descripción**	**Tipo de dato**
OBJECTID	No se tiene descripción.	String
Ruee_Calculado	Identificador único para cada centro	Integer.
Rule_Calculado	No se tiene descripción.	String
Nombre	Nombre del centro educativo	String
Calle	Indica la calle del centro.	String
Nro_de_puerta	Indica el número de puerta del centro.	String
Nro_de_centro	No se tiene descripción.	String
Telefono	Indica el teléfono del centro.	String

Mail	Indica el correo electrónico del centro.	String
Departamento	Indica el departamento del centro.	String
Localidad	Indica la localidad del centro.	String
DEPEND	No se tiene descripción.	Integer
DEPENDID	No se tiene descripción.	Integer
OFERTA_CICLOBASICO	Indica con 1 si el centro ofrece Ciclo Básico y con 0 si no.	Integer
OFERTA_BACHILLERGRAL	Indica con 1 si el centro ofrece Bachillerato y con 0 si no.	Integer
GRADO_ANNO	Indica los grados que ofrece el centro	Integer.
BACHIILER_DIVNOMBRE	Indica la diversificación ofrecida por el centro	String
Turno	No se tiene descripción	Integer
PLAN	Plan Educativo ofrecido por el centro	Integer
HORARIO	Tipo de horarios ofrecidos por el centro ("Diurno", "Diurno extendido", "Nocturno", "Nocturno extendido").	String
MODALIDAD	Modalidad ofrecida por el centro ("Anual" y "Semestral").	String
NOMBRELIC	Nombre del liceo	String
MATRICULA	Cantidad de estudiantes en el centro.	Integer
MATRICULA_PRIVADO	Cantidad de estudiantes en centros privados.	Integer
TAM_MEDIO_GRUPO	Tamaño medio de grupos.	Integer
TAM_MEDIO_GRUPO_PRIVADO	Tamaño medio de grupos en centros privados.	Integer
REPETICION	No se tiene descripción	String
REPETICION_PRIVADO	No se tiene descripción	String
DESERCION	No se tiene descripción	String
DESERCION_PRIVADO	No se tiene descripción	String
CANT_AULAS	No se tiene descripción	Integer
CANT__AULAS__PRIVADAS	No se tiene descripción	Integer
Paraje	Indica el paraje en caso de que el centro se encuentre en un área rural.	String

DescPlan	Enlace a la página web donde se describe el plan ofrecido por el centro	String.
RUEE_RAD	No se tiene descripción	String.
X	Latitud de la ubicación geográfica del centro.	Integer
Y	Longitud de la ubicación geográfica del centro.	Integer
RUEE_RAD_Long	No se tiene descripción	String.

Tabla E.5: Estructura del dataset *CES*

Problemas de calidad

1. La columna "BACHIILER_DIVNOMBRE" tiene varias representaciones para el mismo valor (Por ejemplo: "DIV.HUMANÍSTICO" y "HUMANÍSTICO").

2. Las columnas "OBJECTID", "Mail", "MATRICULA", "MATRICULA_PRIVADO ", "TAM_MEDIO_GRUPO", "TAM_MEDIO_GRUPO_PRIVADO", REPETICION", REPETICION_PRIVADO", "DESERCION", "DESERCION_PRIVADO", ÇANT_AULAS", "CANT__AULAS__PRIVADAS" y "Paraje" solo tienen valores *null*.

3. La columna "Telefono" tienen varias representaciones de los números de teléfono (Por ejemplo: *12345678/12345678*, *1234.56.78 - 1234.56.78*, *12345678-12345678*).

4. Hay diversas columnas con datos en blanco.

5. La columna "Nro_de_Puerta" tiene diferentes representaciones de números de puerta (Por ejemplo, 1234 y 1234/5678) y valores "S/N".

E.2.2. Modelo de calidad de datos

A partir del data profiling de los datos se definió el modelo de calidad de datos (Ver Tabla E.6).

Dimensión	Factor	Métrica

Completitud	Densidad	**ID:** D1F1M1
		Nombre: Datos faltantes (Sin Dato)
		Descripción: Proporción de celdas con el valor “Sin Dato” o “Sin Datos”
		Granularidad: Entity
		Unidades: [0,...,1]
		Método:
		`def sin_datos_metric(ds):`
		`  cant_filas = ds.count()`
		`  cant_columnas = len(ds.columns)`
		`  cant = 0`
		`  for column in ds.columns:`
		`  cant += ds.filter((col(column) == 'Sin Dato') |`
		`  (col(column) == 'Sin Datos')).count()`
		`  return = cant/(cant_filas*cant_columnas)`

		ID: D1F1M2
		Nombre: Datos faltantes (S/N)
		Descripción: Proporción de celdas con el valor “S/N”
		Granularidad: Entity
		Unidades: [0,...,1]
		Método:
		`def sn_datos_metric(ds):`
		`  cant_filas = ds.count()`
		`  cant_columnas = len(ds.columns)`
		`  cant = 0`
		`  for column in ds.columns:`
		`  cant += ds.filter((col(column) == 'S/N')).count()`
		`  return = cant/(cant_filas*cant_columnas)`

		ID: D1F1M3
		Nombre: Datos faltantes (NA)
		Descripción: Proporción de celdas con el valor “NA”
		Granularidad: Entity
		Unidades: [0,...,1]
		Método:
		`def na_datos_metric(ds):`
		`  cant_filas = ds.count()`
		`  cant_columnas = len(ds.columns)`
		`  cant = 0`
		`  for column in ds.columns:`
		`  cant += ds.filter((col(column) == 'NA') |`
		`  (col(column) == 'NA')).count()`
		`  return = cant/(cant_filas*cant_columnas)`

		ID: D1F1M4
		Nombre: Datos en blanco
		Descripción: Proporción de celdas en blanco
		Granularidad: Entity
		Unidades:: [0,...,1]
		Método:
		`def blank_datos_metric(ds):`
		`  cant_filas = ds.count()`
		`  cant_columnas = len(ds.columns)`
		`  cant = 0`
		`  for column in ds.columns:`
		`  cant += ds.filter((col(column) == '')).count()`
		`  return = cant/(cant_filas*cant_columnas)`

Exactitud	Exactitud sintáctica	**ID:** D2F1M1
		Nombre:: Sintaxis de los números de puerta
		Descripción Proporción de números de puerta que tienen entre 1 a 5 dígitos numéricos
		Granularidad: Attribute
		Unidades: [0,...,1]
		Método:
		`def nro_puerta_metric(ds):`
		`  pattern_nro_puerta = r'\d{1,5}'`
		`  cant_filas = ds.count()`
		`  cant_numero_puerta_ok = ds.filter(`
		`  col("nro_puerta")`
		`  .rlike(pattern_nro_puerta)`
		`  ).count()`
		`  return (cant_numero_puerta_ok/cant_filas)`

		ID: D2F1M2
		Nombre: Sintaxis de los números de teléfono
		Descripción: Proporción de celdas que tienen 8 dígitos y comienza con el número 4 o 2
		Granularidad:: Attribute
		Unidades: [0,...,1]
		Método:
		`def telefono_metric(ds):`
		`  pattern_telefono = r'^[24]\d{7}$'`
		`  cant_filas = ds.count()`
		`  cant_numero_puerta_ok = ds.filter(col("telefono")`
		`  .rlike(pattern_telefono))`
		`  .count()`
		`  return (cant_numero_puerta_ok/cant_filas)`

		ID: D2F1M3
		Nombre: Sintaxis de “grado”.
		Descripción: Proporción de celdas en la columna “grado” que tienen un valor numérico en el conjunto $\{x \in \mathbb{N} : 1 \leq x \leq 9\}$.
		Granularidad: Attribute
		Unidades: [0,...,1]
		Método:
		`def grado_metric():`
		`  pattern_grado = r'^[1-9]$'`
		`  cant_filas = df.count()`
		`  cant_numero_puerta_ok = df.filter(`
		`  col("grado")`
		`  .rlike(pattern_grado)`
		`  ).count()`
		`  return cant_numero_puerta_ok/cant_filas`

	Exactitud semántica	**ID:** D2F2M1
		Nombre: Semántica de “departamento”
		Descripción Proporción de celdas que tienen un valor correcto de Departamento.
		Granularidad: Attribute
		Unidades: [0,...,1]
		Método:
		`def deptos_metric(ds):`
		`  cant_filas = df.count()`
		`  df_deptos = get_deptos_dictionary()`
		`  deptos_ok = df.filter(`
		`  col("departamento").isin(df_deptos)`
		`  )`
		`  return deptos_ok.count() / cant_filas`

		ID: D2F2M2
		Nombre: Semántica de “subsistema”
		Descripción: Proporción de celdas que tienen un valor correcto de Subsistema.
		Granularidad: Attribute
		Unidades: [0,...,1]
		Método:
		`def subsistema_metric(ds):`
		`  cant_filas = df.count()`
		`  df_subsistema = get_subsistema_dictionary()`
		`  subsistema_ok = df.filter(`
		`  col("subsistema")`
		`  .isin(df_subsistema)`
		`  )`
		`  return subsistema_ok.count() / cant_filas`

Unicidad	No duplicación	**ID:** D3F1M1
		Nombre: Unicidad de estudiantes.
		Descripción: Proporción de personas NO duplicadas.
		Granularidad: Entity.
		Unidades: 0,1
		Método:
		`def unicidad_estudiante_metric():`
		`  cant_filas = df.count()`
		`  cant_ids_distintos = df.select(`
		`  col('id_persona')`
		`  ).distinct().count()`
		`  if (cant_filas == cant_ids_distintos):`
		`  return 1`
		`  else:`
		`  return 0`

Consistencia	Integridad de Dominio	**ID:** D4F1M1
		Nombre: Consistencia de métricas.
		Descripción: Proporción de celdas en la columna de la métrica con valor mayor o igual a 0.
		Granularidad: Attribute
		Unidades: [0,...,1]
		Método:
		`def metricas_metric(ds, column_name):`
		`  cant_filas = df.count()`
		`  metric_mayor_0 = df.filter(`
		`  col(column_name) > 0`
		`  ).count()`
		`  return cant_matricula_mayor_a_cero/cant_filas`

Tabla E.6: Modelo de calidad de datos

Las métricas D1F1M1, D1F1M2, D1F1M3 y D1F1M4 surgen debido a que existen datos faltantes en los distintos datasets, representados de distintas maneras ("Sin Dato", "Sin Datos", "NA", "S/N", "s/n" y " "). Por lo tanto, se definen 4 métricas asociadas al factor Densidad de la dimensión Completitud, que miden la proporción de celdas con esos valores.

La métrica D2F1M1 surge debido a que los datasets *CES* y *CEIP* contienen una columna que hace referencia a los números de puerta la cual presenta distintas representaciones de los mismos. Por lo tanto, se crea una métrica del factor Exactitud Sintáctica, que mide la proporción de celdas que tienen un valor numérico cuya longitud está entre 1 y 5 dígitos (Se toma el formato definido en el Modelo de Direcciones Geográficas del Uruguay (Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad y de la Información y el Conocimiento (AGESIC), 2012)).

La métrica D2F1M2 surge debido a que los datasets *CES* y *CEIP* tienen columnas que hacen referencia a números telefónicos, donde se pueden ver distintas representaciones de los mismos. Por lo tanto, se crea una métrica del factor Exactitud Sintáctica, que mide la proporción de celdas que tienen un número de teléfono de Uruguay (8 dígitos en total, comenzando con el número 2 o 4).

La métrica D2F1M3 surge debido a que los datasets *datos_estudiantes_año* tienen una columna "grado" que tiene distintos valores, numéricos y de texto, para representar un mismo valor. Por lo tanto, se crea una métrica del factor Exactitud Sintáctica, que mide la proporción de celdas en esta columna cuyo valor es un número en el conjunto $\{x \in \mathbb{N} | 1 \leq x \leq 9\}$ (*Nuevo Plan de Educación Básica Integrada | Administración Nacional de Educación Pública*, s.f.).

La métrica D2F2M1 surge debido a que todos los datasets tienen atributos que hacen referencia a un departamento de Uruguay. Por lo tanto, se define una métrica asociada al factor Exactitud Semántica, que mide la proporción de celdas cuyo valor en esa columna corresponde al nombre de un Departamento.

La métrica D2F2M2 surge debido a que los datasets *datos_estudiantes_año* y *CEIP* tienen un campo llamado "Subsistema"que hace referencia a alguno de los subsistemas de ANEP. Por lo tanto, se crea una métrica del factor Exactitud Semántica, que mide la proporción de celdas cuyo valor se corresponde al nombre de un Subsistema.

La métrica D3F1M1 surge debido a que existe un identificador único para cada estudiante en el dataset *datos_estudiantes_año*. Por lo tanto, se crea una métrica del factor No duplicación, que mide si esta columna tiene valores únicos.

La métrica D4F1M1 surge debido a que los datasets *datos_estudiantes_año* y *CEIP* tienen diversos campos numéricos que hacen referencia a alguna métrica

(Por ejemplo, “matrícula”, “cantidad de días de ingreso a CREA”, entre otros). Por lo tanto, se crea una métrica del factor Integridad de Dominio, que mide la proporción de celdas en estas columnas cuyo valor es mayor o igual a 0.

Por último, a pesar de que los distintos datasets tienen columnas con valores *null*, no se define una métrica de Densidad que mida la proporción de celdas con este valor; ya que estas columnas se descartan en el filtrado inicial de la Landing Zone (Ver Sección 6.4.1).

En la Tabla E.7 se puede observar la instanciación de las métricas de calidad definidas en el modelo.

Métrica	**Parámetros**
D1F1M1 - Datos faltantes (Sin Dato)	*datos_estudiantes_año*
D1F1M2 - Datos faltantes (S/N)	*CEIP*
D1F1M3 - Datos faltantes (NA)	*datos_estudiantes_año*
D1F1M4 - Datos en blanco	*datos_estudiantes_año*
D1F1M4 - Datos en blanco	*CEIP*
D1F1M4 - Datos en blanco	*CES*
D2F1M1 - Sintaxis de los números de puerta	*CEIP*
D2F1M1 - Sintaxis de los números de puerta	*CES*
D2F1M2 - Sintaxis de los números de teléfono	*CEIP*
D2F1M2 - Sintaxis de los números de teléfono	*CES*
D2F1M3 - Sintaxis de “grado”.	*datos_estudiantes_año*
D2F2M1 - Semántica de “departamento”	*CEIP*
D2F2M1 - Semántica de “departamento”	*datos_estudiantes_año*
D2F2M1 - Semántica de “departamento”	*CES*
D2F2M2 - Semántica de “subsistema”	*datos_estudiantes_año*
D2F2M2 - Semántica de “subsistema”	*CEIP*
D3F1M1 - Unicidad de estudiantes.	*datos_estudiantes_año*
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_dias_ingreso_crea’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_entregas_crea’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_comentarios_crea’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_acciones_totales_crea’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_dias_ingreso_matific’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_episodios_fin_matific’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, ’cant_ingreso_pam’
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*,

	'cant_actividades_fin_pam'
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, 'cant_dias_ingreso_biblioteca'
D4F2M1 - Consistencia de métricas.	*datos_estudiantes_año*, 'cant_prestamos_biblioteca'
D4F2M1 - Consistencia de métricas.	*CEIP*, 'cant_aulas'
D4F2M1 - Consistencia de métricas.	*CEIP*, 'matricula'

Tabla E.7: Instanciación de métricas

E.2.3. Medición de la calidad

Antes de realizar la Medición de calidad de los datos se debe definir el Modelo de Calidad, los Métodos de Medición y los Metadatos de Calidad. Los primeros 2 elementos fueron definidos en la sección anterior, mientras que los Metadatos de Calidad a utilizar se definieron en el Capítulo 5.

A continuación se presentan algunos detalles de la implementación de los Metadatos de Calidad y el reporte de la medición, luego de realizada la medición de calidad.

Metadatos de Calidad

Previo a la ejecución de las mediciones, se definieron en la base de datos de metadatos, en Neo4j, las distintas entidades del modelo de metadatos asociadas a calidad (Ver subsección 5.1.2). Para ello se tienen archivos *.csv* donde se definen las dimensiones, factores y métricas. Luego, se tienen scripts que leen de estos archivos y crean las entidades usando consultas con el lenguaje CYPHER.

A continuación se presenta un ejemplo de consulta CYPHER utilizada para la creación de las instancias de métricas (Ver Consulta E.3). Esta consulta espera que las instancias de dimensión y factor ya estén creadas en la base.

Listing E.3: Consulta CYPHER para crear instancias de métricas

```
MATCH (dimension:Dimension {name: 'dimension'})
-[:HAS_DIMENSION]->
(factor: Factor {name: 'name'})
WITH factor, dimension
MERGE (metric: Metric {name:'metric',
        description: 'description',
        granularity: 'granularity',
        method: 'method'})
MERGE (metric)-[:HAS_FACTOR]->(factor)
```

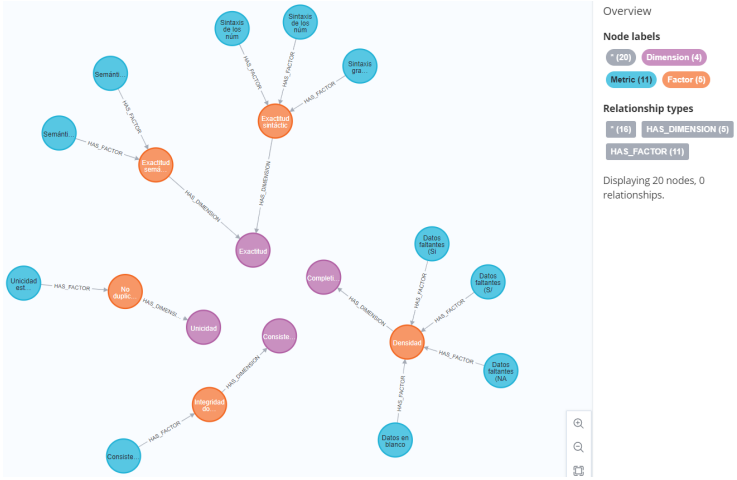

Figura E.2: Nodos asociados a Dimensiones, Factores y Métricas de calidad.

En el ejemplo anterior, los valores *dimension_name*, *factor_name*, *metric_name*, *metric_granularity*, *metric_description* y *metric_method* se obtienen del archivo *.csv* donde están definidas las métricas. En la Figura E.2 se puede ver los nodos generados a partir de esta consulta.

Por último, se presenta una consulta CYPHER que permite obtener las medidas tomadas sobre una tabla y sus columnas (Ver Consulta E.4). En la Figura E.3 se puede ver el resultado de la consulta E.4 para el dataset *CES* en las distintas zonas de la arquitectura.

Listing E.4: Consulta para obtener medidas de calidad asociadas a Columnas y Tablas de un dataset estructurado.

```
    MATCH (zd)<-[:STORED_IN]-(d:Dataset {name: 'name'})
     <-[:BELONGS_TO_TABLE]-(c:Column)
MATCH (c)<-[:COLUMN_MEASURE]-(mc)-[:HAS_METRIC]->(m1)
MATCH (zmc)<-[:MEASURED_IN]-(mc)
MATCH (d)<-[:DATASET_MEASURE]-(md)-[:HAS_METRIC]->(m2)
MATCH (zmd)<-[:MEASURED_IN]-(md)
RETURN zd, d, c, mc, zmc, m1, zmd, m2, md
```

Reporte de medición de calidad

A medida que se procesaron los datos, se realizaron diversas mediciones de calidad según las métricas especificadas en el modelo de calidad de datos. En particular, se realizaron mediciones en la *Landing Zone* y *Trusted Zone* (luego de realizar transformaciones sobre los datos). Luego, en las Tablas E.8, E.9, E.10, E.11, E.12 y E.13 se puede ver una comparación de medidas asociadas a

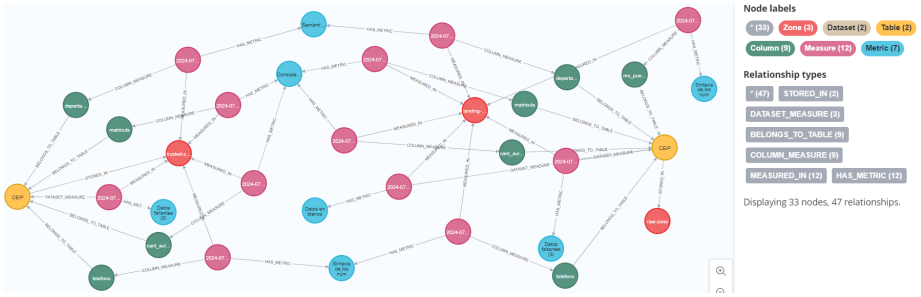

Figura E.3: Metadatos de calidad para el dataset *CES*.

las mismas métricas, sobre los mismos datasets, en distintas zonas. Esto permite ver si las medidas mejoraron luego de realizar transformaciones sobre los datos en la *Trusted Zone*.

Métrica	**1° Medida**	**2° Medida**	**Dato**
Datos en blanco	0.086 - *Landing*		atributos - Entity
Datos faltantes (S/N)	0.007 - *Landing*		atributos - Entity
Semántica de departamentos	0.897 - *Landing*	1.0 - *Trusted*	departamento - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	matrícula - Attribute
Sintaxis de los números de puerta	0.453 - *Landing*		nro_puerta - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_aulas - Attribute
Sintaxis de los números de teléfono	0.69 -*Landing*	0.707 - *Trusted*	teléfono - Attribute
Datos faltantes (Sin Dato)	0.067 - *Trusted*		atributos - Entity

Tabla E.8: Reporte de mediciones para el dataset *CEIP*

Métrica	**1° Medida**	**2° Medida**	**Dato**
Datos en blanco	0.001 - *Landing*		atributos - Entity
Sintaxis de los números de teléfono	0.03 - *Landing*	0.277 - *Trusted*	teléfono - Attribute
Sintaxis de los números de puerta	0.447 - *Landing*		nro_puerta - Attribute
Semántica de departamentos	1.0 - *Landing*	1.0 - *Trusted*	departamento - Attribute

Datos faltantes (Sin Dato)	0.093 - *Trusted*		atributos - Entity

Tabla E.9: Reporte de mediciones para el dataset *CES*

Métrica	**1° Medida**	**2° Medida**	**Dato**
Unicidad de estudiantes	1.0 - *Landing*		atributos - Entity
Datos en blanco	0.0 - *Landing*		atributos - Entity
Datos faltantes (NA)	0.048 - *Landing*		atributos - Entity
Datos faltantes (Sin Dato)	0.052 - *Landing*	0.052 - *Trusted*	atributos - Entity
Semántica de subsistema	1.0 - *Landing*	1.0 - *Trusted*	subsistema - Attribute
Sintaxis de grado	0.413 - *Landing*	0.824 - *Trusted*	grado - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _biblioteca - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_comentarios_crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_ingreso_pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_acciones_totales _crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_prestamos _biblioteca - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_episodio_fin _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_actividades_fin _pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_entregas_crea - Attribute
Semántica de departamento	0.87 - *Landing*	1.0 - *Trusted*	departamento - Attribute

Tabla E.10: Reporte de mediciones para el dataset *datos_estudiantes_2019*

Métrica	**1° Medida**	**2° Medida**	**Dato**
Unicidad de estudiantes	0.0 - *Landing*		atributos - Entity
Datos en blanco	0.0 - *Landing*		atributos - Entity
Datos faltantes (NA)	0.048 - *Landing*		atributos - Entity
Datos faltantes (Sin Dato)	0.049 - *Landing*	0.049 - *Trusted*	atributos - Entity
Semántica de subsistema	1.0 - *Landing*	1.0 - *Trusted*	subsistema - Attribute
Sintaxis de grado	0.413 - *Landing*	0.825 - *Trusted*	grado - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _biblioteca - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_comentarios_crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_ingreso_pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_acciones_totales _crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_prestamos _biblioteca - Attribute
Consistencia de métricas	0.523 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _crea - Attribute
Consistencia de métricas	0.523 - *Landing*	1.0 - *Trusted*	cant_episodio_fin _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_actividades_fin _pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_entregas_crea - Attribute
Semántica de departamento	1.0 - *Landing*	1.0 - *Trusted*	departamento - Attribute

Tabla E.11: Reporte de mediciones para el dataset *datos_estudiantes_2020*

Métrica	**1° Medida**	**2° Medida**	**Dato**
Unicidad de estudiantes	0.0 - *Landing*		atributos - Entity
Datos en blanco	0.0 - *Landing*		atributos - Entity

Datos faltantes (NA)	0.048 - *Landing*		atributos - Entity
Datos faltantes (Sin Dato)	0.049 - *Landing*	0.049 - *Trusted*	atributos - Entity
Semántica de subsistema	1.0 - *Landing*	1.0 - *Trusted*	subsistema - Attribute
Sintaxis de grado	0.419 - *Landing*	0.833 - *Trusted*	grado - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _biblioteca - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_comentarios_crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_ingreso_pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_acciones_totales _crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_prestamos _biblioteca - Attribute
Consistencia de métricas	0.52 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _crea - Attribute
Consistencia de métricas	0.52 - *Landing*	1.0 - *Trusted*	cant_episodio_fin _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_actividades_fin _pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_entregas_crea - Attribute
Semántica de departamento	0.872 - *Landing*	1.0 - *Trusted*	departamento - Attribute

Tabla E.12: Reporte de mediciones para el dataset *datos_estudiantes_2021*

Métrica	**1° Medida**	**2° Medida**	**Dato**
Unicidad de estudiantes	0.0 - *Landing*		atributos - Entity
Datos en blanco	0.0 - *Landing*		atributos - Entity
Datos faltantes (NA)	0.048 - *Landing*		atributos - Entity
Datos faltantes (Sin Dato)	0.049 - *Landing*	0.049 - *Trusted*	atributos - Entity

Semántica de subsistema	1.0 - *Landing*	1.0 - *Trusted*	subsistema - Attribute
Sintaxis de grado	0.427 - *Landing*	0.85 - *Trusted*	grado - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _biblioteca - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_comentarios_crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_ingreso_pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_acciones_totales _crea - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_prestamos _biblioteca - Attribute
Consistencia de métricas	0.521 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_dias_ingreso _crea - Attribute
Consistencia de métricas	0.521 - *Landing*	1.0 - *Trusted*	cant_episodio_fin _matific - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_actividades_fin _pam - Attribute
Consistencia de métricas	1.0 - *Landing*	1.0 - *Trusted*	cant_entregas_crea - Attribute
Semántica de departamento	1.0 - *Landing*	1.0 - *Trusted*	departamento - Attribute

Tabla E.13: Reporte de mediciones para el dataset *datos_estudiantes_2022*

Es claro que las distintas transformaciones realizadas dieron buenos resultados y no generaron nuevos errores en los datos; ya que se puede ver que la segunda medida tomada mejora o se mantiene igual, pero no empeora.

Por último, es importante notar que para el caso de los datasets *datos_estudiates_año*, en la *Landing Zone* se mide la cantidad de celdas con el valor "NA" y "Sin Dato", luego, en la *Trusted Zone* se homogeneizan estos valores, llevando todos los valores faltantes al valor "Sin Dato". A partir de esto, en la medida de la métrica "Datos faltantes (Sin Dato)", en la *Trusted Zone*, se esperaría tener un valor igual a la suma de las medidas obtenidas para las métricas "Datos faltantes (NA)" y "'Datos faltantes (Sin Dato)", en la *Landing Zone*. Sin embargo, el valor de la medida para la métrica "Datos faltantes (Sin Dato)" se mantiene igual, en ambas zonas, debido a que los valores "NA" se encontraban en columnas numéricas y en las transformaciones de la *Trusted Zone* se definió que todas

las columnas numéricas con el valor “Sin Dato” tengan el valor “0”.

E.3. Dashboards

En esta sección se presenta el dashboard creado para la relación “Acceso a plataformas”.

Este dashboard está compuesto por 5 páginas. La primera página muestra un resumen de la información, donde se puede ver la métrica de cantidad de ingresos para cada plataforma según el departamento (Ver Figura E.4), así como según la zona y el departamento (Ver Figura E.5). Por otro lado, se pueden filtrar los datos según el año, por lo que al seleccionar un año distinto, las visualizaciones se adaptan para mostrar los datos asociados a ese año (Ver Figura E.6, donde se seleccionó el año 2022).

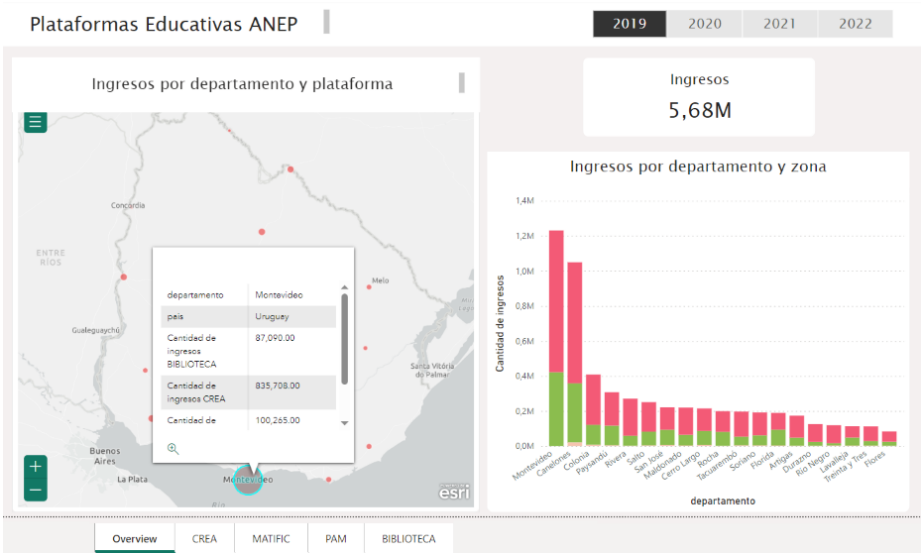

Figura E.4: Tooltip sobre mapa en la página principal del dashboard “Acceso a plataformas” para el año 2019

Luego, las restantes 4 páginas muestran métricas para cada una de las plataformas educativas, CREA, MATIFIC, PAM y BIBLIOTECA. En las Figuras E.7, E.8, E.9 y E.10 se pueden observar los reportes para las distintas plataformas, donde se pueden ver las distintas métricas para cada una de ellas y los filtros de Sexo, Zona y Año que se pueden aplicar (menú superior).

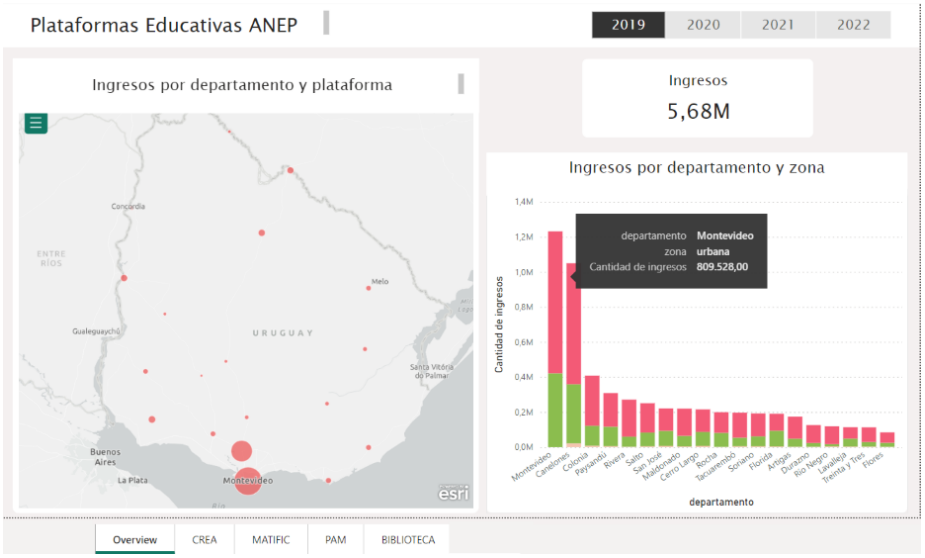

Figura E.5: Tooltip sobre gráfico en la página principal del dashboard "Acceso a plataformas" para el año 2019.

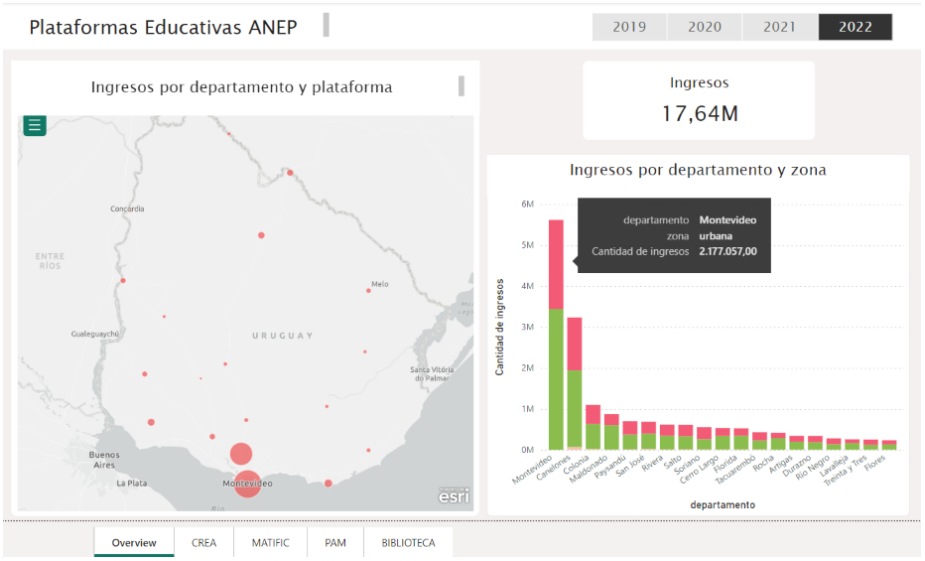

Figura E.6: Datos en dashboard "Acceso a plataformas" para el año 2022.

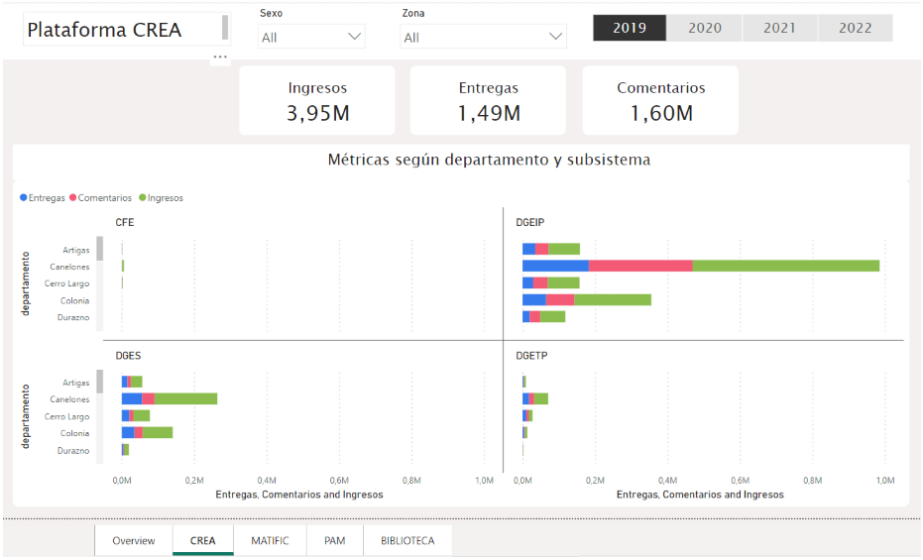

Figura E.7: Página de dashboard "Acceso a plataformas" para la plataforma CREA - año 2019.

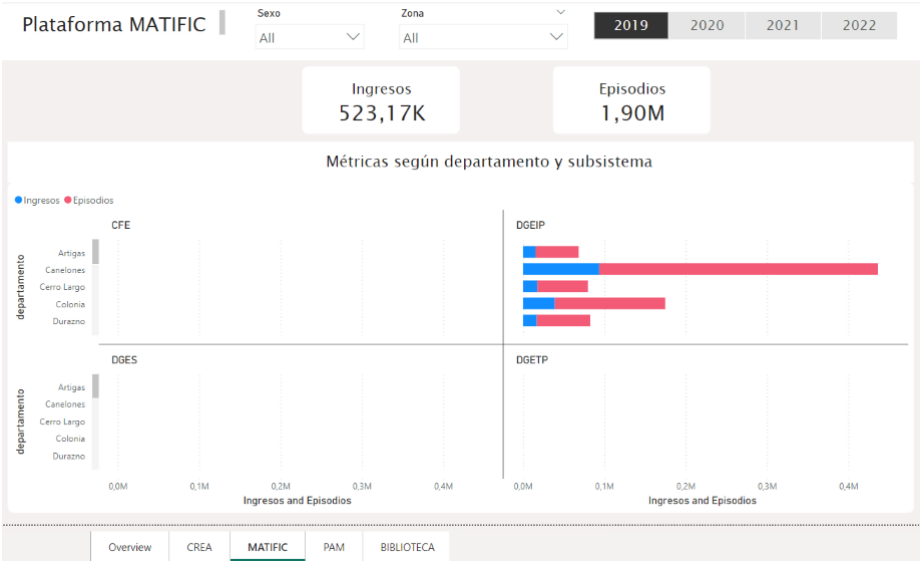

Figura E.8: Página de dashboard "Acceso a plataformas" para la plataforma MATIFIC - año 2019.

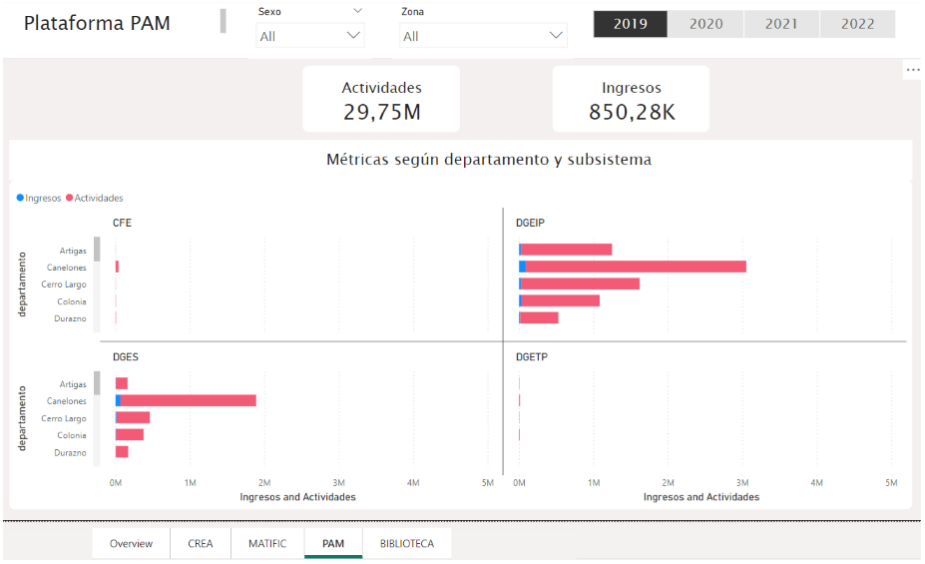

Figura E.9: Página de dashboard "Acceso a plataformas" para la plataforma PAM - año 2019.

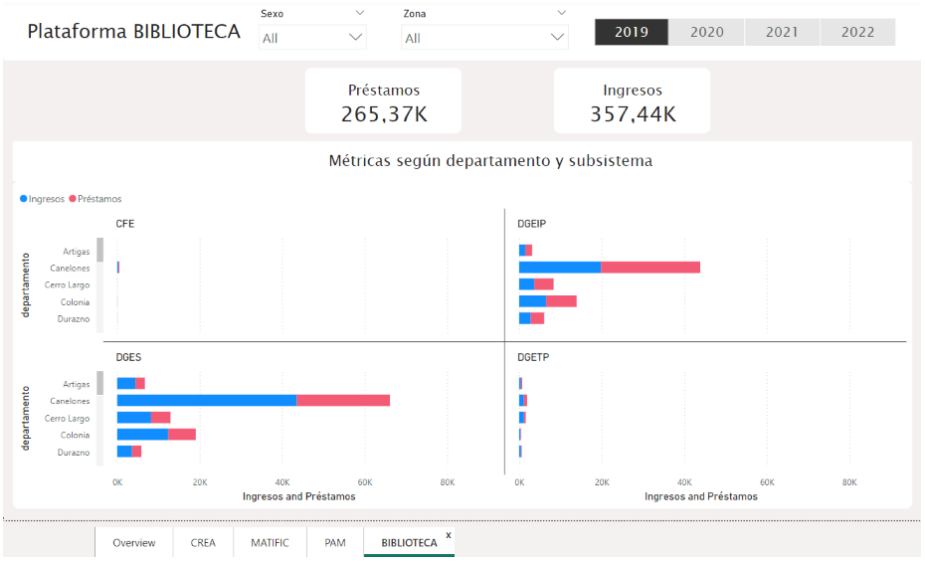

Figura E.10: Página de dashboard "Acceso a plataformas" para la plataforma BIBLIOTECA - año 2019.